Ubuntu Server Guide

Welcome to the Ubuntu Server Guide! This site includes information on using Ubuntu Server for **Ubuntu 20.04 LTS** (Focal Fossa) and later . For an offline version as well as versions for previous releases see below.

Getting Started:

Basic Install	Install Server Edition using the installer
Automated Install	Automate installation using autoinstall
Cloud Images	Run Ubuntu Server Edition on a public cloud provider
Software Containers	Run Ubuntu Server software in dedicated containers

Refining Your Infrastructure:

Networking	Register network devices
Networked Storage	Configure multiple I/O paths for storage
Security	Protect your server and network
Virtualization	Separate your services into virtualized environments
High Availability	Cluster systems redundantly to minimize down-time
Monitoring	Ensure infrastructure availability

Configuring Your Services:

Databases	Install database servers
Backups	Establish your backup plans
Mail Servers	Set up e-mail services
Web Servers	Set up web services

Support

There are a couple of different ways that the Ubuntu Server edition is supported: commercial support and community support. The main commercial support (and development funding) is available from Canonical, Ltd. They supply reasonably- priced support contracts on a per desktop or per-server basis. For more information see the Ubuntu Advantage page.

Community support is also provided by dedicated individuals and companies that wish to make Ubuntu the best distribution possible. Support is provided through multiple mailing lists, IRC channels, forums, blogs, wikis, etc. The large amount of information available can be overwhelming, but a good search engine query can usually provide an answer to your questions. See the Ubuntu Support page for more information.

Improving the Documentation

If you find any errors or have suggestions for improvements to pages, please use the link at the bottom of each topic titled: "Help improve this document in the forum." This link will take you to the Server Discourse forum for the specific page you are viewing. There you can share your comments or let us know about bugs with any page.

PDFs and Previous Releases

Below are links to the previous Ubuntu Server release server guides as well as an offline copy of the current version of this site:

Ubuntu 20.04 LTS (Focal Fossa) and later: PDF Ubuntu 18.04 LTS (Bionic Beaver): Web and PDF

Navigation

Navigation

Level	Path	Navlink
1		Introduction
0		Getting started

Level	Path	Navlink
1		Basic Installation
2	installation	Introduction
2	install/general	Using the installer
2	install/step-by-step	Step by step
2	install/subscription	Attaching your subscription
2	install/reporting-problems	Reporting problems in the installer
2	install/storage	Configuring storage
1		Advanced Installation
2	install/netboot-amd64	amd64 netboot install
2	install/netboot-arm64	arm64 netboot install
2	install/netboot-ppc64el	ppc64el netboot install
2	install/ppc64el	Virtual CDROM and Petitboot on ppc64el
2	install/s390x-zvm	s390x install via z/VM
2	install/s390x-lpar	s390x install via LPAR
1		Automatic Installation
2	install/autoinstall	Introduction
2	install/autoinstall-quickstart	autoinstall quickstart
2	install/autoinstall-quickstart-s390x	autoinstall quickstart on s390x
2	install/autoinstall-reference	autoinstall reference
2	install/autoinstall-schema	autoinstall schema
2	install/vm-autoinstall-on-s390x	z/VM autoinstall on s390x
2	install/lpar-autoinstall-on-s390x	LPAR autoinstall on s390x
1		Cloud Images
2	cloud-images/introduction	Introduction
2	cloud-images/amazon-ec2	Amazon EC2
2	cloud-images/google-cloud-engine	Google Compute Engine
1		ROCK Images
2	rock-images/introduction	Introduction
2	$\operatorname{rock-images/container-customization-with-docker}$	Container Customization with Docker
2	rock-images/multi-node-configuration-with-docker-compose	Multi-node Configuration with Docker-Compose
1		Software
2	package-management	Package Management
2	software/sbom	Software bill of materials
2	upgrade-introduction	Upgrade
2	reporting-bugs	Reporting Bugs
2	kernel-crash-dump	Kernel Crash Dump
0		Services
1		Network
2	network-introduction	Introduction
2	network-configuration	Configuration
2	network-dhcp	DHCP
2	network-ntp	NTP
2	network-dpdk	DPDK
2	openvswitch-dpdk	OpenVswitch-DPDK
1		Multipath
2	device-mapper-multipathing-introduction	Introduction
2	device-mapper-multipathing-configuration	Configuration
2	device-mapper-multipathing-setup	Setup
2	device-mapper-multipathing-usage-debug	Usage & Debug
1		Security
2	security-introduction	Introduction
2	security-users	Users
2	security-smart-cards	Smart Cards
3	security-smart-cards-ssh	SSH
2	security-apparmor	AppArmor
2	security-firewall	Firewall
2	security-certificates	Certificates
2	security-trust-store	CA trust store
2	security-console	Console
1		Virtualization
2	virtualization-introduction	Introduction

Level	Path	Navlink
2	virtualization-qemu	qemu
2	virtualization-libvirt	libvirt
2	virtualization-openstack	openstack
2	virtualization-multipass	multipass
2	virtualization-uvt	uvt
2	virtualization-virt-tools	virt tools
1		Containers
2	containers-lxd	lxd
2	containers-lxc	lxc
1		High Availability
2	ubuntu-ha-introduction	Introduction
2	ubuntu-ha-pacemaker-resource-agents	Pacemaker - Resource Agents
2	ubuntu-ha-pacemaker-fence-agents	Pacemaker - Fence Agents
2	ubuntu-ha-drbd	DRBD
1		Databases
2	databases-introduction	Introduction
2	databases-mysql	Mysql
2	databases-postgresql	PostgreSQL
1		Monitoring
2	logging-monitoring-alerting	LMA
2	monitoring-nagios-munin	Nagios+Munin
2	logwatch	logwatch
1		Backups
2	backups-introduction	Introduction
2	backups-archive-rotation	Archive Rotation
2	backups-bacula	Bacula Chall Carinta
2	backups-sneil-scripts	Shell Scripts
2	tools-rsnapsnot	rsnapsnot Meil Semiesz
1	mail introduction	Introduction
2	mail dovocot	Devecet
2	mail-dovecot	Evim4
$\frac{2}{2}$	mail-postfix	Postfix
2	man-postnx	Web Services
2	web-servers-introduction	Introduction
2	web-servers-apache	Apache
2	proxy-servers-squid	Proxy Servers - Squid
2	lamp-applications	LAMP Applications
2	programming-php	PHP Programming
2	programming-ruby-on-rails	Ruby on Rails Programming
1		OpenLDAP
2	service-ldap-introduction	Introduction
2	service-ldap	Installation
2	service-ldap-access-control	Access Control
2	service-ldap-replication	Replication
2	service-ldap-usage	Simple LDAP User and Group Management
2	service-ldap-with-tls	SSL/TLS
2	service-ldap-backup-restore	Backup and Restore
1		Samba
2	samba-introduction	Introduction
2	samba-active-directory	Joining Active Directory
2	samba-domain-controller	NT4 Domain Controller
2	samba-file-server	File Server
2	samba-print-server	Print Server
2	samba-share-access-control	Share Access Control
2	samba-apparmor-profile	AppArmor Profile
2	samba-openIdap-backend	OpenLDAP Backend
1		Kerberos
2	kerberos-introduction	Introduction
2	service-kerberos	Kerberos Server
2	service-kerberos-principals	Service Principals

Level	Path	Navlink
2	service-kerberos-secondary-kdc	Secondary KDC
2	service-kerberos-workstation-auth	Basic Workstation Authentication
2	service-kerberos-with-openIdap-backend	Kerberos with OpenLDAP backend
1		Network User Authentication with SSSD
2	service-sssd	Introduction
2	service-sssd-ad	Active Directory
2	service-sssd-ldap	LDAP
2	service-sssd-ldap-krb	LDAP and Kerberos
2	service-sssd-troubleshooting	Troubleshooting
1	0	WireGuard VPN
2	wireguard-vpn-introduction	Introduction
2	0	Peer to Site
3	wireguard-vpn-peer2site-introduction	Introduction
3	wireguard-vpn-peer2site-router	On router
3	wireguard-vpn-peer2site-inside	Inside device
2	wireguard-vpn-site2site	Site to Site
2	wireguard-vpn-defaultgw	Default Gateway
2	wireguard-vpn-other-tasks	Other Tasks
2	wireguard-vpn-security	Security Tips
2	wireguard-vpn-troubleshooting	Troubleshooting
1		Other Services
2	service-cups	CUPS
2	service-debuginfod	Debuginfod
3	service-debuginfod-faq	Debuginfod FAQ
2	service-domain-name-service-dns	Domain Name Service (DNS)
2	service-ftp	FTP
2	service-iscsi	iSCSI
2	service-nfs	NFS
2	service-openssh	OpenSSH
2	service-openvpn	OpenVPN
2	service-gitolite	gitolite
2	vpn-clients	VPN Clients
0		Tools
1	tools-byobu	byobu
1	tools-etckeeper	etckeeper
1	tools-munin	munin
1	tools-nagios	nagios
1	pam-motd	pam_motd
1	tools-puppet	Puppet

Redirects

Mapping table

Path	Location
/server/docs/introduction	Introduction
/server/docs/installation-advanced	Installation - Advanced
/server/docs/installation-iscsi	Installation - iSCSI
/server/docs/security-ecryptfs	Security - eCryptfs

Basic installation

This chapter provides an overview of installing Ubuntu Server Edition. There is more detailed documentation on other installer topics.

Preparing to Install

This section explains various aspects to consider before starting the installation.

System requirements

Ubuntu Server Edition provides a common, minimalist base for a variety of server applications, such as file/print services, web hosting, email hosting, etc. This version supports four 64-bit architectures:

- amd64 (Intel/AMD 64-bit)
- arm64 (64-bit ARM)
- ppc64el (POWER8 and POWER9)
- s390x (IBM Z and LinuxONE)

The recommended system requirements are:

- CPU: 1 gigahertz or better
- RAM: 1 gigabyte or more
- Disk: a minimum of 2.5 gigabytes

Server and Desktop Differences

The *Ubuntu Server Edition* and the *Ubuntu Desktop Edition* use the same apt repositories, making it just as easy to install a *server* application on the Desktop Edition as on the Server Edition.

One major difference is that the graphical environment used for the Desktop Edition is not installed for the Server. This includes the graphics server itself, the graphical utilities and applications, and the various user-supporting services needed by desktop users.

Backing Up

Before installing Ubuntu Server Edition you should make sure all data on the system is backed up.

If this is not the first time an operating system has been installed on your computer, it is likely you will need to re-partition your disk to make room for Ubuntu.

Any time you partition your disk, you should be prepared to lose everything on the disk should you make a mistake or something goes wrong during partitioning. The programs used in installation are quite reliable, most have seen years of use, but they also perform destructive actions.

Preparing install media

There are platform specific step-by-step examples for ppc64el installations.

For amd64, download the install image from https://releases.ubuntu.com/20.04/.

There are many ways to boot the installer but the simplest and commonest way is to create a bootable USB stick to boot the system to be installed with (tutorials for other operating systems are also available).

Booting the installer

Plug the USB stick into the system to be installed and start it.

Most computers will automatically boot from USB or DVD, though in some cases this is disabled to improve boot times. If you don't see the boot message and the "Welcome" screen which should appear after it, you will need to set your computer to boot from the install media.

There should be an on-screen message when the computer starts telling you what key to press for settings or a boot menu. Depending on the manufacturer, this could be Escape, F2, F10 or F12. Simply restart your computer and hold down this key until the boot menu appears, then select the drive with the Ubuntu install media.

If you are still having problems, check out the Ubuntu Community documentation on booting from CD/DVD.

After a few moments, the installer will start in its language selection screen.

Willkommen! Bie	nvenue! Welcome! Добро по	жаловать! Welkom!	[Help]
Use UP, DOWN ar	d ENTER keys to select yo English Asturianu Català Hrvatski Nederlands Suomi Français Deutsch Eλληνικά Magyar Latviešu Norsk bokmål Polski Pyccкий Español Українська	ur language.	

welcome_c800 \times 600 3.36 KB

Using the installer

The installer is designed to be easy to use and have sensible defaults so for a first install you can mostly just accept the defaults for the most straightforward install:

- Choose your language
- Update the installer (if offered)
- Select your keyboard layout
- Do not configure networking (the installer attempts to configure wired network interfaces via DHCP, but you can continue without networking if this fails)
- Do not configure a proxy or custom mirror unless you have to in your network
- For storage, leave "use an entire disk" checked, and choose a disk to install to, then select "Done" on the configuration screen and confirm the install
- Enter a username, hostname and password
- Just select Done on the SSH and snap screens
- You will now see log messages as the install is completed
- Select restart when this is complete, and log in using the username and password provided

There is more detailed documentation on all these options.

Using the installer

This document explains how to use the installer in general terms. There is another document that contains documentation for each screen of the installer.

Getting the installer

You can download the server installer for amd64 from https://ubuntu.com/download/server and other architectures from <math>http://cdimage.ubuntu.com/releases/20.04/release/.

Installer images are made (approximately) daily and are available from http://cdimage.ubuntu.com/ubuntu-server/focal/daily-live/current/. These are not tested as extensively as the images from release day, but they contain the latest packages and installer so fewer updates will be required during or after installation.

Installer UI generalities

In general, the installer can be used with the up and down arrow and space or enter keys and a little typing. Tab and shift-tab move the focus down and up respectively. Home / End / Page Up / Page Down can be used to navigate through long lists more quickly in the usual way.

Running the installer over serial

By default, the installer runs on the first virtual terminal, tty1. This is what is displayed on any connected monitor by default. Clearly though, servers do not always have a monitor. Some out-of-band management systems provide a remote virtual terminal, but some times it is necessary to run the installer on the serial port. To do this, the kernel command line needs to have an appropriate console specified on it – a common value is console=ttyS0 but this is not something that can be generically documented.

When running on serial, the installer starts in a basic mode that does using only the ASCII character set and black and white colours. If you are connecting from a terminal emulator such as gnome-terminal that supports unicode and rich colours you can switch to "rich mode" which uses unicode, colours and supports many languages.

Connecting to the installer over SSH

If the only available terminal is very basic, an alternative is to connect via SSH. If the network is up by the time the installer starts, instructions are offered on the initial screen in basic mode. Otherwise, instructions are available from the help menu once networking is configured.

In addition, connecting via SSH is assumed to be capable of displaying all unicode characters, enabling more translations to be used than can be displayed on a virtual terminal.

Help menu

The help menu is always in the top right of the screen. It contains help, both general and for the currently displayed screen, and some general actions.

Switching to a shell prompt

You can switch to a shell at any time by selecting "Enter shell" from the help menu, or pressing Control-Z or F2.

If you are accessing the installer via tty1, you can also access a shell by switching to a different virtual terminal (control-alt-arrow or control-alt-number keys move between virtual terminals).

Global keys

There are some global keys you can press at any time:

Key	Action
ESC	go back
F1	open help menu
Control-Z, F2	switch to shell
Control-L, F3	redraw screen
Control-T, F4	toggle rich mode (colour, unicode) on and off

Using the installer step by step

The installer is designed to be easy to use without constant reference to documentation.

Language selection

Willkommen! Bienvenue!	Welcome! Добро пожаловать! Welkom)! [Help]
USE UP, DOWN and ENTER English (Asturia Català Hrvatsk Nederla Suomi França. Deutsch Eλληνικ Magyar Latvies Norsk k Polski Pycckwi Españo. Україно	keys to select your language. h anu ki ands is h kά šu bokmål й 1 ська	

welcome_c.png 800×600 3.36 KB

This screen selects the language for the installer and the default language for the installed system.

More languages can be displayed if you connect via SSH.

Refresh



refresh.png 800×600 2.91 KB

This screen is shown if there is an update for the installer available. This allows you to get improvements and bug fixes made since release.

If you choose to update, the new version will be downloaded and the installer will restart at the same point of the installation.

Keyboard



keyboard.png 800×600 1.79 KB

Choose the layout and variant of keyboard attached to the system, if any. When running in a virtual terminal, it is possible to guess the layout and variant by answering questions about the keyboard.

Zdev (s390x only)

Zdev setup			
ID	ONLINE	NAMES	
generic-ccw			
0.0.0009		>	
0.0.000c		>	
0.0.000d		>	
0.0.000e		>	
dasd-eckd			
0.0.0190		>	
0.0.0191		>	
0.0.019d		>	
0.0.019e		>	
0.0.0200		> < (close)
0.0.0300		> Enable	·
0.0.0400		> Disabl	e
0.0.0592		>└	

[Continue]

v

[Back

]

This screen is only shown on s390x and allows z-specific configuration of devices.

The list of device can be long. Home / End / Page Up / Page Down can be used to navigate through the list more quickly.

Network



network.png 800×600 2.99 KB

This screen allows the configuration of the network. Ubuntu Server uses netplan to configure networking and the UI of the installer can configure a subset of netplan's capabilities. In particular it can configure DHCP or static addressing, VLANs and bonds.

If networking is present (defined as "at least one interface has a default route") then the installer will install updates from the archive at the end of installation.

Proxy



proxy.png800×600 2.69 KB

The proxy configured on this screen is used for accessing the package repository and the snap store both in the installer environment and in the installed system.

Mirror



mirror.png800 \times 600 1.99 KB

The installer will attempt to use geoip to look up an appropriate default package mirror for your location. If you want or need to use a different mirror, enter its URL here.

Storage



storage_config.png800 \times 600 3.89 KB

Storage configuration is a complicated topic and has its own page for documentation.

Storage configu	ration	[Help]					
FILE SYSTEM SUM	MARY						
MOUNT POINT [/	SIZE TYPE DEVICE T 9.997G new ext4 new part	YPE ition of local disk ▶]					
AVAILABLE DEVIC	ES						
No available	devices						
	Confirm destructi	ve action					
Selecting result in	Continue below will begin th the loss of data on the disk	e installation process and s selected to be formatted.					
You will installat	You will not be able to return to this or a previous screen once the installation has started.						
Are you s	ure you want to continue?						
	[No [Continue]					
	[Done [Reset [Back						

storage_confirm.png800 $\times600$ 3.71 KB

Once the storage configuration is confirmed, the install begins in the background.

Identity



identity.png 800×600 2.43 KB

The default user will be an administrator, able to use sudo (this is why a password is needed, even if SSH public key access is enabled on the next screen).

\mathbf{SSH}

SSH Setup		[Help]
You can choose to in access to your serve	stall the OpenSSH server package to ^.	enable secure remote
[]	Install OpenSSH server	
Import SSH identity:	[No ▼] You can import your SSH keys from	Github or Launchpad.
Import Username:		
[X]	Allow password authentication ove	r SSH
	[Done] [Back]	

$\mathrm{ssh.png800}{\times}600~2.27~\mathrm{KB}$

A default Ubuntu install has no open ports. It is very common to administer servers via SSH so the installer allows it to be installed with the click of a button.

You can import keys for the default user from Github or Launchpad.

If you import a key, then password authentication is disabled by default but it can be re-enabled again if you wish.

Snaps

Fe	eatured Server Snaps	[Help]
Tł pr av	hese are popular snaps ress ENTER to see more vailable.	in server environments. Select or deselect with SPACE, details of the package, publisher and versions	
	<pre>] microk8s] nextcloud] wekan] kata-containers] docker] canonical-livepatch] rocketchat-server] mosquitto] etcd] powershell] stress-ng] sabnzbd] wormhole] aws-cli] google-cloud-sdk] slcli] doctl] doctl] conjure-up] minidlna-escoand] postgresql10] heroku] keepalived] prometheus] juju</pre>	Kubernetes for workstations and appliances Nextcloud Server - A safe home for all your data Open-Source kanban Lightweight virtual machines that seamlessly plug int Docker container runtime Canonical Livepatch Client Group chat server for 100s, installed in seconds. Eclipse Mosquitto MQTT broker Resilient key-value store by CoreOS PowerShell for every system! A tool to load, stress test and benchmark a computer SABnzbd get things from one computer to another, safely Universal Command Line Interface for Amazon Web Servi Command-line interface for Google Cloud Platform prod Python based SoftLayer API Tool. DigitalOcean command line tool Package runtime for conjure-up spells server software with the aim of being fully compliant PostgreSQL is a powerful, open source object-relation CLI client for Heroku High availability VRRP/BFD and load-balancing for Lin The Prometheus monitoring system and time series data Simple, secure and stable devops. Juju keeps complexi	
		[Done] [Back]	

snaps.png 800×600 9.09 KB

If a network connection is enabled, a selection of snaps that are useful in a server environment are presented and can be selected for installation.

Installation logs



install_progress.png800 \times 600 4.62 KB

The final screen of the installer shows the progress of the installer and allows viewing of the full log file. Once the install has completed and security updates installed, the installer waits for confirmation before restarting.



install_done.png 800×600 6.64 KB

Attaching your subscription to Ubuntu

Attaching the Ubuntu Advantage subscription to Ubuntu, brings the enterprise lifecycle, including Linux kernel livepatching, access to FIPS validated packages, and compliance with security profiles such as CIS. This is not required on Ubuntu Pro instances on AWS, Azure or GCP, as these are automatically attached from launch.

NOTE: Not only enterprises can have a subscription, anyone can get a personal subscription for free on up to 3 machines, or 50 if you are an official Ubuntu Community member.

The following instructions explain how to attach your subscription to your Ubuntu systems.

Step 1: Installing the UA client

This step is necessary for Ubuntu Advantage customers or holders of personal subscriptions. If you are an Ubuntu Pro user, your subscription is already attached and you may skip these instructions.

We first need to make sure that we have the latest version of the UA client running:

\$ sudo apt update

\$ sudo apt install ubuntu-advantage-tools

This will ensure that you are running the latest version of the UA client.

Step 2: Attaching your subscription

Once you have checked that you are running the latest version of the UA client, you need to attach the UA token to your UA client to gain access to the Ubuntu Advantage services.

First we need to retrieve our UA token from our Ubuntu Advantage dashboard. To access your dashboard, you need an Ubuntu One account. If you still need to create one, ensure that you use the email address used to create your subscription.

The Ubuntu One account functions as a single-sign-on, so once logged in we can go straight to the Ubuntu Advantage dashboard at ubuntu.com/advantage. Then click on the 'Machines' column in the Your Paid Subscriptions table to reveal your token.

Now we're ready to attach our UA token to the UA client:

```
$ sudo ua attach <your_ua_token>
Enabling default service esm-infra
Updating package lists
ESM Infra enabled
Enabling default service livepatch
Canonical livepatch enabled.
```

This machine is now attached to your account.

Note that Extended Security Maintenance and Livepatch will auto-enable once your token has been attached to your machine.

After attaching the UA client with your token you can also use the UA client to activate most of the Ubuntu Advantage services, including Livepatch, FIPS, and the CIS Benchmark tool.

Reporting problems with the installer

We of course hope that every install with the server installer succeeds. But of course reality doesn't always work that way and there will be failures of various kinds. This section explains the most useful way to report them so that we can fix the bugs causing them, and we'll keep the topic up to date as the installer changes.

The first thing to do is to update your subiquity snap. Not only because we fix issues that cause failures over time but also because we've been working on features to make failure reporting easier.

A failure will result in a crash report being generated which bundles up all the information we need to fully diagnose a failure. These live in /var/crash in the installer environment, and for Ubuntu 19.10 and newer this is persisted to the install media by default (if there is space).

When an error occurs you are presented with a dialog which allows you to upload the report to the error tracker and offers options for continuing. Uploads to the error tracker are non-interactive and anonymous, so they are useful for tracking which kinds of errors are affecting most users, but they do not give us a way to ask you to help diagnose the failure.

You can create a Launchpad bug report, which does let us establish this kind of two way communication, based on the contents of a crash report by using the standard apport-cli tool that is part of Ubuntu. Copy the crash report to another system, run "apport-cli /path/to/report.crash" and follow the prompts.

You can also run apport-cli in the installer environment by switching to a shell but apport won't be able to open a browser to allow you to complete the report so you'll have to type the URL by hand on another machine.

Configuring storage in the server installer

Guided options

Guided storage configuration	[Help]
Configure a guided storage layout, or create a custom one:	
(<u>X</u>) Use an entire disk	
[QEMU NVMe Ctrl_1234 local disk 10.000G ▼]	
[X] Set up this disk as an LVM group	
[] Encrypt the LVM group with LUKS	
Passphrase:	
Confirm passphrase:	
() Custom storage layout	
[Done] [Back]	

storage_guided.png800 $\times 600$ 2.89 KB

Selecting "Use an entire disk" on the Guided storage configuration screen will install Ubuntu onto the selected disk, replacing any partitions or data already there.

You can choose whether or not to set up LVM, and if you do whether or not to encrypt the volume with LUKS. If you encrypt the volume, you need to choose a passphrase that will need to be entered each time the system boots.

If you select "Custom storage layout" no configuration will be applied to the disks.

In either case, the installer moves onto the main storage customization screen.

The main storage screen

Storage configura	tion				[Help]
FILE SYSTEM SUMMAR	۲Y					
MOUNT POINT [/ [/boot	SIZE 8.996G 1.000G	TYPE new ext4 new ext4	DEVICE TY new LVM 1 new parti	/PE Logical volume ition of local disk	•] •]	
AVAILABLE DEVICES						
DEVICE [QEMU NVMe Ctrl_9 unused	5678			TYPE local disk	SIZE 10.000G	▶]
[Create software [Create volume gn	RAID (m roup (LVI	d) ►] M) ►]				
USED DEVICES						
DEVICE [ubuntu–vg (new) ubuntu–lv new	ω, to be	formatted	as ext4,	TYPE LVM volume group mounted at ∕	SIZE 8.996G 8.996G	• 1
[QEMU NVMe Ctrl_ partition 1 net partition 2 net partition 3 net	1234 ມ, bios_ງ ມ, to be ມ, PV of	grub formatted LVM volumo	as ext4, e group ut	local disk mounted at ∕boot puntu–vg	10.000G 1.000M 1.000G 8.997G	•] • •
		[Di [Ri [Bi	one eset ack]]]		

storage_manual.png 800×600 5.65 KB

This screen presents a summary of the current storage configuration. Each device or partition of a device corresponds to a selectable row, and pressing enter or space while a device is selected opens a menu of actions that apply to that device.

Partitions

	DEVICE	TYPE	SIZE	Г		
[QEMU NVMe Ctrl_1234	local disk	10.000G		◀ (close)	
	unused				Info	
					Reformat	•
[QEMU NVMe Ctrl_5678	local disk	10.000G	۰I	Add GPT Partition	
	unused				Format	
					Remove from RAID/LVM	
]	Create software RAID	(md) 🕨]			Use As Boot Device	
Γ	Create volume group	(LVM) 🕨]		L		

To add a partition to a device, select "Add GPT partition" for that device.

Storage c	onfiguration					[Help]
FILE SYST	EM SUMMARY						
MOUNT PI [/ [/boot	DINT SIZE 8.996G 1.000G	TYPE new ext4 new ext4	DEVICE 1 new LVM new part	TYPE logical v ition of	olume local disk •	•] •]	
AVAILABLE	DEVICES						
	Adding	GPT parti	tion to G	EMU NVMe I	Ctrl_5678 —		
Size	e (max 9.998G):	_	_	_	_	_	
	Format:	[ext4		•]			
	Mount:	[/home		•]			
		[[Create Cancel]]			
partiti partiti	on 2 new, to be on 3 new, PV of	formatted LVM volum	as ext4, e group ι	mounted a buntu-vg	at ∕boot	1.000G 8.997G	
		[D [R [B	one eset ack				

add_dialog.png800 $\times600$ 4.35 KB

You can leave size blank to use all the remaining space on the device.

RAID

Stor	age configurat	ion			[Help]
FILE	SYSTEM SUMMAR	RY				1
МО	OUNT POINT	SIZE	TYPE DEVICE	TYPE		I
			Create software RA	ID ("MD") disk		
	Name:	md0				
	RAID Level:	0 (1 (5	striped) mirrored) ◄			
	Devices:	Б 10	_5678	10.000G		
		[]	partition 1 [active ▼] unused partition p partition 2 [active ▼] unused partition p	4.000G of local disk 4.000G of local disk		
	Size:					
			[Create [Cancel]]		
			[Done [Reset [Back			

add_raid.png800 \times 600 3.87 KB

Linux software RAID (RAID stands for "Redundant Array of Inexpensive Disks") can be used to combine several disks into a single device that (usually) is tolerant to any one disk failure.

A software RAID device can be created out of entire disks or unformatted partitions. Select the "Create software RAID (md)" button to open the creation dialog.

The server installer supports creating devices with RAID level 0, 1, 5, 6 or 10. It does not allow customizing other options such as metadata format or RAID10 layout at this time. See the linux RAID documentation for more.

A software RAID device can be formatted and mounted directly, or partitioned into several partitions (or even be used as part of another RAID device or LVM volume group).

\mathbf{LVM}

Storage configuration		[Help]
FILE SYSTEM SUMMARY		*
	Create LVM volume group	
Nan	e: <u>v</u> gO	
Device	s: QEMU NVMe Ctrl_5678 10.000G local disk [] partition 1 4.000G unused partition of local disk [] partition 2 4.000G unused partition of local disk	
Siz	e: -	
] Create encrypted volume	
Passphras		
Confirm passphras	e:	
	[Create] [Cancel]	
	[Back]	

add_lvm.png800 $\times600$ 3.48 KB

LVM (the "Logical Volume Manager") is a system of managing logical volumes, or filesystems, that is much more advanced and flexible than the traditional method of partitioning a disk into one or more segments and formatting that partition with a filesystem. It can be used to combine several disks into one larger pool of storage but it offers advantages even in a single disk system, such as snapshots and easy resizing of logical volumes.

As with RAID, a LVM volume group can be created out of entire disks or unformatted partitions. Select the "Create volume group (LVM)" button to open the creation dialog.

Once a volume group has been created, it can be divided into named logical volumes which can then be formatted and mounted. It generally makes sense to leave some space in the volume group for storage of snapshots and creation of more logical volumes as needed.

The server installer does not supported configuring any of the many, many options LVM supports when creating volume groups and logical volumes.

Selecting boot devices

	DEVICE	TYPE	SIZE			
[QEMU NVMe Ctrl_1234	local disk	10.000G	×	◀ (close)	
	unused				Info	Þ
					Reformat	Þ
[QEMU NVMe Ctrl_5678	local disk	10.000G		Add GPT Partition	Þ
	unused				Format	Þ
					Remove from RAID/LVM	
[Create software RAID	(md) 🕨]			Use As Boot Device	
[]	Create volume group ((LVM) 🕨]				

On all architectures other than s390x, the bootloader needs to be installed to a disk in a way such that the system firmware can find it on boot. By default, the first device to have a partition created on it is selected as a boot device but this can be changed later.

On amd64 and arm64 systems, multiple disks can be selected as boot devices, which means a system can be configured so that it will continue to boot after a failure of any one drive (assuming the root filesystem is placed on a RAID). The bootloader will be installed to each of these drives, and the operating system configured to install new versions of grub to each drive as it is updated.

amd64 systems use grub as the bootloader. amd64 systems can boot in either UEFI or legacy (sometimes called "BIOS") mode (many systems can be configured to boot in either mode) and the bootloader is located completely differently in the two modes.

In legacy mode, the bootloader is read from the first "sector" of a hard drive (exactly which hard drive is up to the system firmware, which can usually be configured in a vendor specific way). The installer will write grub to the start of all disks selected as a boot devices. As Grub does not entirely fit in one sector, a small unformatted partition is needed at the start of the disk, which will automatically be created when a disk is selected as a boot device (a disk with an existing GPT partition table can only be used as a boot device if it has this partition).

In UEFI mode, the bootloader loaded from a "EFI System Partition" (ESP), which is a partition with a particular type GUID. The installer automatically creates an 512MiB ESP on a disk when it is selected as a boot device and will install grub to there (a disk with an existing partition table can only be used as a boot device if it has an ESP – bootloaders for multiple operating systems can be installed into a single ESP). UEFI defines a standard way to configure the way in which the operating system is chosen on boot, and the installer uses this to configure the system to boot the just-installed operating system. One of the ESPs must be mounted at /boot/efi.

Supported arm64 servers boot using UEFI, and are configured the same way as an UEFI-booting amd64 system.

ppc64el systems also load their bootloader (petitboot, a small linux kernel) from a "PReP" partition with a special flag, so in most ways they are similar to a UEFI system. The installer only supports one PReP partition at this time.

Limitations and workarounds

Currently the installer cannot *edit* partition tables. You can use existing partitions or reformat a drive entirely but you cannot, for example, remove a large partition and replace it with two smaller ones.

The installer allows the creation of LVM volume groups and logical volumes and MD raid devices, but does not allow tweaking of the parameters – for example, all logical volumes are linear and all MD raid devics use the default metadata format (1.2).

These limits can both be worked around in the same way: drop to a shell and use the usual shell commands to edit the partition table or create the LV or RAID with desired parameters, and then select these partitions or devices as mount points in the installer. Any changes you make while the installer is running but before altering the storage configuration will reflected in the installer.

The installer cannot yet configure iSCSI mounts, ZFS at all, or btrfs subvolumes.

Netbooting the server installer on amd64

amd64 systems boot in either UEFI or legacy ("BIOS") mode (many systems can be configured to boot in either mode). The precise details depend on the system firmware, but both modes usually support the PXE ("Preboot eXecution Environment") specification, which allows the provisioning of a bootloader over the network.

The process for network booting the live server installer is similar for both modes and goes like this:

- 1. The to-be-installed machine boots, and is directed to network boot.
- 2. The DHCP/bootp server tells the machine its network configuration and where to get the bootloader.
- 3. The machine's firmware downloads the bootloader over tftp and executes it.
- 4. The bootloader downloads configuration, also over tftp, telling it where to download the kernel, ramdisk and kernel command line to use.
- 5. The ramdisk looks at the kernel command line to learn how to configure the network and where to download the server ISO from.
- 6. The ramdisk downloads the ISO and mounts it as a loop device.
- 7. From this point on the install follows the same path as if the ISO was on a local block device.

The difference between UEFI and legacy modes is that in UEFI mode the bootloader is a EFI executable, signed so that is accepted by SecureBoot, and in legacy mode it is PXELINUX. Most DHCP/bootp servers can be configured to serve the right bootloader to a particular machine.

Configuring DHCP/bootp and tftp

There are several implementations of the DHCP/bootp and tftp protocols available. This document will briefly describe how to configure dnsmasq to perform both of these roles.

- 1. Install dnsmasq with "sudo apt install dnsmasq"
- 2. Put something like this in /etc/dnsmasq.conf.d/pxe.conf:

```
interface=<your interface>,lo
bind-interfaces
dhcp-range=<your interface>,192.168.0.100,192.168.0.200
dhcp-boot=pxelinux.0
dhcp-match=set:efi-x86_64,option:client-arch,7
dhcp-boot=tag:efi-x86_64,bootx64.efi
enable-tftp
tftp-root=/srv/tftp
```

(This assumes several things about your network; read man dnsmasq or the default /etc/dnsmasq.conf for lots more options).

 $3.\ restart\ dnsmasq\ with\ sudo\ systemctl\ restart\ dnsmasq.service.$

Serving the bootloaders and configuration.

We need to make this section possible to write sanely

Ideally this would be something like:

```
# apt install cd-boot-images-amd64
# ln -s /usr/share/cd-boot-images-amd64 /srv/tftp/boot-amd64
```

Mode independent set up

1. Download the latest live server ISO for the release you want to install:

wget http://cdimage.ubuntu.com/ubuntu-server/daily-live/current/focal-live-server-amd64.iso

2. Mount it.

```
# mount ubuntu-19.10-live-server-amd64.iso /mnt
```

3. Copy the kernel and initrd from it to where the dnsmasq serves tftp from:

cp /mnt/casper/{vmlinuz,initrd} /srv/tftp/

Setting up the files for UEFI booting

1. Copy the signed shim binary into place:

```
# apt download shim-signed
# dpkg-deb --fsys-tarfile shim-signed*deb | tar x ./usr/lib/shim/shimx64.efi.signed -0 > /srv/tftp/bootx64.efi
```

2. Copy the signed grub binary into place:

```
# apt download grub-efi-amd64-signed
# dpkg-deb --fsys-tarfile grub-efi-amd64-signed*deb | tar x ./usr/lib/grub/x86_64-efi-signed/grubnetx64.efi.signed
0 > /srv/tftp/grubx64.efi
```

3. Grub also needs a font to be available over tftp:

```
# apt download grub-common
# dpkg-deb --fsys-tarfile grub-common*deb | tar x ./usr/share/grub/unicode.pf2 -0 > /srv/tftp/unicode.pf2
```

4. Create /srv/tftp/grub/grub.cfg that contains:

```
set default="0"
set timeout=-1

if loadfont unicode ; then
  set gfxmode=auto
  set locale_dir=$prefix/locale
  set lang=en_US
fi
```

```
terminal_output gfxterm
set menu_color_normal=white/black
set menu_color_highlight=black/light-gray
if background_color 44,0,30; then
  clear
fi
function gfxmode {
        set gfxpayload="${1}"
        if [ "${1}" = "keep" ]; then
                set vt handoff=vt.handoff=7
        else
                set vt handoff=
        fi
}
set linux_gfx_mode=keep
export linux_gfx_mode
menuentry 'Ubuntu 20.04' {
        gfxmode $linux_gfx_mode
        linux /vmlinux $vt_handoff quiet splash
        initrd /initrd
}
```

Setting up the files for legacy boot

1. Download pxelinux.0 and put it into place:

```
# wget http://archive.ubuntu.com/ubuntu/dists/eoan/main/installer-amd64/current/images/netboot/ubuntu-
installer/amd64/pxelinux.0
# mkdir -p /srv/tftp
# mv pxelinux.0 /srv/tftp/
```

2. Make sure to have installed package syslinux-common and then:

cp /usr/lib/syslinux/modules/bios/ldlinux.c32 /srv/tftp/

3. Create /srv/tftp/pxelinux.cfg/default containing:

```
DEFAULT install
LABEL install
KERNEL vmlinuz
INITRD initrd
APPEND root=/dev/ram0 ramdisk_size=1500000 ip=dhcp url=http://cdimage.ubuntu.com/ubuntu-server/daily-
live/current/focal-live-server-amd64.iso
```

As you can see, this downloads the ISO from Ubuntu's servers. You may well want to host it somewhere on your infrastructure and change the url to match.

This configuration is obviously very simple. PXELINUX has many, many options, and you can consult its documentation at https://wiki.syslinux.org/wiki/index.php?title=PXELINUX for more.

Netbooting the Live Server Installer via UEFI PXE on Arm (aarch64, arm64) and x86_64 (amd64)

This document provides the steps needed to install an system via netbooting and subiquity in UEFI mode with Ubuntu 20.04 (or later). The process is applicable to both of the architectures, arm64 and amd64. This process is inpired by this Ubuntu Discourse post for *legacy mode*, which is UEFI's predecessor. Focal (20.04, 20.04.3) and Groovy (20.10) have been tested with the following method.

Configuring TFTP

This article assumes that you have setup your tftp (and/or DHCP/bootp if necessary, depending on your LAN configuration) by following this Ubuntu Discourse post, or you could also consider build your own tftp in this way if your DNS and DHCP is already well configured:

\$ sudo apt install tftpd-hpa

If the installation is successful, check the corresponding TFTP service is active by this command:

\$ systemctl status tftpd-hpa.service

It is expected to show *active (running)* from the output messages. We will also assume your tftp root path is /var/lib/tftpboot in the remaining of this article.

Serving Files

You can skip the whole section of the following manual setup instruction by using this non-official tool. The tool will setup your TFTP server to serve necessary files for netbooting.

Necessary Files

There are several files needed for this process. The following files are needed:

- Ubuntu live server image
 - For arm64 architecture, its image name has a -arm64 suffix. For example, ubuntu-20.04.3-live-server-arm64.iso.
 - For amd64 architecture, its image name has a -amd64 suffix. For example, ubuntu-20.04.3-live-server-amd64.iso.
- grub efi binary (and the corresponding grub.cfg, which is a txt file)
 - For arm64 architecture, it is grubnetaa64.efi.signed.
 - For amd64 architecture, it is grubnetx64.efi.signed.
- initrd extracted from your target Ubuntu live server image (use hwe-initrd instread if you want to boot with HWE kernel) $% \left(\frac{1}{2}\right) =0$
- vmlinuz extracted from your target Ubuntu live server image (use hwe-vmlinuz instead if you want to boot with HWE kernel)

Examples

In the following sections, we will take arm64 image as an example. This means the following files are used:

- Ubuntu 20.04.3 live server image *ubuntu-20.04.3-live-server-arm64.iso* from https://cdimage.ubuntu.com/ ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-arm64.iso
- grub efi binary grubnetaa64.efi.signed from http://ports.ubuntu.com/ubuntu-ports/dists/focal/main/uefi/grub2-arm64/current/grubnetaa64.efi.signed
- initrd extracted from *ubuntu-20.04.3-live-server-arm64.iso*
- vmlinuz extracted from ubuntu-20.04.3-live-server-arm64.iso

Please replace the corresponding files when you want to work on amd64 image. For example, your files may be:

- Ubuntu 20.04.3 live server image *ubuntu-20.04.3-live-server-amd64.iso* from https://releases.ubuntu.com/20.04. 3/ubuntu-20.04.3-live-server-amd64.iso
- grub efi binary grubnetx64.efi.signed from http://archive.ubuntu.com/ubuntu/dists/focal/main/uefi/grub2-amd64/current/grubnetx64.efi.signed
- initrd extracted from *ubuntu-20.04.3-live-server-amd64.iso*
- vmlinuz extracted from ubuntu-20.04.3-live-server-amd64.iso

Download and Serve Grub EFI Binary

The grub binary helps us redirect the downloading path to the target files via grub.cfg. You may refer to this discourse post to get more information about the PXE process and why we need this binary.

\$ sudo wget http://ports.ubuntu.com/ubuntu-ports/dists/focal/main/uefi/grub2-arm64/current/grubnetaa64.efi.signed 0 /var/lib/tftpboot/grubnetaa64.efi.signed

Please note you may need to change **the archive dists name** from **focal** to your target distribution name.

Download and Serve More Files

Fetch the installer by downloading a Ubuntu arm server iso, e.g. 20.04.3 live server arm64 iso. Please note the prefix *live* is significant. We will need the files available only in the live version.

Mount the iso and copy the target files we need to the TFTP folder

\$ sudo mount ./ubuntu-20.04.3-live-server-arm64.iso /mnt \$ sudo mkdir /var/lib/tftpboot/grub /var/lib/tftpboot/casper \$ sudo cp /mnt/boot/grub/grub.cfg /var/lib/tftpboot/grub/ \$ sudo cp /mnt/casper/initrd /var/lib/tftpboot/casper/ \$ sudo cp /mnt/casper/vmlinuz /var/lib/tftpboot/casper/

So, the TFTP root folder should look like this now:

```
$ find /var/lib/tftpboot/
/var/lib/tftpboot/grub
/var/lib/tftpboot/grub/grub.cfg
/var/lib/tftpboot/grubnetaa64.efi.signed
/var/lib/tftpboot/casper
/var/lib/tftpboot/casper/initrd
/var/lib/tftpboot/casper/vmlinuz
```

Finally, let's customize the grub menu so we could install our target image by fetching it directly over the internet.

```
$ sudo chmod +w /var/lib/tftpboot/grub/grub.cfg
$ sudo vi /var/lib/tftpboot/grub/grub.cfg
```

Add an new entry

```
menuentry "Install Ubuntu Server (Focal 20.04.3) (Pull the iso from web)" {
    set gfxpayload=keep
    linux /casper/vmlinuz url=http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-
live-server-arm64.iso only-ubiquity ip=dhcp ---
    initrd /casper/initrd
}
```

ip=dhcp is for the dhcp management setup in the lab. url is used to point to your target image download url. Remember to change them according to your scenario.

If everything goes well, you should get into the expected grub menu of the ephemeral live prompt. Select the entry you just put in grub.cfg, which is Install Ubuntu Server (Focal 20.04.3) (Pull the iso from web) in our example. Waiting a bit for downloading the iso and then you will see the subiquity welcome message. Enjoy the installation!

Appendix

Always Make Sure of the Serving File Names

For example, please make sure the target file name for *linux* and *initrd* is correct. For example, the default initrd binary file name of 20.04.3 is *initrd*, and it is *initrd.lz* for 20.10. Always make sure you serve the right file names. This is a frequent troubleshooting issue. Pay attention on this detail could save a lot of your time.

Booting Screenshots

If your setup is correct, your grub.cfg should redirect the process to an ephemeral environment to download your target image assigned in the grub entry of grub.cfg. You will see a screen like this if you are able to access console or monitor device of your target machine:

5:e6:08:2d:e3				
5:e6:08:2d:e3				
sk				
5.255.255.255 port 67 interval 3 ((xid=0x8	3e74430)		
from 10.228.68.5				
73 on ens3 to 255.255.255.255 port	: 67 (xi	d=0x3044e783)		
om 10.228.68.5 (xid=0x83e74430)				
enewal in 286 seconds.				
(10.228.68.91:80)				
	462k	0:34:48 ETA		
	22.2M	0:01:20 ETA		
*	39.7M	0:01:05 ETA		
жж	59.6M	0:00:57 ETA		
жж	77.7M	0:00:53 ETA		
жжж	97.7M	0:00:50 ETA		
жжж	114M	0:00:49 ETA		
жжж	132M	0:00:47 ETA		
жжжж	149M	0:00:45 ETA		
жжжж	166M	0:00:45 ETA		
жжжж	183M	0:00:43 ETA		
жжжжжж	200M	0:00:42 ETA		
жжжжжж	214M	0:00:42 ETA		
	5:e6:08:2d:e3 5:e6:08:2d:e3 5: from 10.228.68.5 73 on ens3 to 255.255.255.255 port om 10.228.68.5 (xid=0x83e74430) enewal in 286 seconds. (10.228.68.91:80)	5:e6:08:2d:e3 5:e6:08:2d:e3 ck 5:255.255.255 port 67 interval 3 (xid=0x8 from 10.228.68.5 73 on ens3 to 255.255.255 port 67 (xi om 10.228.68.5 (xid=0x83e74430) enewal in 286 seconds. (10.228.68.91:80) 462k (10.228.68.91:80) 462k (22.2M **** **** *** **** **** *** *** **** *** **** **** ***** **** **** ***** **** ***** ******	5:e6:08:2d:e3 5:e6:08:2d:e3 5:255.255.255 port 67 interval 3 (xid=0x83e74430) from 10.228.68.5 73 on ens3 to 255.255.255 port 67 (xid=0x3044e783) om 10.228.68.5 (xid=0x83e74430) enewal in 286 seconds. (10.228.68.91:80) 462k 0:34:48 ETA (10.228.68.91:80) 462k 0:34:48 ETA 22.2M 0:01:20 ETA **** 39.7M 0:00:55 ETA **** ***** 114M 0:00:45 ETA ******** 114M 0:00:45 ETA ******** 116M 0:00:45 ETA ******** ******* ******* 200M 0:00:43 ETA ******** 200M 0:00:42 ETA	5:e6:08:2d:e3 5:e6:08:2d:e3 ck 5:255.255.255 port 67 interval 3 (xid=0x83e74430) from 10.228.68.5 73 on ens3 to 255.255.255 port 67 (xid=0x3044e783) om 10.228.68.5 (xid=0x83e74430) enewal in 286 seconds. (10.228.68.91:80) 462k 0:34:48 ETA (10.228.68.91:80) 462k 0:34:48 ETA 22.2M 0:01:20 ETA **** 39.7M 0:01:05 ETA **** **** 59.6M 0:00:57 ETA **** **** 177.7M 0:00:53 ETA **** **** 114M 0:00:49 ETA ******** ******* 149M 0:00:45 ETA ******** 166M 0:00:45 ETA ******** 183M 0:00:43 ETA ********* 200M 0:00:42 ETA

Ephemeral Env is Downloading Target Image1006 \times 359 3.24 KB

Wait a bit to complete downloading. If you see this subiquity welcome page, the installer is successfully launched via your UEFI PXE setup. Configurations!!

Willkommen! Bienven	ue! Welcome! Добро пожаловать! Welkom!	[Help]
Use UP, DOWN and EN	TER keys to select your language.	
	Asturianu •] Bahasa Indonesia •] Català •] Deutsch •] English •] English (UK) •] Español •] Français •] Hrvatski •] Latviski •] Lietuviškai •] Norsk bokmål •] Polski •] Svenska •] Exληνικά •] Exληνικά •] Pycckwň •] Ykpaïhcьka •]	

Subiquity Welcome Page849 \times 468 3.52 KB

Netbooting the live server installer on IBM Power (ppc64el) with Petitboot

- Open a terminal window on your workstation and make sure the 'ipmitool' package is installed.
- Verify if you can reach the BMC of the IBM Power system via ipmitool with a simple ipmitool call like:

```
$ ipmitool -I lanplus -H Power9Box -U <user> -P <password> power status
Chassis Power is off
or:
$ ipmitool -I lanplus -H Power9Box -U <user> -P <password> fru print 47
Product Name : OpenPOWER Firmware
Product Version : open-power-SUPERMICR0-P9DSU-V2.12-20190404-prod
Product Extra : op-build-1b9269e
Product Extra : buildroot-2018.11.3-12-g222837a
```

Product Extra	: ski	boot-v6.0.19
Product Extra	: hos	tboot-c00d44a-pb1307d7
Product Extra	: occ	-8fa3854
Product Extra	: lin	ux-4.19.30-openpower1-p22d1df8
Product Extra	: pet	itboot-v1.7.5-p8f5fc86
or:		
\$ ipmitool -I la	nplus -H Power9	Box -U <user> -P <password> sol info</password></user>
Set in progres	S	: set-complete
Enabled		: true
Force Encrypti	on	: false
Force Authenti	cation	: false
Privilege Leve	l	: OPERATOR
Character Accu	mulate Level (m	s) : 0
Character Send	Threshold	: 0
Retry Count		: 0
Retry Interval	(ms)	: 0
Volatile Bit R	ate (kbps)	: 115.2
Non-Volatile B	it Rate (kbps)	: 115.2
Payload Channe	l	: 1 (0×01)
Payload Port		: 623

- Open a second terminal and activate serial-over-LAN (sol), so that you have two terminal windows open:
 - 1. to control the BMC via IPMI
 - 2. for the serial-over-LAN console
- Activate serial-over-LAN:

```
$ ipmitool -I lanplus -H Power9Box -U <user> -P <password> sol activate
...
```

• And power the system on in the 'control terminal' and watch the sol console:

```
$ ipmitool -I lanplus -H Power9Box -U <user> -P <password> power on
...
```

It takes some time to see the first lines in the sol console:

```
[SOL Session operational. Use ~? for help]
--== Welcome to Hostboot ==--
```

```
2.77131|secure|SecureROM valid - enabling functionality
3.15860|secure|Booting in secure mode.
5.59684|Booting from SBE side 0 on master proc=00050000
5.60502|ISTEP 6. 5 - host_init_fsi
5.87228|ISTEP 6. 6 - host_set_ipl_parms
6.11032|ISTEP 6. 7 - host_discover_targets
6.67868|HWAS|PRESENT> DIMM[03]=A0A0000000000000
6.67870|HWAS|PRESENT> Proc[05]=880000000000000
6.67871|HWAS|PRESENT> Core[07]=3FFF0C33FFC30000
6.98988|ISTEP 6. 8 - host_update_master_tpm
7.22711|SECURE|Security Access Bit> 0xC000000000000000
7.22731|ISTEP 6. 9 - host_gard
7.43353|HWAS|FUNCTIONAL> DIMM[03]=A0A0000000000000
7.43354|HWAS|FUNCTIONAL> Proc[05]=8800000000000000
7.43356|HWAS|FUNCTIONAL> Core[07]=3FFF0C33FFC30000
7.44509|ISTEP 6.10 - host_revert_sbe_mcs_setup
```

• After a moment the system reaches the Petitboot screen:

Petitboot (v1.7.5-p8f5fc86)

9006-12P 1302NXA

```
[Network: enP2p1s0f0 / 0c:c4:7a:87:04:d8]
Execute
netboot enP2p1s0f0 (pxelinux.0)
```

[CD/DVD: sr0 / 2019-10-17-13-35-12-00] Install Ubuntu Server [Disk: sda2 / 295f571b-b731-4ebb-b752-60aadc80fc1b] Ubuntu, with Linux 5.4.0-14-generic (recovery mode) Ubuntu, with Linux 5.4.0-14-generic Ubuntu System information System configuration System status log Language Rescan devices Retrieve config from URL Plugins (0)

*Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, g=log, h=help

Select '*Exit to shell'

Notice:

Make sure you really watch the sol, since the petitboot screen (above) has a time out (usually 10 or 30 seconds) and afterwards it automatically proceeds and it tries to boot from the configured devices (usually disk). This can be prevented by just navigating in petitboot.

The petitboot shell is small Linux based OS:

• • •

```
Exiting petitboot. Type 'exit' to return.
You may run 'pb-sos' to gather diagnostic data
```

Notice:

In case one needs to gather system details and diagnostic data for IBM support, this can be done here by running 'pb-sos' (see msg).

• Now download the 'live-server' ISO image (notice that 'focal-live-server-ppc64el.iso' uses subiquity, 'focal-server-s390x.iso' uses d-i):

Again for certain web locations a proxy needs to be used:

```
/ # export http_proxy=http://squid.proxy:3128  # in case a proxy is required
/ #
/ # wget http://cdimage.ubuntu.com/ubuntu-server/daily-live/current/focal-live-server-ppc64el.iso
Connecting to <proxy-ip>:3128 (<proxy-ip>:3128)
focal-live-server-pp 100% |...... 922M 0:00:00 ETA
```

• Next is to loop-back mount the ISO:

```
/ # mkdir iso
/ # mount -o loop focal-live-server-ppc64el.iso iso
```

Or in case autodetect of type iso9660 is not supported or not working, explicitly specify the 'iso9660' type:

/ # mount -t iso9660 -o loop focal-live-server-ppc64el.iso iso

• Now load kernel and initrd from the loop-back mount, specify any needed kernel parameters and get it going:

```
/ # kexec -l ./iso/casper/vmlinux --initrd=./iso/casper/initrd.gz --append="ip=dhcp url=http://cdimage.ubuntu.com/
server/daily-live/current/focal-live-server-ppc64el.iso http_proxy=http://squid.proxy:3128 --- quiet"
/ # kexec -e
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
...
```

Note that in order to boot with and install the hwe kernel (if available), just substitute vmlinux with vmlinux-hwe in the first kexec line.

• The system now performs the initial boot of the installer:

```
[ 1200.687004] kexec_core: Starting new kernel
```

[1277.493883374,5] OPAL: Switch to big-endian OS [1280.465061219,5] OPAL: Switch to little-endian OS ln: /tmp/mountroot-fail-hooks.d//scripts/init-premount/lvm2: No such file or directory Internet Systems Consortium DHCP Client 4.4.1 Copyright 2004-2018 Internet Systems Consortium. All rights reserved. For info, please visit https://www.isc.org/software/dhcp/ Listening on LPF/enP2p1s0f3/0c:c4:7a:87:04:db Sending on LPF/enP2p1s0f3/0c:c4:7a:87:04:db Listening on LPF/enP2p1s0f2/0c:c4:7a:87:04:da Sending on LPF/enP2p1s0f2/0c:c4:7a:87:04:da Listening on LPF/enP2p1s0f1/0c:c4:7a:87:04:d9 Sending on LPF/enP2p1s0f1/0c:c4:7a:87:04:d9 Listening on LPF/enP2p1s0f0/0c:c4:7a:87:04:d8 Sending on LPF/enP2p1s0f0/0c:c4:7a:87:04:d8 Sending on Socket/fallback DHCPDISCOVER on enP2p1s0f3 to 255.255.255.255 port 67 interval 3 (xid=0x8d5704c) DHCPDISCOVER on enP2p1s0f2 to 255.255.255.255 port 67 interval 3 (xid=0x94b25b28) DHCPDISCOVER on enP2p1s0f1 to 255.255.255.255 port 67 interval 3 (xid=0x4edd0558) DHCPDISCOVER on enP2p1s0f0 to 255.255.255.255 port 67 interval 3 (xid=0x61c90d28) DHCPOFFER of 10.245.71.102 from 10.245.71.3 DHCPREQUEST for 10.245.71.102 on enP2p1s0f0 to 255.255.255.255 port 67 (xid=0x280dc961) DHCPACK of 10.245.71.102 from 10.245.71.3 (xid=0x61c90d28) bound to 10.245.71.102 -- renewal in 236 seconds. Connecting to 91.189.89.11:3128 (91.189.89.11:3128) focal-live-server-pp 1% | | 14.0M 0:01:04 ETA focal-live-server-pp 4% |* | 45.1M 0:00:38 ETA focal-live-server-pp 8% |** | 76.7M 0:00:33 ETA | 105M 0:00:31 ETA focal-live-server-pp 11% |*** focal-live-server-pp 14% |**** | 133M 0:00:29 ETA focal-live-server-pp 17% |***** | 163M 0:00:27 ETA focal-live-server-pp 20% |***** | 190M 0:00:26 ETA focal-live-server-pp 24% |****** | 222M 0:00:25 ETA focal-live-server-pp 27% |******* | 253M 0:00:23 ETA focal-live-server-pp 30% |******** | 283M 0:00:22 ETA focal-live-server-pp 34% |******** | 315M 0:00:21 ETA focal-live-server-pp 37% |********* | 343M 0:00:20 ETA focal-live-server-pp 39% |*********** | 367M 0:00:19 ETA focal-live-server-pp 42% |************ | 392M 0:00:18 ETA focal-live-server-pp 45% |************* | 420M 0:00:17 ETA | 451M 0:00:16 ETA focal-live-server-pp 52% |*************** | 482M 0:00:15 ETA focal-live-server-pp 55% |***************** | 514M 0:00:14 ETA focal-live-server-pp 59% |******************** | 546M 0:00:13 ETA . focal-live-server-pp 62% |********************** | 578M 0:00:11 ETA 607M 0:00:10 ETA focal-live-server-pp 69% |************************ | 637M 0:00:09 ETA | 669M 0:00:08 ETA | 700M 0:00:07 ETA | 729M 0:00:06 ETA focal-live-server-pp 97% |*********************************** | 897M 0:00:00 ETA focal-live-server-pp 100% |******************************** 922M 0:00:00 ETA mount: mounting /cow on /root/cow failed: No such file or directory

Connecting to plymouth: Connection refused
passwd: password expiry information changed.
[47.202736] /dev/loop3: Can't open blockdev
[52.672550] cloud-init[3759]: Cloud-init v. 20.1-10-g71af48df-0ubuntul running
'init-local' at Wed, 18 Mar 2020 15:18:07 +0000. Up 51.87 seconds.
...

• And you will eventually reach the initial subiquity installer screen:

Willkommen! Bienvenue! Welcome! Добро пожаловать! Welkom	
Use UP, DOWN and ENTER keys to select your language.	
[English	•]
[Asturianu	▶]
[Català	•]
[Hrvatski	•]
[Nederlands	•]
[Suomi	▶]
[Français	▶]
[Deutsch	▶]
[Ελληνικά	•]
[Magyar	•]
[Latviešu	•]
[Norsk bokmål	•]
[Polski	•]
[Русский	•]
[Español	•]
[Українська	•]

* The rest is (subiquity-) installation as usual...

Using a virtual CDROM and Petitboot to start a live server installation on IBM Power (ppc64el)

There is also documentation on booting the installer over the network.

• Notice:

Not all IBM Power machines come with the capability to install via a virtual CDROM !

- A separate system (ideally in the same network, because of ipmitool) is needed to host the ppc64el ISO Image file, that is later used as virtual CDROM.
- Login to this separate host and make sure that the ipmitool package is installed:

\$ sudo apt install ipmitool

as well as Samba:

\$ sudo apt install samba

• Next is to setup and configure Samba:

```
$ sudo touch /etc/samba/smb.conf && sudo tee -a /etc/samba/smb.conf <<EOF
[winshare]
   path=/var/winshare
   browseable = yes
   read only = no
   guest ok = yes
EOF</pre>
```

And do a quick verification that the required lines are in:

```
$ tail -n 5 /etc/samba/smb.conf
[winshare]
path=/var/winshare
browseable = yes
```
read only = no guest ok = yes

• (Optional)

For downloading the image you may have to use a proxy server:

```
$ sudo touch ~/.wgetrc && sudo tee -a ~/.wgetrc <<EOF
use_proxy=yes
http_proxy=squid.proxy:3128
https_proxy=squid.proxy:3128
EOF</pre>
```

• The ISO image needs to be downloaded now:

\$ wget http://cdimage.ubuntu.com/ubuntu/releases/focal/release/ubuntu-20.04-live-server-ppc64el.iso -directory-prefix=/var/winshare

The proxy can also be passed over as wget argument, like this:

\$ wget -e use_proxy=yes -e http_proxy=squid.proxy:3128 http://cdimage.ubuntu.com/ubuntu/releases/focal/release/ubu 20.04-live-server-ppc64el.iso --directory-prefix=/var/winshare

• Change file mode of the ISO image file:

```
$ sudo chmod -R 755 /var/winshare/
$ ls -l /var/winshare/
-rwxr-xr-x 1 ubuntu ubuntu 972500992 Mar 23 08:02 focal-live-server-ppc64el.iso
```

- Restart and check the Samba service:
- \$ sudo service smbd restart

```
$ sudo service smbd status
```

```
    smbd.service - Samba SMB Daemon
Loaded: loaded (/lib/systemd/system/smbd.service; enabled; vendor
preset: ena
Active: active (running) since Tue 2020-02-04 15:17:12 UTC; 4s ago
```

```
Docs: man:smbd(8)
    man:samba(7)
    man:smb.conf(5)
```

• Test Samba share:

```
ubuntu@host:~$ smbclient -L localhost
WARNING: The "syslog" option is deprecated
Enter WORKGROUP\ubuntu's password:
   Sharename Type
                             Comment
    - - - - - - - - - -
                   - - - -
                             -----
                             Printer Drivers
   print$
                 Disk
                 Disk
   winshare
                  IPC
                             IPC Service (host server (Samba, Ubuntu))
   IPC$
   Reconnecting with SMB1 for workgroup listing.
   Server
                        Comment
    ----
                         - - - - - - -
   Workgroup
                        Master
    - - - - - - - - - -
                        ----
   WORKGROUP
                        host
```

- Get the IP address of the Samba host:
 - \$ ip -4 -brief address show

lo	UNKNOWN	127.0.0.1/8
ibmveth2	UNKNOWN	10.245.246.42/24

• (Optional)

Even more testing if the Samba share is accessible from remote:

```
user@workstation:~$ mkdir -p /tmp/test
user@workstation:~$ sudo mount -t cifs -o
username=guest,password=guest //10.245.246.42/winshare /tmp/test/
user@workstation:~$ ls -la /tmp/test/
total 1014784
drwxr-xr-x 2 root root 0 May 4 15:46 .
drwxrwxrwt 18 root root 420 May 4 19:25 ..
-rwxr-xr-x 1 root root 1038249984 May 3 19:37 ubuntu-20.04-live-server-ppc64el.iso
```

• Now use a browser and navigate to the BMC of the Power system that should be installed (let's assume the BMC's IP address is 10.245.246.247):

firefox http://10.245.246.247/

- Login to the BMC and find and select: Virtual Media --> CDROM
- Enter the IP address of the Samba share: 10.245.246.42 and the path to the Samba share:

\winshare\focal-live-server-ppc64el.iso

• Click Save and Mount (make sure that the virtual CDROM is really properly mounted !)

CD-ROM Image:

This option allows you to share a CD-ROM image over a Windows Share with a maximum size of 4.7GB. This image will be emulated to the host as USB device.

```
Device 1 There is an iso file mounted.
Device 2 No disk emulation set.
Device 3 No disk emulation set.
<Refresh Status>
```

Share host: 10.245.246.42
Path to image: \winshare\focal-live-server-ppc64el.iso
User (optional):
Password (optional):
<Save> <Mount> <Unmount>

• Notice:

It's important that you see a status like:

Device 1 There is an iso file mounted

Only in this case the virtual CDROM is properly mounted and you will see the boot / install from CDROM entry in petitboot:

[CD/DVD: sr0 / 2020-03-23-08-02-42-00] Install Ubuntu Server

• Now use the ipmitool to boot the system into the petitboot loader:

\$ ipmitool -I lanplus -H 10.245.246.247 -U ADMIN -P <password> power status
\$ ipmitool -I lanplus -H 10.245.246.247 -U ADMIN -P <password> sol activate
\$ ipmitool -I lanplus -H 10.245.246.247 -U ADMIN -P <password> power on
Chassis Power Control: Up/On

• And reach the Petitboot screen:

Petitboot (v1.7.5-p8f5fc86)

9006-12C B0S0026

```
[Network: enP2pls0f0 / ac:lf:6b:09:c0:52]
execute
```

netboot enP2p1s0f0 (pxelinux.0)

```
System information
System configuration
System status log
Language
Rescan devices
Retrieve config from URL
*Plugins (0)
Exit to shell
```

Enter=accept, e=edit, n=new, x=exit, l=language, g=log, h=help
Default boot cancelled

• And make sure that booting from CDROM is enabled:

Petitboot (v1.7.5-p8f5fc86)

9006-12C B0S0026

```
[Network: enP2pls0f0 / ac:lf:6b:09:c0:52]
Execute
netboot enP2pls0f0 (pxelinux.0)
[Disk: sda2 / ebdb022b-96b2-4f4f-ae63-69300ded13f4]
Ubuntu, with Linux 5.4.0-12-generic (recovery mode)
Ubuntu, with Linux 5.4.0-12-generic
Ubuntu
System information
System configuration
System status log
Language
Rescan devices
Retrieve config from URL
*Plugins (0)
Exit to shell
```

Enter=accept, e=edit, n=new, x=exit, l=language, g=log, h=help
[sda3] Processing new Disk device

```
Petitboot System Configuration
```

Autoboot:	<pre>() Disabled (*) Enabled</pre>
Boot Order:	 (0) Any CD/DVD device (1) disk: sda2 [uuid: ebdb022b-96b2-4f4f-ae63-69300ded13f4] (2) net: enP2p1s0f0 [mac: ac:1f:6b:09:c0:52]
	[Add Device] [Clear & Boot Any] [Clear]
Timeout:	30 seconds
Network:	(*) DHCP on all active interfaces() DHCP on a specific interface() Static IP configuration

tab=next, shift+tab=previous, x=exit, h=help
Petitboot System Configuration

```
Network:
               (*) DHCP on all active interfaces
               () DHCP on a specific interface
               () Static IP configuration
DNS Server(s):
                                                 (eg. 192.168.0.2)
               (if not provided by DHCP server)
HTTP Proxy:
HTTPS Proxy:
Disk R/W:
               ( ) Prevent all writes to disk
               (*) Allow bootloader scripts to modify disks
Boot console: (*) /dev/hvc0 [IPMI / Serial]
               () /dev/tty1 [VGA]
               Current interface: /dev/hvc0
                    0K
                          ] [ Help ] [ Cancel ]
               ſ
```

tab=next, shift+tab=previous, x=exit, h=help

• Now select the 'Install Ubuntu Server' entry below the CD/DVD entry:

```
[CD/DVD: sr0 / 2020-03-23-08-02-42-00]
* Install Ubuntu Server
```

• And let Petitboot boot from the (virtual) CDROM image:

Sent SIGKILL to all processes
[119.355371] kexec_core: Starting new kernel
[194.483947394,5] OPAL: Switch to big-endian OS
[197.454615202,5] OPAL: Switch to little-endian OS

• Finally the initial subiquity installer screen will show up in the console:

Willkommen! Bienvenue! Welcome! Добро пожаловать! Welkom

Use UP, DOWN and ENTER keys to select your language.

[English	۲]
[Asturianu	۲]
[Català	۲]
[Hrvatski	۲]
[Nederlands	۲]
[Suomi	۲]
[Français	۲]
[Deutsch	۲]
[Ελληνικά	۲]
[Magyar	۲]
[Latviešu	۲]
[Norsk bokmål	۲]
[Polski	►]
[Русский	►]
[Español	►]
[Українська	►]

• The rest of the installation is business as usual ...

Interactive live server installation on IBM z/VM (s390x)

Doing a manual live installation like described here - means without specifying a parmfile - is supported since Ubuntu Server LTS 20.04.3 ('focal') and any newer release, like 20.10 ('groovy').

• The following guide assumes that a z/VM guest got defined, and that it is able to either reach the public

cdimage.ubuntu.com server or an internal FTP or HTTP server that hosts an Ubuntu Server 20.04 installer image, like this 20.04 aka focal daily live image here: http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso

• Find a place to download the installer image:

```
user@workstation:~$ wget
http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso
--2020-08-08 16:01:52--
http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso
Resolving cdimage.ubuntu.com (cdimage.ubuntu.com)... 2001:67c:1560:8001::1d, 2001:67c:1360:8001::27, 2001:67c:1360
Connecting to cdimage.ubuntu.com
(cdimage.ubuntu.com)|2001:67c:1560:8001::1d|:80... connected.
HTTP request sent, awaiting response... 200 0K
Length: 705628160 (673M) [application/x-iso9660-image]
Saving to: 'ubuntu-20.04.3-live-server-s390x.iso'
ubuntu-20.04.3-live 100%[======>] 672.94M 37.1MB/s in
17s
```

2020-08-08 16:02:10 (38.8 MB/s) - 'ubuntu-20.04.3-live-server-s390x.iso' saved [705628160/705628160]

```
user@workstation:~$
```

• Now loop-back mount the ISO to extract four files that are needed for a z/VM guest installation

```
user@workstation:~$ mkdir iso
user@workstation:~$ sudo mount -o loop ubuntu-20.04.3-live-server-s390x.iso iso
user@workstation:~$
```

```
user@workstation:~$ ls -1 ./iso/boot/{ubuntu.exec,parmfile.*,kernel.u*,initrd.u*}
./iso/boot/initrd.ubuntu
./iso/boot/kernel.ubuntu
./iso/boot/parmfile.ubuntu
./iso/boot/ubuntu.exec
user@workstation:~$
```

- Now transfer these four files to your z/VM guest (for example to its 'A' file mode), using either the 3270 terminal emulator or ftp.
- Then log on to your z/VM guest that you want to use for the installation, in this example it will be guest '10.222.111.24'
- And execute the ubuntu REXX script to kick-off the installation:

```
ubuntu

00: 0000004 FILES PURGED

00: RDR FILE 0125 SENT FROM 10.222.111.24 PUN WAS 0125 RECS 101K CPY 001 A NOHOLD NO

KEEP

00: RDR FILE 0129 SENT FROM 10.222.111.24 PUN WAS 0129 RECS 0001 CPY 001 A NOHOLD NO

KEEP

00: RDR FILE 0133 SENT FROM 10.222.111.24 PUN WAS 0133 RECS 334K CPY 001 A NOHOLD NO

KEEP

00: 0000003 FILES CHANGED

00: 0000003 FILES CHANGED

01: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

02: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

03: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

03: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

04: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

05: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

06: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial CPU reset from CPU 00.

07: 0.390935| Initramfs unpacking failed: Decoding failed

Unable to find a medium container a live file system
```

• In the usual case that no parmfile got configured, the installation system now offers to interactively configure the basic network:

Attempt interactive netboot from a URL? yes no (default yes): yes Available qeth devices:

```
0.0.0600 0.0.0603
zdev to activate (comma separated, optional): 0600
QETH device 0.0.0600:0.0.0601:0.0.0602 configured
Two methods available for IP configuration:
    * static: for static IP configuration
    * dhcp: for automatic IP configuration
static dhcp (default 'dhcp'): static
ip: 10.222.111.24
gateway (default 10.222.111.1): .
dns (default .):
vlan id (optional):
http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso (default)
url: ftp://10.11.12.2:21/ubuntu-live-server-20.04.3/ubuntu-20.04.3-live-server-s390x.iso
http proxy (optional):
```

- Make sure that the same version of the ISO image is referenced at the 'url:' setting, that was used to extract the installer files kernel and initrd. But it can be at a different location, for example directly referencing the public cdimage.ubuntu.com server: http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso
- The boot-up of the live-server installation now completes:

```
Configuring networking...
QETH device 0.0.0600:0.0.0601:0.0.0602 already configured
IP-Config: enc600 hardware address 02:28:0a:00:00:39 mtu 1500
IP-Config: enc600 guessed broadcast address 10.222.111255
IP-Config: enc600 complete:
address: 10.222.111.24
                  broadcast: 10.222.111255 netmask: 255.255.255.0
gateway: 10.222.111.1
                  dns0
                        : 10.222.111.1
                                      dns1 : 0.0.0.0
rootserver: 0.0.0.0 rootpath:
filename :
Connecting to 10.11.12.2:21 (10.11.12.2:21)
focal-live-server-s 5% !*
                                        ! 35.9M 0:00:17 ETA
focal-live-server-s 19% !*****
                                        ! 129M 0:00:08 ETA
focal-live-server-s 33% !********
                                        ! 225M 0:00:05 ETA
! 330M 0:00:04 ETA
! 403M 0:00:03 ETA
! 506M 0:00:01 ETA
passwd: password expiry information changed.
QETH device 0.0.0600:0.0.0601:0.0.0602 already configured
no search or nameservers found in /run/net-enc600.conf /run/net-*.conf /run/net6
-*.conf
- 594.766372| /dev/loop3: Can't open blockdev
- 595.610434| systemd-1|: multi-user.target: Job getty.target/start deleted to
break ordering cycle starting with multi-user.target/start
¬ ¬0;1;31m SKIP ¬0m| Ordering cycle found, skipping ¬0;1;39mLogin Prompts ¬0m
- 595.623027| systemd-1|: Failed unmounting /cdrom.
¬ ¬0;1;31mFAILED ¬0m| Failed unmounting ¬0;1;39m/cdrom ¬0m.
- 598.973538| cloud-init-1256|: Cloud-init v. 20.2-45-g5f7825e2-0ubuntul runnin
g 'init-local' at Thu, 04 Jun 2020 12:06:46 +0000. Up 598.72 seconds.
- 599.829069| cloud-init-1288|: Cloud-init v. 20.2-45-g5f7825e2-0ubuntul runnin
 'init' at Thu, 04 Jun 2020 12:06:47 +0000. Up 599.64 seconds.
a
599.829255| cloud-init-1288|: ci-info: ! Device ! Up !
                                               Address
       Mask
             ! Scope !
  1
                        Hw-Address
                                  1
```

```
----+
¬ 599.829333| cloud-init¬1288|: ci-info: ! enc600 ! True ! 10.222.111.24
  ! 255.255.255.0 ! global ! 02:28:0a:00:00:39 !
- 599.829376| cloud-init-1288|: ci-info: ! enc600 ! True ! fe80::28:aff:fe00:3
/64 ! . ! link ! 02:28:0a:00:00:39 !
 599.829416| cloud-init-1288|: ci-info: ! lo ! True !
                                             127.0.0.1
  ! 255.0.0.0 ! host ! .
                                !
 599.829606| cloud-init-1288|: ci-info: ! lo ! True !
                                              ::1/128
  ! . ! host ! .
                               !
 ----+
+----+
- 599.829789| cloud-init-1288|: ci-info: ! Route ! Destination ! Gateway
  Genmask ! Interface ! Flags !
1
+----+
- 599.829858| cloud-init-1288|: ci-info: ! 0 ! 0.0.0.0 ! 10.222.111.1
   0.0.0.0 ! enc600 ! UG !
1
- 599.829896| cloud-init-1288|: ci-info: ! 1 ! 10.222.1110 ! 0.0.0.0
! 255.255.255.0 ! enc600 ! U !
+----+
¬ 599.829998| cloud-init¬1288|: ci-info: +-----+-----+-----+-----+------
----+
- 599.830031| cloud-init-1288|: ci-info: ! Route ! Destination ! Gateway ! Inte
rface ! Flags !
----+
- 599.830096| cloud-init-1288|: ci-info: ! 1 ! fe80::/64 ! :: ! en
c600 ! U !
- 599.830131| cloud-init-1288|: ci-info: ! 3 ! local ! :: !
                                                      en
c600 ! U !
- 599.830164| cloud-init-1288|: ci-info: ! 4 ! ff00::/8 ! :: ! en
c600 ! U !
¬ 599.830212| cloud-init¬1288|: ci-info: +-----+-----+-----+-----+------+------
----+
- 601.077953| cloud-init-1288|: Generating public/private rsa key pair.
- 601.078101| cloud-init-1288|: Your identification has been saved in /etc/ssh/
ssh_host_rsa_key
- 601.078136| cloud-init-1288|: Your public key has been saved in /etc/ssh/ssh
host_rsa_key.pub
- 601.078170| cloud-init-1288|: The key fingerprint is:
- 601.078203| cloud-init-1288|: SHA256:kHtkABZwk8AE80fy0KPzTRcYpht4iXdZmJ37Cgi3
fJ0 root§ubuntu-server
- 601.078236| cloud-init-1288|: The key's randomart image is:
¬ 601.078274| cloud-init¬1288|: +---¬RSA 3072|----+
¬ 601.078307| cloud-init¬1288|: !o+*+B++*..
                                  . !
- 601.078340| cloud-init-1288|: ! o.X+=+=+
                                  !
¬ 601.078373| cloud-init¬1288|: ! +.0.= oo
                                  - !
¬ 601.078406| cloud-init¬1288|: ! ++.+.=0
¬ 601.078439| cloud-init¬1288|: !
                         *.=.oSo
¬ 601.078471| cloud-init¬1288|: ! = +.E .
                                   1
¬ 601.078503| cloud-init¬1288|: ! . . .
                                  - 1
- 601.078537| cloud-init-1288|: !
- 601.078570| cloud-init-1288|: !
- 601.078602| cloud-init-1288|: +----SHA256|----+
- 601.078635| cloud-init-1288|: Generating public/private dsa key pair.
- 601.078671| cloud-init-1288|: Your identification has been saved in /etc/ssh/
```

```
ssh_host_dsa_key
- 601.078704| cloud-init-1288|: Your public key has been saved in /etc/ssh/ssh
host dsa key.pub
- 601.078736| cloud-init-1288|: The key fingerprint is:
- 601.078767| cloud-init-1288|: SHA256:ZBNyksVVYZVhKJeL+PWKpsdUcn21yiceX/DboXQd
Pq0 root§ubuntu-server
¬ 601.078800| cloud-init¬1288|: The key's randomart image is:
  601.078835| cloud-init-1288|: +----DSA 1024|----+
  601.078867| cloud-init¬1288|: !
                                     0++...+=+0 !
  601.078899| cloud-init-1288|: !
                                     .+...+...!
- 601.078932| cloud-init-1288|: !
                                       +. + 0 0!
- 601.078964| cloud-init-1288|: !
                                      0..0 = 00.!
- 601.078996| cloud-init-1288|: !
                                      S. =..0++!
¬ 601.079029| cloud-init¬1288|: !
                                         0 *.*=!
¬ 601.079061| cloud-init¬1288|: !
                                        o .o.B.*!
- 601.079094| cloud-init-1288|: !
                                         = .oEo.!
- 601.079135| cloud-init-1288|: !
                                        .+
                                            1
¬ 601.079167| cloud-init¬1288|: +----¬SHA256|----+
- 601.079199| cloud-init-1288|: Generating public/private ecdsa key pair.
- 601.079231| cloud-init-1288|: Your identification has been saved in /etc/ssh/
ssh_host_ecdsa_key
- 601.079263| cloud-init-1288|: Your public key has been saved in /etc/ssh/ssh_
host_ecdsa_key.pub
- 601.079295| cloud-init-1288|: The key fingerprint is:
  601.079327| cloud-init-1288|: SHA256:Bitar9fVHUH2FnYVSJJnldprdAcM5Est0dmRWFTU
i8k root§ubuntu-server
  601.079362| cloud-init-1288|: The key's randomart image is:
-
- 601.079394| cloud-init-1288|: +----ECDSA 256|---+
- 601.079426| cloud-init-1288|: !
                                     o**0%&!
¬ 601.079458| cloud-init¬1288|: !
                                           0.0B+=!
¬ 601.079491| cloud-init¬1288|: !
                                          B *o+!
                                     0
  601.079525| cloud-init¬1288|: !
                                           . E.=o!
- 601.079557| cloud-init-1288|: !
                                   o.S....+!
- 601.079589| cloud-init-1288|: ! o o . . . .o !
¬ 601.079621| cloud-init¬1288|: ! . .... . !
- 601.079653| cloud-init-1288|: !
                                                1
                                    .. .
- 601.079685| cloud-init-1288|: ! ...
                                                1
- 601.079717| cloud-init-1288|: +-----SHA256|-----+
  601.079748| cloud-init-1288|: Generating public/private ed25519 key pair.
  601.079782| cloud-init-1288|: Your identification has been saved in /etc/ssh/
ssh host ed25519 key
- 601.079814| cloud-init-1288|: Your public key has been saved in /etc/ssh/ssh_
host_ed25519_key.pub
  601.079847| cloud-init-1288|: The key fingerprint is:
-
  601.079879| cloud-init-1288|: SHA256:yWsZ/5+7u7D3SIcd7HYnyajXyeWnt5nQ+ZI3So3b
eN8 root§ubuntu-server
  601.079911| cloud-init-1288|: The key's randomart image is:
  601.079942| cloud-init-1288|: +---ED25519 256|--+
  601.079974| cloud-init-1288|: !
                                                 T
-
- 601.080010| cloud-init-1288|: !
                                                 I
- 601.080042| cloud-init-1288|: !
                                                 1
- 601.080076| cloud-init-1288|: !
                                               . !
                                        . .
- 601.080107| cloud-init-1288|: !
                                        S
                                               0 !
- 601.080139| cloud-init-1288|: !
                                        = 0=++!
  601.080179| cloud-init-1288|: !
                                        + . 0**§=!
                                        . 00+&B%!
  601.080210| cloud-init¬1288|: !
  601.080244| cloud-init-1288|: !
                                         ..o*%/E!
-
  601.080289| cloud-init-1288|: +-----SHA256|-----+
-
  612.293731| cloud-init-2027|: Cloud-init v. 20.2-45-g5f7825e2-Oubuntu1 runnin
-
g 'modules:config' at Thu, 04 Jun 2020 12:06:59 +0000. Up 612.11 seconds.
  612.293866| cloud-init-2027|: Set the following 'random' passwords
-
  612.293940| cloud-init-2027|: installer:wgYsAPzYQbFYqU2X2hYm
ci-info: no authorized SSH keys fingerprints found for user installer.
```

<14>Jun 4 12:07:00 ec2: ###### <14>Jun 4 12:07:00 ec2: ----BEGIN SSH HOST KEY FINGERPRINTS-----<14>Jun 4 12:07:00 ec2: 1024 SHA256:ZBNyksVVYZVhKJeL+PWKpsdUcn21yiceX/DboXQdPq0 root&ubuntu-server (DSA) <14>Jun 4 12:07:00 ec2: 256 SHA256:Bitar9fVHUH2FnYVSJJnldprdAcM5Est0dmRWFTUi8k root§ubuntu-server (ECDSA) <14>Jun 4 12:07:00 ec2: 256 SHA256:yWsZ/5+7u7D3SIcd7HYnyajXyeWnt5nQ+ZI3So3beN8 root§ubuntu-server (ED25519) <14>Jun 4 12:07:00 ec2: 3072 SHA256:kHtkABZwk8AE80fy0KPzTRcYpht4iXdZmJ37Cgi3fJ0 root§ubuntu-server (RSA) <14>Jun 4 12:07:00 ec2: ----END SSH HOST KEY FINGERPRINTS-----###### ----BEGIN SSH HOST KEY KEYS----ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBIXM6t1/ 35ot/aPI59ThIJBzg+qGJJ17+1ZVHfzMEDbsTwpM7e9pstPZUM7W1IHWqDvLQDBm/hGg4u8ZGEqmIMI= root§ubuntu-server ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIN7QtU+en+RGruj2zuxWgkMqLmh+35/GR/0E0D16k4nA root§ubuntu-server ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDJdKT7iUAvSjkUqI1l3fHysE+Gj7ulwGgGjYh639px kcHEbbS3V48eR0Y9BmDISEHfjYXGY2wEH0tGJjNRR0GJhZJVNR+qAqJBioj9d/TwXEgwLP8eAy9aVtJB K1rIylnMQltx/SIhqiymjHLCtKlVoIS4l0frT9FiF54Qi/JeJlwGJIW3W2XqcY90DT0Q5q3PSmlZ8KTR imTf9Fy7WJEPA08b3fimYWsz9enuS/gECEUGV3M1MvrzpAQju27NUE0pSMZHR62IMxGvIjYIu3dUkAzm VYd+ACBBzyhVbiw7W1CQW/ohik3wyosUyi9nJq2Iq0A7kkGH+1XoYq/e4/MoqxhIK/oaiudYAkaCWmP1 r/fBa3hlf0f7mVHvxA3tWZc2wYUxFPTmePvpydP2PSctHMhgboaHrGIY2CdSqg8SUdPKrOE= root§ub untu-server -----END SSH HOST KEY KEYS------ 612.877357| cloud-init-2045|: Cloud-init v. 20.2-45-g5f7825e2-Oubuntul runnin 'modules:final' at Thu, 04 Jun 2020 12:07:00 +0000. Up 612.79 seconds. q 612.877426| cloud-init-2045|: ci-info: no authorized SSH keys fingerprints fo und for user installer. - 612.877468| cloud-init-2045|: Cloud-init v. 20.2-45-g5f7825e2-0ubuntu1 finish ed at Thu, 04 Jun 2020 12:07:00 +0000. Datasource DataSourceNoCloud ¬seed=/var/l ib/cloud/seed/nocloud|¬dsmode=net|. Up 612.87 seconds → 612.877509| cloud-init→2045|: Welcome to Ubuntu Server InstallerÜ - 612.877551| cloud-init-2045|: Above you will find SSH host keys and a random password set for the `installer` user. You can use these credentials to ssh-in a nd complete the installation. If you provided SSH keys in the cloud-init datasou rce, they were also provisioned to the installer user. - 612.877634| cloud-init-2045|: If you have access to the graphical console, li ke TTY1 or HMC ASCII terminal you can complete the installation there too. It is possible to connect to the installer over the network, which might allow the use of a more capable terminal. To connect, SSH to installer§10.222.111.24. The password you should use is "KRuXtz5dURAyPkcjcUvA".

The host key fingerprints are:

RSA SHA256:3IvYMkU05lQSKBx0VZUJMzdtXpz3RJl3dEQsg3UWc54 ECDSA SHA256:xd1xnkBpn49DUbuP8uWro2mu1GM4MtnqR2WEWg1fS30 ED25519 SHA256:Hk3+/4+X7NJBH16/e/6xFhNXsbHBs0vt6i8YEFUepko

Ubuntu Focal Fossa (development branch) ubuntu-server ttyS0

- The next step is now to remotely connect to the install system and to proceed with the subiquity installer.
- Please notice that at the end of the installer boot up process all needed data is provided to proceed with running the installer in a remote ssh shell.

The command to execute locally is:

user@workstation:~\$ ssh installer@10.222.111.24

• And a temporary random password for the installation got created and shared as well, here:

"KRuXtz5dURAyPkcjcUvA"

(Use it without the leading and trailing double quotes.)

```
user@workstation:~$ ssh installer@10.222.111.24
The authenticity of host '10.222.111.24 (10.222.111.24)' can't be established.
ECDSA key fingerprint is
SHA256:xdlxnkBpn49DUbuP8uWro2mu1GM4MtnqR2WEWg1fS3o.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.222.111.24' (ECDSA) to the list of known hosts.
installer@10.222.111.24's password: KRuXtz5dURAyPkcjcUvA
```

• One may now temporarily see some login messages like these:

Welcome to Ubuntu Focal Fossa (development branch) (GNU/Linux 5.4.0-42-generic s390x)

```
* Documentation: https://help.ubuntu.com
 * Management:
                   https://landscape.canonical.com
 * Support:
                   https://ubuntu.com/advantage
 System information as of Wed Jun 3 17:32:10 UTC 2020
                                                              146
 System load:
                 0.0
                           Memory usage: 2%
                                              Processes:
                           Swap usage: 0%
 Usage of /home: unknown
                                             Users logged in: 0
0 updates can be installed immediately.
0 of these updates are security updates.
```

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

• And eventually the initial subiquity installer screen shows up:

Willkommen! Bienvenue! Welcome! ????? ?????????! Welkom!

Use UP, DOWN and ENTER keys to select your language.

[English	>]
[Asturianu	>]
[Cataln	>]
[Hrvatski	>]
[Nederlands	>]
[Suomi	>]
[Francais	>]
[Deutsch	>]
[Magyar	>]
[Latvie?u	>]
[Norsk bokm?l	>]
[Polski	>]
[Espanol	>]

• From here just proceed with the installation as usual ... (I'm leaving some pretty standard screenshots here just to give an example for a basic installation ...)

	Layout:	[English (US)		v]	
	Variant:	[English (US)		v]	
		[Identify keyb	oard]		
		[Done [Back]]		
Zdev setup					
ID		ONLINE NAMES			^
generic-ccw					
0.0.0009			>		
			>		
			>		
0.0.0000			-		
dasd-ockd					
			>		
0.0.0191			>		i
0.0.019d			>		
0.0.019e			>		1
0.0.0200			>		
0.0.0300			>		
0.0.0400			>		
0.0.0592			>		v
		[Continue	1		

[Back

]

• If the list is long, hit the 'End' key that will automatically scroll you down to the bottom of the Z devices list and screen.

^ | | |

Zdev setup			
ID	ONLINE	NAMES	
generic-ccw			
0.0.0009			>
0.0.000c			>
0.0.000d			>
0.0.000e			>
dasd-eckd			
0.0.0190			>
0.0.0191			>
0.0.019d			>
0.0.019e			>
0.0.0200			> < (close)
0.0.0300			> Enable

0.0.0400 0.0.0592

> L

]

^

٧

^

v

[Continue] [Back

Zdev setup		
ID	ONLINE NAM	IES
generic-ccw		
0.0.0009		>
0.0.000c		>
0.0.000d		>
0.0.000e		>
dasd-eckd		
0.0.0190		>
0.0.0191		>
0.0.019d		>
0.0.019e		>
0.0.0200	online dag	sda >
0 0 0300	Untile uas	>
0.0.0000		>
0.0.0 4 00 0 0 0592		~
0.0.0392		
	[Contir	nue l
	[Back]
	Lock	
Zdev setup		
======================================		
dasd-eckd		
0.0.0190		>
0.0.0191		>
0.0.019d		>
0.0.019e		>
0.0.0200	online das	ida >
0.0.0300		>
0.0.0400		>
0.0.0592		>
geth		
0.0.0600:0.0.0601:0.0.0602	end	:600 >
0.0.0603:0.0.0604:0.0.0605	ent	>
dasd-eckd		
0.0.1607		>
	[Contir	nue]
	[Back	1
	2 2000	
Network connections		
Configure at least one inte	rface this s	erver can use to talk to oth
machines, and which prefera	bly provides	s sufficient access for updat
-		
NAME TYPE NOTES		
[enc600 eth -	>]	

static 10.222.111.24/24

02:28:0a:00:00:39 / Unknown Vendor / Unknown Model

		[[Done Back]]	
=	Configure proxy				
-	If this system details here.	requires a proxy	to connect	to the internet,	enter its
	Proxy address:	If you need to u enter the proxy blank.	se a HTTP p information	roxy to access th here. Otherwise,	e outside world, leave this
		The proxy inform of "http://[[use	ation shoul r][:pass]@]	d be given in the host[:port]/".	standard form
		[[Done Back]]	
=	Configure Ubunt	u archive mirror			=====
=	If you use an a	lternative mirror	for Ubuntu	, enter its detai	ls here.
	Mirror address:	http://ports.ub You may provide of the default.	untu.com/ub an archive	untu-ports mirror that will	be used instead

[Done]
[Back]

Guided storage configuration

Configure a guided storage layout, or create a custom one:	
(X) Use an entire disk	
[0X0200 local disk 6.876G	
[] Set up this disk as an LVM group	
[] Encrypt the LVM group with LUKS	
Passphrase:	
Confirm passphrase:	
() Custom storage layout	
[Done] [Back]	
Storage configuration	
FILE SYSTEM SUMMARY	
MOUNT POINT SIZE TYPE DEVICE TYPE [/ 6.875G new ext4 new partition of local disk >]	
AVAILABLE DEVICES	
No available devices	
[Create software RAID (md) >] [Create volume group (LVM) >]	
USED DEVICES	
[Done] [Reset]	
[Back]	
Storage configuration	
FILE SYSTEM SUMMARY	
Confirm destructive action	
<pre></pre>	 .
<pre>You will not be able to return to this or a previous screen once installation has started.</pre>	the
Are you sure you want to continue?	
 [No] [Continue]	

[Reset] [Back] Profile setup Enter the username and password you will use to log in to the system. You can configure SSH access on the next screen but a password is still needed for sudo. Your name: Ed Example Your server's name: 10.222.111.24 The name it uses when it talks to other computers. Pick a username: ubuntu Choose a password: ******* Confirm your password: ******* [Done 1 _____ _____ SSH Setup You can choose to install the OpenSSH server package to enable secure remote access to your server. [] Install OpenSSH server Import SSH identity: [No v] You can import your SSH keys from Github or Launchpad. Import Username: [X] Allow password authentication over SSH [Done] [Back] • It's an nice and convenient new feature to add the users ssh keys during the installation to the system, since that makes the system login password-less already on the initial login! _____ SSH Setup _____ You can choose to install the OpenSSH server package to enable secure remote

access to your server.

[X] Install OpenSSH server

Import SSH identity:	[from Launchpad v] You can import your SSH keys from Github or Launchpac	d.
Launchpad Username:	user Enter your Launchpad username.	
[X]	Allow password authentication over SSH	
	[Done] [Back]	
SSH Setup		
You can choose to ir access to your serve	nstall the OpenSSH server package to enable secure remo er.	te
[Confirm SSH keys	1
 Keys with the fo use them? 	ollowing fingerprints were fetched. Do you want to	
2048 SHA256:joGs (RSA) 521 SHA256:T3Jz> frank.heimes@car	dfW7NbJRkg17sRyXaegoR0iZEdDWdR9Hpbc2KIw user@W520 xvB6K1GzXJpP5NFgX4yXvk0jhhgvbw01F7/fZ2c nonical.com (ECDSA)	
	[Yes] [No]	
	[Done] [Back]	
Featured Server Snap)5	
These are popular sr SPACE, press ENTER t versions available.	naps in server environments. Select or deselect with to see more details of the package, publisher and	
<pre>[] kata-containers [] docker [] mosquitto [] etcd [] stress-ng [] sabnzbd</pre>	Lightweight virtual machines that seamlessly plug into Docker container runtime Eclipse Mosquitto MQTT broker Resilient key-value store by CoreOS A tool to load, stress test and benchmark a computer s SABnzbd	~ ~ ~ ~ ~
[] wormhole [] slcli [] doctl [] keepalived [] juju	get things from one computer to another, safely Python based SoftLayer API Tool. DigitalOcean command line tool High availability VRRP/BFD and load-balancing for Linu Simple, secure and stable devops. Juju keeps complexit	> > > >

[Done] [Back]

Install complete!

configuring raid (mdadm) service	^
installing kernel	
setting up swap	
apply networking config	
writing etc/fstab	
configuring multipath	
updating packages on target system	
configuring pollinate user-agent on target	
updating initramfs configuration	
finalizing installation	
running 'curtin hook'	
curtin command hook	
executing late commands	
final system configuration	
configuring cloud-init	
installing openssh-server	v

[View full log]

Installation complete!

Finished install!	
apply networking config	^
writing etc/fstab	
configuring multipath	
updating packages on target system	
configuring pollinate user-agent on target	
updating initramfs configuration	
finalizing installation	
running 'curtin hook'	
curtin command hook	
executing late commands	
final system configuration	
configuring cloud-init	
installing openssh-server	
restoring apt configuration	
downloading and installing security updates	v

[View full log] [Reboot]

Installation complete!

^
rget

```
[ Connection to 10.222.111.24 closed by remote host.
                                                                                             [Rebooting...]
Connection to 10.222.111.24 closed.
user@workstation:~$
   • Type a 'reset' to clear the screen and to revert it back to the defaults:
     user@workstation:~$ reset
     user@workstation:~$
   • Now remove the old host key, since the system got a new ones during the installation:
     user@workstation:~$ ssh-keygen -f "/home/user/.ssh/known hosts" -R "10.222.111.24"
     # Host 10.222.111.24 found: line 159
     /home/user/.ssh/known_hosts updated.
     Original contents retained as /home/user/.ssh/known_hosts.old
     user@workstation:~$
   • And finally login to the newly installed z/VM guest:
     user@workstation:~$ ssh ubuntu@10.222.111.24
     Warning: Permanently added the ECDSA host key for IP address
     '10.222.111.24' to the list of known hosts.
     Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-42-generic s390x)
       * Documentation: https://help.ubuntu.com
       * Management:
                         https://landscape.canonical.com
       * Support:
                         https://ubuntu.com/advantage
       System information as of Wed 03 Jun 2020 05:50:05 PM UTC
       System load: 0.08
                                      Memory usage: 2%
                                                          Processes:
                                                                           157
       Usage of /: 18.7% of 6.70GB Swap usage:
                                                    0% Users logged in: 0
     0 updates can be installed immediately.
     0 of these updates are security updates.
     The programs included with the Ubuntu system are free software;
     the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.
     Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.
     To run a command as administrator (user "root"), use "sudo <command>".
     See "man sudo root" for details.
     ubuntu@10.222.111.24:~$ uptime
      17:50:09 up 1 min, 1 user, load average: 0.08, 0.11, 0.05
     ubuntu@10.222.111.24:~$ lsb release -a
     No LSB modules are available.
     Distributor ID: Ubuntu
     Description: Ubuntu 20.04.3 LTS
                 20.04
     Release:
     Codename: focal
     ubuntu@10.222.111.24:~$ uname -a
     Linux 10.222.111.24 5.4.0-42-generic #30-Ubuntu SMP Wed Aug 05 16:57:22 UTC 2020 s390x s390x s390x GNU/Linux
     ubuntu@10.222.111.24:~$ exit
     logout
     Connection to 10.222.111.24 closed.
     user@workstation:~$
```

v

```
Done !
```

Interactive live server installation on IBM Z LPAR (s390x)

Doing a manual live installation like described here - means without specifying a parmfile - is supported since Ubuntu Server LTS 20.04.3 ('focal') and any newer release, like 20.10 ('groovy').

- The following guide assumes that an FTP installation server is in place, that can be used by the 'Load from Removable Media and Server' task of the HMC.
- Download the 'focal daily live image' from here (later 20.04.3 image): http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso and extract and make it available via your FTP-hosted installation server.
- Open the IBM Z Hardware Management Console (HMC) and navigate to 'Systems Management' --> '<your system indicated by it's serial number>'
- Select the LPAR that you are going to install Ubuntu Server on; in this example LPAR 's1lp11' is used
- Now select menu: 'Recovery' --> 'Load from Removable Media or Server' task
- And fill out the 'Load from Removable Media or Server' form like follows (adapt the settings to your particular installation environment):

```
Load from Removable Media, or Server - <serial>:sllpl1
Use this task to load operating system software or utility programs
from a CD / DVD-ROM or a server that can be accessed using FTP.
Select the source of the software:
o Hardware Management Console CD / DVD-ROM
o Hardware Management Console CD / DVD-ROM and assign for operating system use
o Hardware Management Console USB flash memory drive
o Hardware Management Console USB flash memory drive
o Hardware Management Console USB flash memory drive and assign for operating system use
* FTP Source
Host computer: install-server
User ID: ftpuser
Password: *******
Account (optional):
File location (optional): ubuntu-live-server-20.04.3/boot
```

You may need to adjust the File location according to your install server environment.

• Confirm the entered data

```
Load from Removable Media or Server - Select Software to Install -
<serial>:sllp11
Select the software to install.
Select Name Description
* ubuntu-live-server-20.04.3/boot/ubuntu.ins Ubuntu for IBM Z (default kernel)
```

• Confirm again that jobs might be canceled if proceeding

Load from Removable Media or Server Task Confirmation -<serial>:sllpll Load will cause jobs to be cancelled. Do you want to continue with this task? ACT33501

• And confirm a last time that it's understood that the task is disruptive:

Disruptive Task Confirmation : Load from Removable Media or Server <serial>:s1lp11

Attention: The Load from Removable Media or Server task is disruptive.

Executing the Load from Removable Media or Server task may adversely affect the objects listed below. Review the confirmation text for each object before continuing with the Load from Removable Media or Server task.

Objects that will be affected by the Load from Removable Media or Server task

System NameTypeOS NameStatusConfirmation Text<serial>:sllpl1ImageOperatingLoad from Removable Mediaor Server causes operations to be disrupted, since the target iscurrently in use and operating normally.

Do you want to execute the Load from Removable Media or Server task?

• The 'Load from Removable media or Server' task is now executed ...

Load from Removable media or Server Progress - P00B8F67:S1LPB Turn on context sensitive help. Function duration time: 00:55:00 Elapsed time: 00:00:04 Select Object Name Status * <serial> s1lp11 Please wait while the image is being loaded.

• This may take a moment, but you will soon see:

```
Load from Removable media or Server Progress - <serial>:sllpl1
Function duration time: 00:55:00
Elapsed time: 00:00:21
Select Object Name Status
* <serial> sllpl1 Success
```

 Close the 'Load from Removable media or Server' task and open the console aka 'Operating System Messages' instead.

And in case no parmfile got configured or provided, one will find the following lines in the 'Operating System Messages' task:

```
Operating System Messages - <serial>:s1lp11
```

Message

```
Unable to find a medium container a live file system
Attempt interactive netboot from a URL?
yes no (default yes):
```

- So one will now by default land in the interactive network configuration menu (again, only if no parmfile got prepared with sufficient network configuration information).
- Proceed with the interactive network configuration, here in this case in a VLAN environment:

```
Unable to find a medium container a live file system
Attempt interactive netboot from a URL?
yes no (default yes):
yes
Available geth devices:
0.0.c000 0.0.c003 0.0.c006 0.0.c009 0.0.c00c 0.0.c00f
zdev to activate (comma separated, optional):
0.0.c000
QETH device 0.0.c000:0.0.c001:0.0.c002 configured
Two methods available for IP configuration:
* static: for static IP configuration
* dhcp: for automatic IP configuration
static dhcp (default 'dhcp'):
static
ip:
10.222.111.11
gateway (default 10.222.111.1):
10.222.111.1
dns (default 10.222.111.1):
10.222.111.1
vlan id (optional):
1234
http://cdimage.ubuntu.com/ubuntu/releases/20.04.3/release/ubuntu-20.04.3-live-server-s390x.iso (default)
url:
ftp://10.11.12.2:21/ubuntu-live-server-20.04.3/ubuntu-20.04.3-live-server-s390x.iso
```

http_proxy (optional):

• After the last interactive step here, that this is about an optional proxy configuration, the installer will complete it's boot-up process:

```
Configuring networking...
IP-Config: encc000.1234 hardware address 3e:00:10:55:00:ff mtu 1500
IP-Config: encc000.1234 guessed broadcast address 10.222.111.255
IP-Config: encc000.1234 complete:
address: 10.222.111.11
                    broadcast: 10.222.111.255 netmask: 255.255.255.0
                    dns0 : 10.222.111.1 dns1 : 0.0.0.0
gateway: 10.222.111.1
rootserver: 0.0.0.0 rootpath:
filename :
Connecting to 10.11.12.2:21 (10.11.12.2:21)
                                              | 72.9M 0:00:08 ETA
focal-live-server-s 10% |***
focal-live-server-s 25% |*******
                                              | 168M 0:00:05 ETA
| 279M 0:00:04 ETA
| 390M 0:00:02 ETA
.
focal-live-server-s 75% |***************************

      focal-live-server-s
      75%
      |************************
      |
      501M
      0:00:01
      ETA

      focal-live-server-s
      89%
      |**********************************
      |
      595M
      0:00:00
      ETA

ip: RTNETLINK answers: File exists
no search or nameservers found in /run/net-encc000.1234.conf / run/net-*.conf /run/net6-*.conf
[ 399.808930] /dev/loop3: Can't open blockdev
[[0;1;31m SKIP [0m] Ordering cycle found, skipping [0;1;39mLogin Prompts[0m
[ 401.547705] systemd[1]: multi-user.target: Job getty.target/start deleted to
break ordering cycle starting with multi-user.target/start
[ 406.241972] cloud-init[1321]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntul running
'init-local' at Wed, 03 Jun 2020 17:07:39 +0000. Up 406.00 seconds.
[ 407.025557] cloud-init[1348]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntul running
'init' at Wed, 03 Jun 2020 17:07:40 +0000. Up 406.87 seconds.
[ 407.025618] cloud-init[1348]: ci-info: ++++Net device info++++
[ 407.025658] cloud-init[1348]: ci-info: +-----+----+----+-----+
-----+
[ 407.025696] cloud-init[1348]: ci-info: | Device | Up |
                                                            Addr
ess | Mask | Scope | Hw-Address |
-----+
[ 407.025766] cloud-init[1348]: ci-info: | encc000 | True | fe80::3ca7:10f
f:fea5:c69e/64 | . | link | 72:5d:0d:09:ea:76 |
[ 407.025802] cloud-init[1348]: ci-info: | encc000.1234 | True |
                                                          10.245.
236.11 | 255.255.255.0 | global | 72:5d:0d:09:ea:76 |
[ 407.025837] cloud-init[1348]: ci-info: | encc000.1234 | True | fe80::3ca7:10f
f:fea5:c69e/64 | . | link | 72:5d:0d:09:ea:76 |
[ 407.025874] cloud-init[1348]: ci-info: | lo | True |
                                                            127.0
.0.1 | 255.0.0.0 | host |
                                      .
                                             [ 407.025909] cloud-init[1348]: ci-info: |
                                     lo
                                             | True |
                                                            ::1/
128 | . | host |
                                      .
                                             [ 407.025944] cloud-init[1348]: ci-info: +-----+----+-----+------
----+
[ 407.025982] cloud-init[1348]: ci-info: ++++++++++Route I
[ 407.026017] cloud-init[1348]: ci-info: +-----+------
+----+
[ 407.026072] cloud-init[1348]: ci-info: | Route | Destination | Gateway
   Genmask | Interface | Flags |
Т
[ 407.026107] cloud-init[1348]: ci-info: +----+------
+----+
[ 407.026141] cloud-init[1348]: ci-info: | 0 | 0.0.0.0 | 10.222.111.1
   0.0.0.0 | encc000.1234 | UG |
407.026176] cloud-init[1348]: ci-info: | 1 | 10.222.111.0 | 0.0.0.0
ſ
```

| 255.255.255.0 | encc000.1234 | U | +----+ [407.026246] cloud-init[1348]: ci-info: +++++++++++++++Route IPv6 info+++ [407.026280] cloud-init[1348]: ci-info: +-----+-----+-----+-----+-----+ ----+ [407.026315] cloud-init[1348]: ci-info: | Route | Destination | Gateway | Int erface | Flags | [407.026355] cloud-init[1348]: ci-info: +-----+-----+-----+-----+ ----+ [407.026390] cloud-init[1348]: ci-info: | 1 | fe80::/64 | :: | en cc000 | U | [407.026424] cloud-init[1348]: ci-info: | 2 | fe80::/64 | :: | encc 000.1234 | U | [407.026458] cloud-init[1348]: ci-info: | 4 | en local :: | U | cc000 [407.026495] cloud-init[1348]: ci-info: | 5 | local :: | encc 000.1234 | U | [407.026531] cloud-init[1348]: ci-info: | 6 | ff00::/8 | :: 1 en cc000 | U | [407.026566] cloud-init[1348]: ci-info: | 7 | ff00::/8 | :: | encc 000.1234 | U [407.026600] cloud-init[1348]: ci-info: +-----+-----+-----+------ - - - - - - - + - - - - - + [407.883058] cloud-init[1348]: Generating public/private rsa key pair. [407.883117] cloud-init[1348]: Your identification has been saved in /etc/ssh/ ssh host rsa key [407.883154] cloud-init[1348]: Your public key has been saved in /etc/ssh/ssh host_rsa_key.pub [407.883190] cloud-init[1348]: The key fingerprint is: [407.883232] cloud-init[1348]: SHA256:KX5cHC4YL9dXpvhnP6eSfS+J/zmKgg9zdlEzaEb+ RTA root@ubuntu-server [407.883267] cloud-init[1348]: The key's randomart image is: [407.883302] cloud-init[1348]: +---[RSA 3072]----+ [407.883338] cloud-init[1348]: | . E.. I [407.883374] cloud-init[1348]: | 0.01 [407.883408] cloud-init[1348]: | .= +0. [407.883443] cloud-init[1348]: | + =000++ ſ 407.883478] cloud-init[1348]: | + S *.o. [407.883512] cloud-init[1348]: | . = 0 0.[407.883546] cloud-init[1348]: | .0+0 ..+0. | [407.883579] cloud-init[1348]: | 0=.. =0+++ [407.883613] cloud-init[1348]: | ++*0| [407.883648] cloud-init[1348]: +----[SHA256]----+ [407.883682] cloud-init[1348]: Generating public/private dsa key pair. [407.883716] cloud-init[1348]: Your identification has been saved in /etc/ssh/ ssh host dsa key [407.883750] cloud-init[1348]: Your public key has been saved in /etc/ssh/ssh host_dsa_key.pub [407.883784] cloud-init[1348]: The key fingerprint is: [407.883817] cloud-init[1348]: SHA256:xu3vlG1BReKDy3DsuMZc/lg5y/+nhzlEmLDk/qFZ AmO root@ubuntu-server [407.883851] cloud-init[1348]: The key's randomart image is: 407.883905] cloud-init[1348]: +---[DSA 1024]----+ [[407.883941] cloud-init[1348]: | [407.883975] cloud-init[1348]: | 0.00 [407.884008] cloud-init[1348]: | +.0+0+ | [407.884042] cloud-init[1348]: | ...E*oo.. | [407.884076] cloud-init[1348]: | S+o =.. | [407.884112] cloud-init[1348]: | . +0+00.0 [407.884145] cloud-init[1348]: | **+0*0 | [407.884179] cloud-init[1348]: | .00.*+00

```
[ 407.884212] cloud-init[1348]: |
                                          .+ ===|
[ 407.884246] cloud-init[1348]: +----[SHA256]----+
[ 407.884280] cloud-init[1348]: Generating public/private ecdsa key pair.
[ 407.884315] cloud-init[1348]: Your identification has been saved in /etc/ssh/
ssh host ecdsa key
[ 407.884352] cloud-init[1348]: Your public key has been saved in /etc/ssh/ssh
host_ecdsa_key.pub
[ 407.884388] cloud-init[1348]: The key fingerprint is:
ſ
  407.884422] cloud-init[1348]: SHA256:P+hBF3fj/pu6+0KaywUYii3Lyuc09Za9/a2elCD0
gdE root@ubuntu-server
[ 407.884456] cloud-init[1348]: The key's randomart image is:
[ 407.884490] cloud-init[1348]: +---[ECDSA 256]---+
[ 407.884524] cloud-init[1348]: |
[ 407.884558] cloud-init[1348]: |
                                               [ 407.884591] cloud-init[1348]: |
                                       ..E.o |
                                   0.0000.|
[ 407.884625] cloud-init[1348]: |
  407.884660] cloud-init[1348]: |
ſ
                                   o +S.+.. .
[ 407.884694] cloud-init[1348]: |
                                   . +..*00.+ . |
[ 407.884728] cloud-init[1348]: |
                                   = 0+=.+.+
[ 407.884762] cloud-init[1348]: | . o.....++o oo|
[ 407.884795] cloud-init[1348]: | oo. . +.*@*+|
[ 407.884829] cloud-init[1348]: +----[SHA256]----+
[ 407.884862] cloud-init[1348]: Generating public/private ed25519 key pair.
[ 407.884896] cloud-init[1348]: Your identification has been saved in /etc/ssh/
ssh host ed25519 key
[ 407.884930] cloud-init[1348]: Your public key has been saved in /etc/ssh/ssh
host ed25519 key.pub
[ 407.884966] cloud-init[1348]: The key fingerprint is:
[ 407.884999] cloud-init[1348]: SHA256:CbZpkR9eFHuB1sCDZwSdSdwJzy9FpsIWRIyc9ers
hZ0 root@ubuntu-server
[ 407.885033] cloud-init[1348]: The key's randomart image is:
[ 407.885066] cloud-init[1348]: +--[ED25519 256]--+
[ 407.885100] cloud-init[1348]: | ../%X..o |
                                     .=0&*+=
[ 407.885133] cloud-init[1348]: |
                                               [ 407.885167] cloud-init[1348]: |
                                   = .+*.* .
                                              - 1
[ 407.885200] cloud-init[1348]: |
                                   . B = + o
[ 407.885238] cloud-init[1348]: |
                                    + S . . .
[ 407.885274] cloud-init[1348]: |
                                    .
                                      000
[ 407.885308] cloud-init[1348]: |
                                        + E
ſ
  407.885345] cloud-init[1348]: |
                                        . .
[ 407.885378] cloud-init[1348]: |
                                               .
[ 407.885420] cloud-init[1348]: +----[SHA256]----+
[ 418.521933] cloud-init[2185]: Cloud-init v. 20.2-45-g5f7825e2-Oubuntul runnin
g 'modules:config' at Wed, 03 Jun 2020 17:07:52 +0000. Up 418.40 seconds.
[ 418.522012] cloud-init[2185]: Set the following 'random' passwords
[ 418.522053] cloud-init[2185]: installer:C7BZrW76s4mJzmpf4eUy
ci-info: no authorized SSH keys fingerprints found for user installer.
<14>Jun 3 17:07:52 ec2:
######
<14>Jun 3 17:07:52 ec2: ----BEGIN SSH HOST KEY FINGERPRINTS-----
<14>Jun 3 17:07:52 ec2: 1024 SHA256:xu3vlG1BReKDy3DsuMZc/lg5y/+nhzlEmLDk/qFZAm0
root@ubuntu-server (DSA)
<14>Jun 3 17:07:52 ec2: 256 SHA256:P+hBF3fj/pu6+0KaywUYii3Lyuc09Za9/a2elCD0gdE
root@ubuntu-server (ECDSA)
<14>Jun 3 17:07:52 ec2: 256 SHA256:CbZpkR9eFHuB1sCDZwSdSdwJzy9FpsIWRIyc9ershZ0
root@ubuntu-server (ED25519)
<14>Jun 3 17:07:52 ec2: 3072 SHA256:KX5cHC4YL9dXpvhnP6eSfS+J/zmKgg9zdlEzaEb+RTA
root@ubuntu-server (RSA)
<14>Jun 3 17:07:52 ec2: ----END SSH HOST KEY FINGERPRINTS-----
######
----BEGIN SSH HOST KEY KEYS-----
```

ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBC2zp4Fq r1+NJ0IEQIISbX+EzeJ6ucXSLi2xEvurgwq8iMYT6yY0XB0Pc/XzeFa6vBCDZk3SSSW6Lq83y7VmdRQ= root@ubuntu-server

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIJFzgips94nJNoR4QumiyqlJoSlZ48P+NVrd7zgD5k4T
root@ubuntu-server

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQChKo060715FAjd6ImK7qZbWnL/cpgQ2A2gQEqFN0+1
joF/41ygxuw5aG0IQ0bWFpV9jDsMF5z4qHKzX8tFCpKC0s4uR8QBxh1dDm4wcwcgtAfVLqh7S4/R9Sqa
IFnkCzxThhNeMarcRrutY0mIzspmCg/QvfE1wrXJz1+Rt0J7GiuHHqpm76fX+6ZF1BYhkA87dXQiID2R
yUubSXKGg0Ntz1gSzPqD3GB+HxRHHHLT5/Xq+njPq8jIUpqSoHtkBupsyVmcD9gDbz6vng2PuBHwZP9X
17Qty0wxddxk4xIXaTup4g8bH1oF/czsWqVxNdfB7XqzR0FU0D9rMIB+DwBihsmH1kRik4wwLi6IH4hu
xrykKvfb1xcZe65kR42oDI7JbBwxvxGr0Kx8DrEXnBp0WozS0IDm2ZPh3ci/0uCJ4LTItByyCfAe/gyR
5si4SkmXrIXf5BnErZRgyJnfxKXmsFaSh7wf15w6GmsgzyD9sI2jES9+4By32ZzY0lDpi0s= root@ub
untu-server

----END SSH HOST KEY KEYS-----

[418.872320] cloud-init[2203]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntul runnin
g 'modules:final' at Wed, 03 Jun 2020 17:07:52 +0000. Up 418.79 seconds.

[418.872385] cloud-init[2203]: ci-info: no authorized SSH keys fingerprints fo und for user installer.

[418.872433] cloud-init[2203]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntul finish ed at Wed, 03 Jun 2020 17:07:52 +0000. Datasource DataSourceNoCloud [seed=/var/l ib/cloud/seed/nocloud][dsmode=net]. Up 418.86 seconds

[418.872484] cloud-init[2203]: Welcome to Ubuntu Server Installer!

[418.872529] cloud-init[2203]: Above you will find SSH host keys and a random password set for the `installer` user. You can use these credentials to ssh-in a nd complete the installation. If you provided SSH keys in the cloud-init datasou rce, they were also provisioned to the installer user.

[418.872578] cloud-init[2203]: If you have access to the graphical console, li ke TTY1 or HMC ASCII terminal you can complete the installation there too.

It is possible to connect to the installer over the network, which might allow the use of a more capable terminal.

To connect, SSH to installer@10.222.111.11.

The password you should use is "C7BZrW76s4mJzmpf4eUy".

The host key fingerprints are:

RSA SHA256:KX5cHC4YL9dXpvhnP6eSfS+J/zmKgg9zdlEzaEb+RTA ECDSA SHA256:P+hBF3fj/pu6+0KaywUYii3Lyuc09Za9/a2elCD0gdE ED25519 SHA256:CbZpkR9eFHuB1sCDZwSdSdwJzy9FpsIWRIyc9ershZ0

Ubuntu Focal Fossa (development branch) ubuntu-server sclp_line0 ubuntu-server login:

- At this point one can proceed with the regular installation either by using Recovery --> Integrated ASCII Console or or with a remote ssh session.
- If the 'Integrated ASCII Console' got opened (and maybe 'F3' was hit to refresh the task), the initial subiquity installation screen is presented, that looks like this:

Willkommen! Bienvenue! Welcome! ????? ????????! Welkon	n! [Help]
Use UP, DOWN and ENTER keys to select your language.	
[English	>]
[Asturianu	>]
[Cataln	>]
[Hrvatski	>]
[Nederlands	>]
[Suomi	>]
[Francais	>]
[Deutsch	>]
[Magyar	>]

[Latvie?u	>]
[Norsk bokm?l	>]
[Polski	>]
[Espanol	>]

- But since the user experience is nicer in a remote ssh session, it's recommended to use that. However, with certain network environments it will just not be possible to go with a remote shell, and the 'Integrated ASCII Console' will be the only option.
- Please notice that at the end of the installer boot up process **all** needed information is provided to proceed with a remote shell.
- The command to execute locally is:

user@workstation:~\$ ssh installer@10.222.111.11

• A temporary random password for the installation got created and shared as well:

"C7BZrW76s4mJzmpf4eUy"

(Use it without the leading and trailing double quotes.)

• Hence the remote session for the installer can be opened by:

```
user@workstation:~$ ssh installer@10.222.111.11
The authenticity of host '10.222.111.11 (10.222.111.11)' can't be established.
ECDSA key fingerprint is SHA256:P+hBF3fj/pu6+0KaywUYii3Lyuc09Za9/a2elCD0gdE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.222.111.11' (ECDSA) to the list of known hosts.
installer@10.222.111.11's password: C7BZrW76s4mJzmpf4eUy
```

• One may swiftly see some login messages like the following ones:

Welcome to Ubuntu Focal Fossa (development branch) (GNU/Linux 5.4.0-42-generic s390x)

- * Documentation: https://help.ubuntu.com
- * Management: https://landscape.canonical.com
- * Support: https://ubuntu.com/advantage

System information as of Wed Jun 3 17:32:10 UTC 2020

System load:0.0Memory usage:2%Processes:146Usage of /home:unknownSwap usage:0%Users logged in:0

0 updates can be installed immediately.0 of these updates are security updates.

The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

Eventually the initial subiquity installer screen shows up:

Willkommen! Bienvenue! Welcome! ????? ????????! Welkom!

Use UP, DOWN and ENTER keys to select your language.

English	>]
Asturianu	>]
Cataln	>]
Hrvatski	>]
Nederlands	>]
Suomi	>]
Francais	>]
Deutsch	>]

>]
>]
>]
>]
>]

• From here just proceed with the installation as usual ...

(I'm leaving some pretty standard screen shots here just to give an example for a basic installation $\ldots)$

Layout: [English (US) v]

Variant: [English (US) v]

[Identify keyboard]

[Done] [Back]

Zdev setup		
ID	ONLINE NAMES	
dasd-eckd		
0.0.1600	>	
0.0.1601	>	
0.0.1602	>	
0.0.1603	>	
0.0.1604	>	
0.0.1605	>	
0.0.1606	>	
0.0.1607	>	
0.0.1608	>	
0.0.1609	>	
0.0.160a	>	
0.0.160b	>	
0.0.160c	>	
0.0.160d	>	
	[Continue]	
	[Back]	
Zdev setup		
ID	ONLINE NAMES	
dasd-eckd		
0.0.1600	>	
0 0 1601		

0.0.1602		> Enable
0.0.1603		> Disable
0.0.1604		>
0.0.1605		>
0.0.1606		>
0.0.1607		>
0.0.1608		>
0.0.1609		>
0.0.160a		>
0.0.160b		>
0.0.160c		>
0.0.160d		> V
	[Continue]
	[Back]

	Z	d	e	v		s	e	t	u	р			
==	=	=	=	=	=	=	=	=	=	=	=	=	

 	 _
 	 _

ID	ONLINE	NAMES		^
dasd-eckd				Ι
0.0.1600			>	
0.0.1601	online	dasda	>	
0.0.1602			>	
0.0.1603			>	
0.0.1604			>	
0.0.1605			>	
0.0.1606			>	
0.0.1607			>	
0.0.1608			>	
0.0.1609			>	
0.0.160a			>	
0.0.160b			>	
0.0.160c			>	
0.0.160d			>	v

[Continue]

- [Back]
- One may hit the 'End' key here, that will automatically scroll down to the bottom of the Z devices list and screen:

=				
	Zdev setup			
=		>	^	
	0.0.fle0:0.0.fle1	>		
	0.0.fle2:0.0.fle3	>		
	0.0.fle4:0.0.fle5	>		
	0.0.fle6:0.0.fle7	>		
	0.0.fle8:0.0.fle9	>		
	0.0.flea:0.0.fleb	>		
	0.0.flec:0.0.fled	>		
	0.0.flee:0.0.flef	>		
	0.0.f1f0:0.0.f1f1	>		
	0.0.f1f2:0.0.f1f3	>		
	0.0.f1f4:0.0.f1f5	>		
	0.0.f1f6:0.0.f1f7	>		
	0.0.f1f8:0.0.f1f9	>		
	0.0.flfa:0.0.flfb	>		
	0.0.flfc:0.0.flfd	>		
	0.0.flfe:0.0.flff	>	v	

	[Continue] [Back]
Network connect	tions
Configure at le machines, and v	which preferably provides sufficient access for updates.
NAME [encc000 72:00:bb:00:a	TYPE NOTES eth - >] aa:11 / Unknown Vendor / Unknown Model
[encc000.1234 static VLAN 1234 on	vlan - >] 10.222.111.11/24 interface encc000
[Create bond >	>]
	[Continue] [Back]
• Depending of In that case just ignore a (If you wait	on the installer version you are using you may face a little bug here. the button will be named 'Continue without network', but the network is there. If you see that and continue long enough the label will be refreshed an corrected.)
Configure proxy	/
If this system details here.	requires a proxy to connect to the internet, enter its
Proxy address:	If you need to use a HTTP proxy to access the outside world, enter the proxy information here. Otherwise, leave this blank.
	The proxy information should be given in the standard form of "http://[[user][:pass]@]host[:port]/".
	[Done] [Back]
Configure Ubun	tu archive mirror
If you use an a	alternative mirror for Ubuntu, enter its details here.
Mirror address	: http://ports.ubuntu.com/ubuntu-ports You may provide an archive mirror that will be used instead of the default.

[Done] [Back]	
Guided storage configuration	
Configure a guided storage layout, or create a custom one:	
(X) Use an entire disk	
[0X1601 local disk 6.877G	v]
[] Set up this disk as an LVM group	
[] Encrypt the LVM group with LUKS	
Passphrase:	
Confirm passphrase:	
() Custom storage layout	
[Done] [Back]	
Storage configuration	
FILE SYSTEM SUMMARY	^
MOUNT POINT SIZE TYPE DEVICE TYPE [/ 6.875G new ext4 new partition of local disk >]	
AVAILABLE DEVICES	
No available devices	
[Create software RAID (md) >] [Create volume group (LVM) >]	
USED DEVICES	v
[Done] [Reset] [Back]	

FILE SYSTEM SUMMARY

 Confirm destructive action — Selecting Continue below will begin the installation process and result in the loss of data on the disks selected to be formatted. You will not be able to return to this or a previous screen once the installation has started. Are you sure you want to continue? [No] [Continue] [Reset] [Back] _____ -------Profile setup Enter the username and password you will use to log in to the system. You can configure SSH access on the next screen but a password is still needed for sudo. Your name: Ed Example Your server's name: s1lp11 The name it uses when it talks to other computers. Pick a username: ubuntu Choose a password: ******* Confirm your password: ******* [Done] SSH Setup You can choose to install the OpenSSH server package to enable secure remote access to your server. [] Install OpenSSH server Import SSH identity: [No v] You can import your SSH keys from Github or Launchpad.

Import Username:

[X] Allow password authentication over SSH

[Done] [Back]

• It's an nice and convenient new feature to add the users ssh keys during the installation to the system, since that makes the system login password-less already on the initial login!

You can choose to ins access to your serve	tall the OpenSSH serve	r package to enable sec	ure remote
[X]	Install OpenSSH serve	r	
Import SSH identity:	[from Launchpad v] You can import your S	SH keys from Github or	Launchpad.
Launchpad Username:	user Enter your Launchpad	username.	
[X]	Allow password authen	tication over SSH	
	[Done [Back]	
SSH Setup			
SSH Setup You can choose to ins access to your serve	tall the OpenSSH serve	r package to enable sec	ure remote
SSH Setup You can choose to ins access to your serven	tall the OpenSSH serve . Confirm SSH k	eys —	ure remote
SSH Setup You can choose to ins access to your server Keys with the for use them?	tall the OpenSSH serve Confirm SSH k	ere fetched. Do you want	ure remote
SSH Setup You can choose to ins access to your server Keys with the fol use them? 2048 SHA256:joGsd (RSA) 521 SHA256:T31zx)	tall the OpenSSH serve Confirm SSH k Confirm SSH k Nowing fingerprints we	r package to enable sec reys re fetched. Do you want corviZEdDWdR9Hpbc2KIw us	ure remote to er@W520
SSH Setup You can choose to ins access to your server Keys with the for use them? 2048 SHA256:joGso (RSA) 521 SHA256:T3Jzxv ed.example@acme.o	tall the OpenSSH serve Confirm SSH k Confirm SSH k lowing fingerprints we miamcaoincinaäonnväine B6K1Gzidvoidhoidsaoica com (ECDSA)	er package to enable sec reys re fetched. Do you want orviZEdDWdR9Hpbc2KIw us k0jhhgvbw01F7/fZ2c	ure remote to er@W520
SSH Setup You can choose to ins access to your server Keys with the fol use them? 2048 SHA256:joGso (RSA) 521 SHA256:T3Jzxv ed.example@acme.o	tall the OpenSSH serve Confirm SSH k Confirm SSH k lowing fingerprints we miamcaoincinaäonnväine /B6K1Gzidvoidhoidsaoica com (ECDSA) [Yes	r package to enable sec reys ere fetched. Do you want corviZEdDWdR9Hpbc2KIw us k0jhhgvbw01F7/fZ2c	ure remote to er@W520
SSH Setup You can choose to ins access to your server Keys with the fol use them? 2048 SHA256:joGso (RSA) 521 SHA256:T3Jzxv ed.example@acme.o	tall the OpenSSH serve Confirm SSH k Confirm SSH k lowing fingerprints we miamcaoincinaäonnväine /B6K1Gzidvoidhoidsaoica com (ECDSA) [Yes [No	r package to enable sec reys ere fetched. Do you want porviZEdDWdR9Hpbc2KIw us k0jhhgvbw01F7/fZ2c]]	ure remote to er@W520
SSH Setup You can choose to ins access to your server Keys with the fol use them? 2048 SHA256:joGsd (RSA) 521 SHA256:T3Jzxv ed.example@acme.d	stall the OpenSSH serve 	<pre>ir package to enable sec eys are fetched. Do you want corviZEdDWdR9Hpbc2KIw us k0jhhgvbw01F7/fZ2c]]</pre>	ure remote

These are popular snaps in server environments. Select or deselect with SPACE, press ENTER to see more details of the package, publisher and versions available.

[]	kata-containers	Lightweight virtual machines that seamlessly plug into	>
[]	docker	Docker container runtime	>
[]	mosquitto	Eclipse Mosquitto MQTT broker	>
[]	etcd	Resilient key-value store by CoreOS	>
[]	stress-ng	A tool to load, stress test and benchmark a computer \boldsymbol{s}	>
[]	sabnzbd	SABnzbd	>
[]	wormhole	get things from one computer to another, safely	>
[]	slcli	Python based SoftLayer API Tool.	>
[]	doctl	DigitalOcean command line tool	>
[]	keepalived	High availability VRRP/BFD and load-balancing for Linu	>
[]	juju	Simple, secure and stable devops. Juju keeps complexit	>

[Done] [Back

]

Install complete!

configuring raid (mdadm) service	^
installing kernel	
setting up swap	
apply networking config	
writing etc/fstab	
configuring multipath	
updating packages on target system	
configuring pollinate user-agent on target	
updating initramfs configuration	
finalizing installation	
running 'curtin hook'	
curtin command hook	
executing late commands	
final system configuration	
configuring cloud-init	
installing openssh-server \	v

[View full log]

_____ Installation complete! _____

Finished install!	
apply networking config	^
writing etc/fstab	
configuring multipath	
updating packages on target system	
configuring pollinate user-agent on target	
updating initramfs configuration	
finalizing installation	
running 'curtin hook'	
curtin command hook	
executing late commands	
final system configuration	
configuring cloud-init	
installing openssh-server	
<pre>restoring apt configuration</pre>	
downloading and installing security updates	v

[View full log] [Reboot]

Installation complete!

Finished install!	
apply networking config	^
writing etc/fstab	
configuring multipath	
updating packages on target system	
configuring pollinate user-agent on target	
updating initramfs configuration	
finalizing installation	
running 'curtin hook'	
curtin command hook	
executing late commands	
final system configuration	
<pre>configuring cloud-init</pre>	
installing openssh-server	
<pre>restoring apt configuration</pre>	
downloading and installing security updates	v

[Connection to 10.222.111.11 closed by remote host. Connection to 10.222.111.11 closed. user@workstation:~\$ [Rebooting...]

• Now type 'reset' to clear the screen and to reset it to it's defaults.

user@workstation:~\$ reset

user@workstation:~\$

• Before proceeding one needs to remove to old temporary host key of the target system, since it was only for the use during installation:

user@workstation:~\$ ssh-keygen -f "/home/user/.ssh/known_hosts" -R "sllpll"
Host sllpl1 found: line 159
/home/user/.ssh/known_hosts updated.
Original contents retained as /home/user/.ssh/known_hosts.old
user@workstation:~\$

• And assuming the post-installation reboot is done, one can now login:

user@workstation:~\$ ssh ubuntu@sllp11
Warning: Permanently added the ECDSA host key for IP address
'10.222.111.11' to the list of known hosts.
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-42-generic s390x)

- * Documentation: https://help.ubuntu.com
- * Management: https://landscape.canonical.com
- * Support: https://ubuntu.com/advantage

System information as of Wed 03 Jun 2020 05:50:05 PM UTC

System load: 0.08Memory usage: 2%Processes:157Usage of /:18.7% of 6.70GBSwap usage:0%Users logged in: 0

0 updates can be installed immediately.0 of these updates are security updates.

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details.

```
ubuntu@s1lp11:~$ uptime
17:50:09 up 1 min, 1 user, load average: 0.08, 0.11, 0.05
ubuntu@s1lp11:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:
                Ubuntu 20.04.3 LTS
Release:
            20.04
Codename:
            focal
ubuntu@s1lp11:~$ uname -a
Linux sllp11 5.4.0-42-generic #30-Ubuntu SMP Wed Aug 05 16:57:22 UTC 2020 s390x s390x s390x GNU/Linux
ubuntu@s1lp11:~$ exit
logout
Connection to s1lp11 closed.
user@workstation:~$
```

Done !

Automated Server Installs

Introduction

The server installer for 20.04 supports a new mode of operation: automated installation, autoinstallation for short. You might also know this feature as unattended or handsoff or preseeded installation.

Autoinstallation lets you answer all those configuration questions ahead of time with an *autoinstall config* and lets the installation process run without any interaction.

Differences from debian-installer preseeding

preseeds are the way to automate an installer based on debian-installer (aka d-i).

autoinstalls for the new server installer differ from preseeds in the following main ways:

- the format is completely different (cloud-init config, usually yaml, vs debconf-set-selections format)
- when the answer to a question is not present in a preseed, d-i stops and asks the user for input. autoinstalls are not like this: by default, if there is any autoinstall config at all, the installer takes the default for any unanswered question (and fails if there is no default).
 - You can designate particular sections in the config as "interactive", which means the installer will still stop and ask about those.

Providing the autoinstall config

The autoinstall config is provided via cloud-init configuration, which is almost endlessly flexible. In most scenarios the easiest way will be to provide user-data via the nocloud data source.

The autoinstall config should be provided under the autoinstall key in the config. For example:

#cloud-config
autoinstall:
 version: 1
 ...

Running a truly automatic autoinstall

Even if a fully noninteractive autoinstall config is found, the server installer will ask for confirmation before writing to the disks unless autoinstall is present on the kernel command line. This is to make it harder to accidentally create a USB stick that will reformat a machine it is plugged into at boot. Many autoinstalls will be done via netboot, where the kernel command line is controlled by the netboot config – just remember to put autoinstall in there!

Quick start

So you just want to try it out? Well we have the page for you.

Creating an autoinstall config

When any system is installed using the server installer, an autoinstall file for repeating the install is created at /var/log/installer/autoinstall-user-data.

Translating a preseed file

If you have a preseed file already, the autoinstall-generator snap can assist in translating that preseed data to an autoinstall file. See this discussion for more details.

The structure of an autoinstall config

The autoinstall config has full documentation.

Technically speaking the config is not defined as a textual format, but cloud-init config is usually provided as YAML so that is the syntax the documentation uses.

A minimal config is:

```
version: 1
identity:
    hostname: hostname
    username: username
    password: $crypted_pass
Here is an example file that shows off most features:
```

```
version: 1
reporting:
    hook:
        type: webhook
        endpoint: http://example.com/endpoint/path
early-commands:
    - ping -c1 198.162.1.1
locale: en US
keyboard:
    layout: gb
    variant: dvorak
network:
    network:
        version: 2
        ethernets:
            enp0s25:
               dhcp4: yes
            enp3s0: {}
            enp4s0: {}
        bonds:
            bond0:
                dhcp4: yes
                interfaces:
                    - enp3s0
                    - enp4s0
                parameters:
                    mode: active-backup
                    primary: enp3s0
proxy: http://squid.internal:3128/
apt:
    primary:
        - arches: [default]
          uri: http://repo.internal/
    sources:
        my-ppa.list:
            source: "deb http://ppa.launchpad.net/curtin-dev/test-archive/ubuntu $RELEASE main"
            keyid: B59D 5F15 97A5 04B7 E230 6DCA 0620 BBCF 0368 3F77
storage:
```

```
layout:
        name: lvm
identity:
    hostname: hostname
    username: username
    password: $crypted pass
ssh:
    install-server: yes
    authorized-keys:
      - $key
    allow-pw: no
snaps:
    - name: go
      channel: 1.14/stable
      classic: true
debconf-selections: |
    bind9
               bind9/run-resolvconf
                                        boolean false
packages:
    - libreoffice
    - dns-server^
user-data:
    disable_root: false
late-commands:
    - sed -ie 's/GRUB_TIMEOUT=.*/GRUB_TIMEOUT=30/' /target/etc/default/grub
error-commands:
    - tar c /var/log/installer | nc 192.168.0.1 1000
```

Many keys and values correspond straightforwardly to questions the installer asks (e.g. keyboard selection). See the reference for details of those that do not.

Error handling

Progress through the installer is reported via the error-commands are executed and the traceback printed to the console. The server then just waits.

Possible future directions

We might want to extend the 'match specs' for disks to cover other ways of selecting disks.

Autoinstall Quick Start

The intent of this page is to provide simple instructions to perform an autoinstall in a VM on your machine.

This page assumes you are on the amd64 architecture. There is a version for s390x too.

Providing the autoinstall data over the network

This method is the one that generalizes most easily to doing an entirely network-based install, where a machine netboots and then is automatically installed.

Download the ISO

Go to the 20.04 ISO download page and download the latest Ubuntu 20.04 live-server ISO.

Mount the ISO

sudo mount -r ~/Downloads/ubuntu-20.04-live-server-amd64.iso /mnt

Write your autoinstall config

This means creating cloud-init config as follows:

mkdir -p ~/www cd ~/www
```
cat > user-data << 'EOF'
#cloud-config
autoinstall:
    version: 1
    identity:
        hostname: ubuntu-server
        password: "$6$exDY1mhS4KUYCE/2$zmn9ToZwTKLhCw.b4/b.ZRTIZM30JZ4Qr0Q2a0XJ8yk96xpcCof0kxKwuX1kqLG/ygbJ1f8wxED22bTL4F46
        username: ubuntu
EOF
touch meta-data</pre>
```

The crypted password is just "ubuntu".

Serve the cloud-init config over http

Leave this running in one terminal window:

cd ~/www python3 -m http.server 3003

Create a target disk

truncate -s 10G image.img

Run the install!

```
kvm -no-reboot -m 1024 \
    -drive file=image.img,format=raw,cache=none,if=virtio \
    -cdrom ~/Downloads/ubuntu-20.04-live-server-amd64.iso \
    -kernel /mnt/casper/vmlinuz \
    -initrd /mnt/casper/initrd \
    -append 'autoinstall ds=nocloud-net;s=http://_gateway:3003/'
```

This will boot, download the config from the server set up in the previous step and run the install. The installer reboots at the end but the -no-reboot flag to kvm means that kvm will exit when this happens. It should take about 5 minutes.

Boot the installed system

```
kvm -no-reboot -m 1024 \
    -drive file=image.img,format=raw,cache=none,if=virtio
```

This will boot into the freshly installed system and you should be able to log in as ubuntu/ubuntu.

Using another volume to provide the autoinstall config

This is the method to use when you want to create media that you can just plug into a system to have it be installed.

Download the live-server ISO

Go to the 20.04 ISO download page and download the latest Ubuntu 20.04 live-server ISO.

Create your user-data & meta-data files

```
mkdir -p ~/cidata
cd ~/cidata
cat > user-data << 'EOF'
#cloud-config
autoinstall:
   version: 1
   identity:
    hostname: ubuntu-server
   password: "$6$exDY1mhS4KUYCE/2$zmn9ToZwTKLhCw.b4/b.ZRTIZM30JZ4Qr0Q2a0XJ8yk96xpcCof0kxKwuX1kqLG/ygbJ1f8wxED22bTL4F46
    username: ubuntu
EOF
touch meta-data</pre>
```

The crypted password is just "ubuntu".

Create an ISO to use as a cloud-init data source

sudo apt install cloud-image-utils
cloud-localds ~/seed.iso user-data meta-data

Create a target disk

truncate -s 10G image.img

Run the install!

```
kvm -no-reboot -m 1024 \
    -drive file=image.img,format=raw,cache=none,if=virtio \
    -drive file=~/seed.iso,format=raw,cache=none,if=virtio \
    -cdrom ~/Downloads/ubuntu-20.04-live-server-amd64.iso
```

This will boot and run the install. Unless you interrupt boot to add 'autoinstall' to the kernel command line, the installer will prompt for confirmation before touching the disk.

The installer reboots at the end but the -no-reboot flag to kvm means that kvm will exit when this happens.

The whole process should take about 5 minutes.

Boot the installed system

```
kvm -no-reboot -m 1024 \
    -drive file=image.img,format=raw,cache=none,if=virtio
```

This will boot into the freshly installed system and you should be able to log in as ubuntu/ubuntu.

Autoinstall Quick Start for s390x

The intent of this page is to provide simple instructions to perform an autoinstall in a VM on your machine on s390x.

This page is just a slightly adapted page of Automated Server Install Quickstart mapped to s390x.

Download an ISO

At the time of writing (just before focal release), the best place to go is here: http://cdimage.ubuntu.com/ubuntu/releases/20.04/release/

wget http://cdimage.ubuntu.com/ubuntu/releases/20.04/release/ubuntu-20.04-live-server-s390x.iso -P ~/Downloads

Mount the ISO

```
mkdir -p ~/iso
sudo mount -r ~/Downloads/ubuntu-20.04-live-server-s390x.iso ~/iso
```

Write your autoinstall config

This means creating cloud-init config as follows:

```
mkdir -p ~/www
cd ~/www
cat > user-data << 'EOF'
#cloud-config
autoinstall:
    version: 1
    identity:
    hostname: ubuntu-server
    password: "$6$exDY1mhS4KUYCE/2$zmn9ToZwTKLhCw.b4/b.ZRTIZM30JZ4Qr0Q2a0XJ8yk96xpcCof0kxKwuX1kqLG/ygbJ1f8wxED22bTL4F46
    username: ubuntu
EOF
touch meta-data
The crypted password is just "ubuntu".</pre>
```

Serve the cloud-init config over http

Leave this running in one terminal window:

cd ~/www python3 -m http.server 3003

Create a target disk

Proceed with a second terminal window:

```
sudo apt install qemu-utils
```

• • •

```
qemu-img create -f qcow2 disk-image.qcow2 10G
Formatting 'disk-image.qcow2', fmt=qcow2 size=10737418240 cluster_size=65536 lazy_refcounts=off refcount_bits=16
```

```
qemu-img info disk-image.qcow2
image: disk-image.qcow2
file format: qcow2
virtual size: 10 GiB (10737418240 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

Run the install!

```
sudo apt install qemu-kvm
...
You may need to add the default user to the kvm group: <
>
sudo usermod -a -G kvm ubuntu # re-login to make the changes take effect
kvm -no-reboot -name auto-inst-test -nographic -m 2048 \
    -drive file=disk-image.qcow2,format=qcow2,cache=none,if=virtio \
    -cdrom ~/Downloads/ubuntu-20.04-live-server-s390x.iso \
    -kernel ~/iso/boot/kernel.ubuntu \
    -initrd ~/iso/boot/initrd.ubuntu \
    -append 'autoinstall ds=nocloud-net;s=http://_gateway:3003/ console=ttysclp0'
```

This will boot, download the config from the server set up in the previous step and run the install. The installer reboots at the end but the -no-reboot flag to kvm means that kvm will exit when this happens. It should take about 5 minutes.

Boot the installed system

```
kvm -no-reboot -name auto-inst-test -nographic -m 2048 \
    -drive file=disk-image.qcow2,format=qcow2,cache=none,if=virtio
```

This will boot into the freshly installed system and you should be able to log in as ubuntu/ubuntu.

Automated Server Installs Config File Reference

Overall format

The autoinstall file is YAML. At top level it must be a mapping containing the keys described in this document. Unrecognized keys are ignored.

Schema

Autoinstall configs are validated against a JSON schema before they are used.

Command lists

Several config keys are lists of commands to be executed. Each command can be a string (in which case it is executed via "sh -c") or a list, in which case it is executed directly. Any command exiting with a non-zero return code is considered an error and aborts the install (except for error-commands, where it is ignored).

Top-level keys

version

type: integer
default: no default

A future-proofing config file version field. Currently this must be "1".

interactive-sections

type: list of strings default: []

A list of config keys to still show in the UI. So for example:

Would stop on the network screen and allow the user to change the defaults. If a value is provided for an interactive section it is used as the default.

You can use the special section name of "*" to indicate that the installer should ask all the usual questions – in this case, the autoinstall.yaml file is not really an "autoinstall" file at all, instead just a way to change the defaults in the UI.

Not all config keys correspond to screens in the UI. This documentation indicates if a given section can be interactive or not.

If there are any interactive sections at all, the reporting key is ignored.

early-commands

type: command list default: no commands can be interactive: no

A list of shell commands to invoke as soon as the installer starts, in particular before probing for block and network devices. The autoinstall config is available at /autoinstall.yaml (irrespective of how it was provided) and the file will be re-read after the early-commands have run to allow them to alter the config if necessary.

locale

type: string
default: en_US.UTF-8
can be interactive: yes, always interactive if any section is

The locale to configure for the installed system.

refresh-installer

type: mapping
default: see below
can be interactive: yes

Controls whether the installer updates to a new version available in the given channel before continuing.

The mapping contains keys:

update

type: boolean default: no

Whether to update or not.

channel

type: string
default: "stable/ubuntu-\$REL"

The channel to check for updates.

keyboard

type: mapping, see below default: US English keyboard can be interactive: yes

The layout of any attached keyboard. Often systems being automatically installed will not have a keyboard at all in which case the value used here does not matter.

The mapping's keys correspond to settings in the /etc/default/keyboard configuration file. See its manual page for more details.

The mapping contains keys:

layout

type: string
default: "us"

Corresponds to the $\mathsf{XKBLAYOUT}$ setting.

variant

type: string default: ""

Corresponds to the XKBVARIANT setting.

toggle

type: string or null
default: null

Corresponds to the value of grp: option from the XKBOPTIONS setting. Acceptable values are (but note that the installer does not validate these): caps_toggle, toggle, rctrl_toggle, rshift_toggle, rwin_toggle, menu_toggle, alt_shift_toggle, ctrl_shift_toggle, ctrl_alt_toggle, alt_caps_toggle, lctrl_lshift_toggle, lalt_toggle, lctrl_toggle, lshift_toggle, lwin_toggle, sclk_toggle

The version of subiquity released with 20.04 GA does not accept null for this field due to a bug.

network

type: netplan-format mapping, see below **default:** DHCP on interfaces named eth* or en* **can be interactive:** yes

netplan formatted network configuration. This will be applied during installation as well as in the installed system. The default is to interpret the config for the install media, which runs DHCPv4 on any interface with a name matching "eth*" or "en*" but then disables any interface that does not receive an address.

For example, to run dhcp6 on a particular NIC:

```
network:
  version: 2
  ethernets:
    enp0s31f6:
    dhcp6: yes
```

Note that thanks to a bug, the version of subiquity released with 20.04 GA forces you to write this with an extra "network:" key like so:

```
network:
  network:
    version: 2
    ethernets:
        enp0s31f6:
        dhcp6: yes
```

Later versions support this syntax too for compatibility but if you can assume a newer version you should use the former.

proxy

type: URL or null
default: no proxy
can be interactive: yes

The proxy to configure both during installation and for apt and for snapd in the target system.

 \mathbf{apt}

type: mapping
default: see below
can be interactive: yes

Apt configuration, used both during the install and once booted into the target system.

This uses the same format as curtin which is documented at https://curtin.readthedocs.io/en/latest/topics/apt_source. html, with one extension: the geoip key controls whether a geoip lookup is done.

The default is:

```
apt:
```

```
preserve_sources_list: false
primary:
    arches: [i386, amd64]
    uri: "http://archive.ubuntu.com/ubuntu"
    arches: [default]
    uri: "http://ports.ubuntu.com/ubuntu-ports"
geoip: true
```

If geoip is true and the mirror to be used is the default, a request is made to https://geoip.ubuntu.com/lookup and the mirror uri to be used changed to be http://CC.archive.ubuntu.com/ubuntu where CC is the country code returned by the lookup (or similar for ports). If this section is not interactive, the request is timed out after 10 seconds.

Any supplied config is merged with the default rather than replacing it.

If you just want to set a mirror, use a config like this:

```
apt:
    primary:
        - arches: [default]
        uri: YOUR_MIRROR_GOES_HERE
```

To add a ppa:

apt:

```
sources:
    curtin-ppa:
        source: ppa:curtin-dev/test-archive
```

storage

type: mapping, see below
default: use "lvm" layout in a single disk system, no default in a multiple disk system
can be interactive: yes

Storage configuration is a complex topic and the description of the desired configuration in the autoinstall file can necessarily also be complex. The installer supports "layouts", simple ways of expressing common configurations.

Supported layouts

The two supported layouts at the time of writing are "lvm" and "direct".

```
storage:
    layout:
    name: lvm
storage:
    layout:
    name: direct
```

By default these will install to the largest disk in a system, but you can supply a match spec (see below) to indicate which disk to use:

```
storage:
  layout:
    name: lvm
    match:
        serial: CT*
storage:
  layout:
    name: disk
    match:
        ssd: yes
```

(you can just say "match: {}" to match an arbitrary disk)

The default is to use the lvm layout.

action-based config

For full flexibility, the installer allows storage configuration to be done using a syntax which is a superset of that supported by curtin, described at https://curtin.readthedocs.io/en/latest/topics/storage.html.

If the "layout" feature is used to configure the disks, the "config" section will not be used.

As well as putting the list of actions under the 'config' key, the grub and swap curtin config items can be put here. So a storage section might look like:

```
storage:
    swap:
        size: 0
    config:
        - type: disk
        id: disk0
        serial: ADATA_SX8200PNP_XXXXXXXXXX
        - type: partition
        ...
```

The extensions to the curtin syntax are around disk selection and partition/logical volume sizing.

Disk selection extensions

Curtin supported identifying disks by serial (e.g. Crucial_CT512MX100SSD1_14250C57FECE) or by path (e.g. /dev/sdc) and the server installer supports this as well. The installer additionally supports a "match spec" on a disk action that supports more flexible matching.

The actions in the storage config are processed in the order they are in the autoinstall file. Any disk action is assigned a matching disk – chosen arbitrarily from the set of unassigned disks if there is more than one, and causing the installation to fail if there is no unassigned matching disk.

A match spec supports the following keys:

- model: foo: matches a disk where ID_VENDOR=foo in udev, supporting globbing
- path: foo: matches a disk where DEVPATH=foo in udev, supporting globbing (the globbing support distinguishes this from specifying path: foo directly in the disk action)
- serial: foo: matches a disk where ID_SERIAL=foo in udev, supporting globbing (the globbing support distinguishes this from specifying serial: foo directly in the disk action)
- ssd: yes | no: matches a disk that is or is not an SSD (vs a rotating drive)

• size: largest|smallest: take the largest or smallest disk rather than an arbitrary one if there are multiple matches (support for smallest added in version 20.06.1)

So for example, to match an arbitrary disk it is simply:

type: disk
 id: disk0

To match the largest ssd:

```
    type: disk
        id: big-fast-disk
        match:
            ssd: yes
            size: largest
```

To match a Seagate drive:

```
- type: disk
id: data-disk
match:
    model: Seagate
```

partition/logical volume extensions

The size of a partition or logical volume in curtin is specified as a number of bytes. The autoinstall config is more flexible:

- You can specify the size using the "1G", "512M" syntax supported in the installer UI
- You can specify the size as a percentage of the containing disk (or RAID), e.g. "50%"
- For the last partition specified for a particular device, you can specify the size as "-1" to indicate that the partition should fill the remaining space.
- type: partition id: boot-partition device: root-disk
- size: 10%
 type: partition
 id: root-partition
 size: 20G
- type: partition id: data-partition device: root-disk size: -1

identity

type: mapping, see below default: no default can be interactive: yes

Configure the initial user for the system. This is the only config key that must be present (unless the user-data section is present, in which case it is optional).

A mapping that can contain keys, all of which take string values:

realname

The real name for the user. This field is optional.

username

The user name to create.

hostname

The hostname for the system.

password

The password for the new user, crypted. This is required for use with sudo, even if SSH access is configured.

 \mathbf{ssh}

type: mapping, see below
default: see below
can be interactive: yes

Configure ssh for the installed system. A mapping that can contain keys:

install-server

type: boolean
default: false

Whether to install OpenSSH server in the target system.

authorized-keys

type: list of strings
default: []

A list of SSH public keys to install in the initial user's account.

allow-pw

type: boolean
default: true if authorized_keys is empty, false otherwise

snaps

type: list
default: install no extra snaps
can be interactive: yes

A list of snaps to install. Each snap is represented as a mapping with required name and optional channel (defaulting to stable) and classic (defaulting to false) keys. For example:

snaps:

 name: etcd channel: edge classic: false

debconf-selections

type: string
default: no config
can be interactive: no

The installer will update the target with debconf set-selection values. Users will need to be familiar with the package debconf options.

packages

type: list
default: no packages
can be interactive: no

A list of packages to install into the target system. More precisely, a list of strings to pass to "apt-get install", so this includes things like task selection (dns-server[^]) and installing particular versions of a package (my-package=1-1).

late-commands

type: command list default: no commands can be interactive: no Shell commands to run after the install has completed successfully and any updates and packages installed, just before the system reboots. They are run in the installer environment with the installed system mounted at /target. You can run curtin in-target -- \$shell_command (with the version of subiquity released with 20.04 GA you need to specify this as curtin in-target --target=/target -- \$shell_command) to run in the target system (similar to how plain in-target can be used in d-i preseed/late_command).

error-commands

type: command list default: no commands can be interactive: no

Shell commands to run after the install has failed. They are run in the installer environment, and the target system (or as much of it as the installer managed to configure) will be mounted at /target. Logs will be available at /var/log/installer in the live session.

reporting

type: mapping
default: type: print which causes output on tty1 and any configured serial consoles
can be interactive: no

The installer supports reporting progress to a variety of destinations. Note that this section is ignored if there are any interactive sections; it only applies to fully automated installs.

The config, and indeed the implementation, is 90% the same as that used by curtin.

Each key in the reporting mapping in the config defines a destination, where the type sub-key is one of:

The rsyslog reporter does not yet exist

- print: print progress information on tty1 and any configured serial console. There is no other configuration.
- rsyslog: report progress via rsyslog. The destination key specifies where to send output.
- webhook: report progress via POSTing JSON reports to a URL. Accepts the same configuration as curtin.
- none: do not report progress. Only useful to inhibit the default output.

Examples:

The default configuration is:

reporting: builtin: type: print

Report to rsyslog:

reporting: central: type: rsyslog destination: @192.168.0.1

Suppress the default output:

reporting: builtin: type: none

Report to a curtin-style webhook:

```
reporting:
hook:
type: webhook
endpoint: http://example.com/endpoint/path
consumer_key: "ck_foo"
consumer_secret: "cs_foo"
token_key: "tk_foo"
token_secret: "tk_secret"
level: INF0
```

user-data

type: mapping
default: {}
can be interactive: no

Provide cloud-init user-data which will be merged with the user-data the installer produces. If you supply this, you don't need to supply an identity section (but then it's your responsibility to make sure that you can log into the installed system!).

JSON Schema for autoinstall config

Introduction

The server installer validates the provided autoinstall config against a JSON Schema.

How the config is validated

Although the schema is presented below as a single document, and if you want to pre-validate your config you should validate it against this document, the config is not actually validated against this document at run time. What happens instead is that some sections are loaded, validated and applied first, before all other sections are validated. In detail:

- 1. The reporting section is loaded, validated and applied.
- 2. The error commands are loaded and validated.
- 3. The early commands are loaded and validated.
- 4. The early commands, if any, are run.
- 5. The config is reloaded, and now all sections are loaded and validated.

This is so that validation errors in most sections can be reported via the reporting and error-commands configuration, as all other errors are.

Schema

{

The JSON schema for autoinstall data is as follows:

```
"type": "object",
"properties": {
    "version": {
        "type": "integer",
        "minumum": 1,
        "maximum": 1
    },
    "early-commands": {
        "type": "array",
        "items": {
             "type": [
                 "string",
                 "array"
            ],
            "items": {
                 "type": "string"
            }
        }
    },
    "reporting": {
        "type": "object",
        "additionalProperties": {
            "type": "object",
            "properties": {
                 "type": {
                     "type": "string"
                 }
            },
```

"required": [

```
"type"
        ],
        "additionalProperties": true
    }
},
"error-commands": {
    "type": "array",
    "items": {
        "type": [
            "string",
            "array"
        ],
        "items": {
            "type": "string"
        }
    }
},
"user-data": {
    "type": "object"
},
"packages": {
    "type": "array",
    "items": {
       "type": "string"
    }
},
"debconf-selections": {
    "type": "string"
},
"locale": {
    "type": "string"
},
"refresh-installer": {
    "type": "object",
    "properties": {
        "update": {
            "type": "boolean"
        },
        "channel": {
            "type": "string"
        }
    },
    "additionalProperties": false
},
"keyboard": {
    "type": "object",
    "properties": {
        "layout": {
            "type": "string"
        },
        "variant": {
           "type": "string"
        },
        "toggle": {
            "type": [
                "string",
                "null"
            ]
        }
    },
    "required": [
        "layout"
    ],
```

```
"additionalProperties": false
},
"network": {
    "oneOf": [
        {
            "type": "object",
            "properties": {
                "version": {
                    "type": "integer",
                    "minimum": 2,
                    "maximum": 2
                },
                "ethernets": {
                    "type": "object",
                    "properties": {
                        "match": {
                            "type": "object",
                            "properties": {
                                 "name": {
                                    "type": "string"
                                },
                                 "macaddress": {
                                    "type": "string"
                                },
                                "driver": {
                                    "type": "string"
                                }
                            },
                            "additionalProperties": false
                        }
                    }
                },
                "wifis": {
                    "type": "object",
                    "properties": {
                        "match": {
                            "type": "object",
                            "properties": {
                                "name": {
                                    "type": "string"
                                },
                                 "macaddress": {
                                    "type": "string"
                                },
                                "driver": {
                                    "type": "string"
                                }
                            },
                            "additionalProperties": false
                        }
                    }
                },
                "bridges": {
                    "type": "object"
                },
                "bonds": {
                    "type": "object"
                },
                "tunnels": {
                    "type": "object"
                },
                "vlans": {
                    "type": "object"
```

```
}
    },
    "required": [
        "version"
    ]
},
{
    "type": "object",
    "properties": {
        "network": {
            "type": "object",
            "properties": {
                "version": {
                    "type": "integer",
                    "minimum": 2,
                    "maximum": 2
                },
                "ethernets": {
                    "type": "object",
                    "properties": {
                        "match": {
                            "type": "object",
                            "properties": {
                                "name": {
                                    "type": "string"
                                },
                                 "macaddress": {
                                    "type": "string"
                                },
                                "driver": {
                                     "type": "string"
                                }
                            },
                            "additionalProperties": false
                        }
                    }
                },
                "wifis": {
                    "type": "object",
                    "properties": {
                        "match": {
                            "type": "object",
                            "properties": {
                                "name": {
                                    "type": "string"
                                },
                                 "macaddress": {
                                    "type": "string"
                                },
                                 "driver": {
                                     "type": "string"
                                }
                            },
                            "additionalProperties": false
                        }
                    }
                },
                "bridges": {
                    "type": "object"
                },
                "bonds": {
                    "type": "object"
                },
```

```
"tunnels": {
                            "type": "object"
                        },
                        "vlans": {
                            "type": "object"
                        }
                    },
                    "required": [
                        "version"
                    ]
                }
            },
            "required": [
                "network"
            ]
        }
    ]
},
"proxy": {
    "type": [
        "string",
        "null"
    ],
    "format": "uri"
},
"apt": {
    "type": "object",
    "properties": {
        "preserve_sources_list": {
            "type": "boolean"
        },
        "primary": {
           "type": "array"
        },
        "geoip": {
           "type": "boolean"
        },
        "sources": {
            "type": "object"
        }
    }
},
"storage": {
    "type": "object"
},
"identity": {
    "type": "object",
    "properties": {
        "realname": {
            "type": "string"
        },
        "username": {
            "type": "string"
        },
        "hostname": {
           "type": "string"
        },
        "password": {
            "type": "string"
        }
    },
    "required": [
        "username",
```

```
"hostname",
            "password"
       ],
        "additionalProperties": false
   },
    "ssh": {
        "type": "object",
        "properties": {
            "install-server": {
                "type": "boolean"
            },
            "authorized-keys": {
                "type": "array",
                "items": {
                    "type": "string"
                }
            },
            "allow-pw": {
                "type": "boolean"
            }
       }
   },
    "snaps": {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "name": {
                    "type": "string"
                },
                "channel": {
                    "type": "string"
                },
                "classic": {
                    "type": "boolean"
                }
            },
            "required": [
               "name"
            ],
            "additionalProperties": false
       }
   },
    "late-commands": {
        "type": "array",
        "items": {
            "type": [
                "string",
                "array"
            ],
            "items": {
                "type": "string"
            }
       }
   }
},
"required": [
   "version"
],
"additionalProperties": true
```

}

Regeneration

The schema above can be regenerated by running "make schema" in a subiquity source checkout.

Non-interactive IBM z/VM (s390x) installation using autoinstall

This non-interactive installation uses 'autoinstall', which can be considered as successor of d-i / preseed on Ubuntu. This is a detailed step-by-step guide, incl. output and logs, that are partially a bit shortened - indicated by '...' - to limit the size of this document.

The example z/VM guest here uses DASD storage and is connected to regular (non VLAN) network. For a zFCP and a VLAN network example, please see the Non-interactive IBM LPAR (s390x) installation using autoinstall.

• Start with the preparation of the (ftp) install server (if it not already exists).

```
user@local:~$ ssh admin@installserver.local
  admin@installserver:~$ mkdir -p /srv/ftp/ubuntu-daily-live-server-20.04
  admin@installserver:~$ wget http://cdimage.ubuntu.com/ubuntu-server/focal/daily-live/current/focal-
  live-server-s390x.iso --directory-prefix=/srv/ftp/ubuntu-daily-live-server-20.04
  --2020-06-26 12:18:48-- http://cdimage.ubuntu.com/ubuntu-server/focal/daily-live/current/focal-live-
  server-s390x.iso
  Resolving cdimage.ubuntu.com (cdimage.ubuntu.com)... 2001:67c:1560:8001::1d, 2001:67c:1360:8001::28, 2001:67c:1360
  Connecting to cdimage.ubuntu.com (cdimage.ubuntu.com)|2001:67c:1560:8001::1d|:80... connected.
  HTTP request sent, awaiting response... 200 OK
  Length: 700952576 (668M) [application/x-iso9660-image]
  Saving to: 'focal-live-server-s390x.iso'
  focal-live-server-s 100%[=======>] 668.48M 2.73MB/s
                                                                      in 4m 54s
  2020-06-26 12:23:42 (2.27 MB/s) - 'focal-live-server-s390x.iso' saved [700952576/700952576]
  admin@installserver:~$
 The ISO image needs to be extracted now. Since files in it's boot folder need to be modified, loopback mount is
  not an option here:
  admin@installserver:~$ cd /srv/ftp/ubuntu-daily-live-server-20.04
  admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ mkdir iso
  admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ sudo mount -o loop ./focal-live-server-
  s390x.iso ./iso
  [sudo] password for admin:
  mount: /home/user/iso-test/iso: WARNING: device write-protected, mounted read-only.
  admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ ls -l
  total 684530
  -rw-rw-r-- 1 user user 700952576 Jun 26 10:12 focal-live-server-s390x.iso
  dr-xr-xr-x 10 root
                                     2048 Jun 26 10:12 iso
                        root
  admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$
• Now make sure an ftp server is running in the installserver with /srv/ftp as ftp-server root (as used in this
  example).
• Now prepare an autoinstall (http) server, that hosts the configuration data for the non-interactive installation.
```

```
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ mkdir -p /srv/www/autoinstall/zvmguest
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ cd /srv/www/autoinstall/zvmguest
admin@installserver:/srv/www/autoinstall/zvmguest$
admin@installserver:/srv/www/autoinstall/zvmguest$ echo "instance-id: $(uuidgen || openssl rand -
base64 8)" > meta-data
admin@installserver:/srv/www/autoinstall/zvmguest$ cat meta-data
instance-id: 2c2215fb-6a38-417f-b72f-376blcc44f01
admin@installserver:/srv/www/autoinstall/zvmguest$
admin@installserver:/srv/www/autoinstall/zvmguest$
admin@installserver:/srv/www/autoinstall/zvmguest$ vi user-data
admin@installserver:/srv/www/autoinstall/zvmguest$ cat user-data
inginstallserver:/srv/www/autoinstall/zvmguest$ cat user-data
admin@installserver:/srv/www/autoinstall/zvmguest$ cat user-data
inginstallserver:/srv/www/autoinstall/zvmguest$ cat user-data
admin@installserver:/srv/www/autoinstall/zvmguest$ cat user-data
inginstallserver:/srv/www/autoinstall/zvmguest$ cat user-data
```

```
refresh-installer:
   update: yes
 reporting:
   builtin:
      type: print
 apt:
   preserve_sources_list: false
   primary:
    - arches: [amd64, i386]
     uri: http://archive.ubuntu.com/ubuntu
    - arches: [default]
     uri: http://ports.ubuntu.com/ubuntu-ports
 keyboard:
   layout: en
   variant: us
 locale: en_US
 identity:
   hostname: zvmguest
   password:
"$6$ebJ1f8wxED22bTL4F46P0"
     username: ubuntu
   user-data:
     timezone: America/Boston
      users:
        - name: ubuntu
          password:
"$6$KwuxED22bTL4F46P0"
          lock_passwd: false
   early-commands:
      - touch /tmp/lets_activate_the_s390x_devices
      - chzdev dasd -e 1f00
      - touch /tmp/s390x_devices_activation_done
   network:
      version: 2
     ethernets:
        enc600:
          addresses: [10.11.12.23/24]
          gateway4: 10.11.12.1
          nameservers:
            addresses: [10.11.12.1]
   ssh:
      install-server: true
     allow-pw: true
      authorized-keys: ['ssh-rsa meQwtZ user@workstation # ssh-import-id lp:user']
 admin@installserver:~$
```

• For s390x installations the *early-commands* section is the interesting part:

early-commands:

- touch /tmp/lets_activate_the_s390x_devices
- chzdev dasd -e 1f00
- touch /tmp/s390x_devices_activation_done

The first and last *early-commands* are of course optional and only frame and indicate the real s390x command activation.

In this particular example a single DASD ECKD disk with the address 1f00 is enabled. zFCP disk storage can be enabled via their host (host-bus-adapters) addresses, for example e000 (*chzdev zfcp -e* e000) and e100 (*chzdev zfcp -e* e000). These have certain LUNs assigned that are all automatically discovered and activated by *chzdev zfcp-lun -e --online*.

An activation of a QETH device would looks like this: chzdev qeth -e 0600 - as usual.

- For more details about the autoinstall config options, please have a look at the autoinstall reference and autoinstall schema page.
- Now make sure a http server is running with /srv/www as web-server root (in this particular example).

- Login in to your z/VM system using your preferred 3270 client, for example x3270 or c3270.
- Now transfer installer kernel, initrd, parmfile and exec file to the z/VM system that is used for the installation. Put these files for example on File Mode (Fm) A (aka disk A):

```
listfiles
UBUNTU EXEC A1
KERNEL UBUNTU A1
INITRD UBUNTU A1
PARMFILE UBUNTU A1
```

• Now specify the needed autoinstall parameters in the parmfile:

```
xedit PARMFILE UBUNTU A
PARMFILE UBUNTU 01 F 80 Trunc=80 Size=3 Line=0 Col=1 Alt=0
00000 * * * Top of File * * *
00001 ip=10.11.12.23::10.11.12.1:255.255.0:zvmguest:enc600:none:10.11.12.1
00002 url=ftp://installserver.local:21/ubuntu-daily-live-server-20.04/focal-li
ve-ser
00003 ver-s390x.iso autoinstall ds=nocloud-net;s=http://installserver.local:80
00004 /autoinstall/zvmguest/ --- quiet
00005 * * * End of File * *
```

- *Note:* In case of any issues hitting the 80 characters barrier of the file, you may just keep writing parameters across two lines, just make sure there are no unwanted whitespaces. To see all 80 characters in one line, disable the prefix area on the left. "prefix off | on" will be your friend use it in the command area.
- Now start the z/VM installation by executing the UBUNTU REXX script with UBUNTU.
- Now monitor the IPL (the bootup process) of the install system. This is quite crucial, because during this process a temporary installation password is generated and displayed. The line(s) look similar to this:

```
...
|37.487141| cloud-init-1873|: Set the following 'random' passwords
|37.487176| cloud-init-1873|: installer: **i7UFdP8fhiVVMme3qqH8**
...
```

That is needed for remotely connecting to the installer via ssh in the next step.

• So start the REXX script:

UBUNTU 00: 0000004 FILES PURGED 00: RDR FILE 1254 SENT FROM zvmguest PUN WAS 1254 RECS 102K CPY 001 A NOHOLD NO KEEP 00: RDR FILE 1258 SENT FROM zvmguest PUN WAS 1258 RECS 0003 CPY 001 A NOHOLD NO **KEEP** 00: RDR FILE 1262 SENT FROM zvmguest PUN WAS 1262 RECS 303K CPY 001 A NOHOLD NO KEEP 00: 0000003 FILES CHANGED 00: 0000003 FILES CHANGED 01: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial C PU reset from CPU 00. 02: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial C PU reset from CPU 00. 03: HCPGSP2627I The virtual machine is placed in CP mode due to a SIGP initial C PU reset from CPU 00. 0.403380| Initramfs unpacking failed: Decoding failed ln: /tmp/mountroot-fail-hooks.d//scripts/init-premount/lvm2: No such file or dir ectorv QETH device 0.0.0600:0.0.0601:0.0.0602 configured

```
IP-Config: enc600 hardware address 02:28:0b:00:00:51 mtu 1500
IP-Config: enc600 guessed broadcast address 10.11.12.255
IP-Config: enc600 complete:
address: 10.11.12.23 broadcast: 10.210.210.255 netmask: 255.255.255.0
gateway: 10.11.12.1 dns0 : 10.11.12.1 dns1 : 0.0.0.0
host : zvmguest
rootserver: 0.0.0.0 rootpath:
 filename :
Connecting to installserver.local:21 (10.11.12.2:21)
                                                     ! 15.2M 0:00:48 ETA
focal-live-server-s3 2% !
focal-live-server-s3 16% !*****
                                                     ! 126M 0:00:09 ETA
focal-live-server-s3 31% !********
                                                    ! 236M 0:00:06 ETA
focal-live-server-s3 46% !*************
                                                    ! 347M 0:00:04 ETA
focal-live-server-s3 60% !********************
                                                    ! 456M 0:00:03 ETA
! 563M 0:00:02 ETA
mount: mounting /cow on /root/cow failed: No such file or directory
Connecting to plymouth: Connection refused
passwd: password expiry information changed.
   16.748137| /dev/loop3: Can't open blockdev
   17.908156| systemd¬1|: Failed unmounting /cdrom.
¬ ¬0;1;31mFAILED ¬0m| Failed unmounting ¬0;1;39m/cdrom ¬0m.
¬ ¬0;32m OK ¬Om| Listening on ¬0;1;39mJournal Socket ¬Om.
        Mounting -0;1;39mHuge Pages File System -0m...
        Mounting ¬0;1;39mPOSIX Message Queue File System ¬0m...
        Mounting -0;1;39mKernel Debug File System -0m...
        Starting ¬0;1;39mJournal Service ¬0m...
[ 61.190916] cloud-init[2076]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 running
'modules:final' at Fri, 26 Jun 2020 11:02:01 +0000. Up 61.09 seconds.
[ 61.191002] cloud-init[2076]: ci-info: no authorized SSH keys fingerprints fo
und for user installer.
[ 61.191071] cloud-init[2076]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 finished at Fri, 26 Jun 2020 11:02:01 +0000
Datasource DataSourceNoCloudNet [seed=cmdline,
/var/lib/cloud/seed/nocloud,
http://installserver.local:80/autoinstall/zvmguest/]
[dsmode=net]. Up 61.18 seconds
[ 61.191136] cloud-init[2076]: Welcome to Ubuntu Server Installer!
   61.191202] cloud-init[2076]: Above you will find SSH host keys and a random
[
password set for the `installer` user. You can use these credentials to ssh-in
and complete the installation. If you provided SSH keys in the cloud-init datasource,
they were also provisioned to the installer user.
[ 61.191273] cloud-init[2076]: If you have access to the graphical console,
like TTY1 or HMC ASCII terminal you can complete the installation there too.
It is possible to connect to the installer over the network, which
might allow the use of a more capable terminal.
To connect, SSH to installer@10.11.12.23.
The password you should use is ''i7UFdP8fhiVVMme3qqH8''.
The host key fingerprints are:
RSA
       SHA256:rBXLeUke3D4gKdsruKEajHjocxc9hr3PI
ECDSA
       SHA256:KZZYFswtKxFXWQPuQS9QpOBUoS6RHswis
ED25519 SHA256:s+5tZfagx0zffC6gYRGW3t1KcBH6f+Vt0
Ubuntu 20.04 LTS ubuntu-server sclp_line0
```

• On short notice, you can even login to the system with the user 'installer' and the temporary password that was given at the end of the boot-up process (see above) of the installation system:

```
user@workstation:~$ ssh installer@zvmguest
  The authenticity of host 'zvmguest (10.11.12.23)' can't be established.
  ECDSA key fingerprint is SHA256:0/dU/D8jJAEGQcbqKGE9La24IRxUPLpzzs5li9F6Vvk.
  Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
  Warning: Permanently added 'zvmguest, 10.11.12.23' (ECDSA) to the list of known hosts.
  installer@zvmguest's password:
  Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-37-generic s390x)
   * Documentation: https://help.ubuntu.com
   * Management: https://landscape.canonical.com
   * Support:
                     https://ubuntu.com/advantage
    System information as of Fri Jun 26 11:08:18 UTC 2020
    System load:
                    1.25
                              Memory usage: 4%
                                                 Processes:
                                                                   192
    Usage of /home: unknown
                              Swap usage: 0%
                                                 Users logged in: 0
  0 updates can be installed immediately.
  0 of these updates are security updates.
  The list of available updates is more than a week old.
  To check for new updates run: sudo apt update
  The programs included with the Ubuntu system are free software;
  the exact distribution terms for each program are described in the
  individual files in /usr/share/doc/*/copyright.
  Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
  applicable law.
  the installer running on /dev/tty1 will perform the autoinstall
  press enter to start a shell
• Please notice that it informs about a running autoinstall process:
  the installer running on /dev/tty1 will perform the autoinstall
• Nevertheless, we can quickly check some things - of course only until the autoinstall process got finished and the
  post-install reboot got triggered:
  root@ubuntu-server:/# ls -l /tmp/lets activate the s390x devices
  -rw-r--r- 1 root root 0 Jun 26 11:08 /tmp/lets_activate_the_s390x_devices
  -rw-r--r-- 1 root root 0 Jun 26 11:09 /tmp/s390x_devices_activation_done
  root@ubuntu-server:/# lszdev | grep yes
  dasd-eckd 0.0.1f00
                                                                yes yes
               0.0.0600:0.0.0601:0.0.0602
  geth
                                                                yes no
                                                                           enc600
  root@ubuntu-server:/#
• If you wait long enough you'll recognize that the remote session get's closed:
  root@ubuntu-server:/# Connection to zvmguest closed by remote host.
  Connection to zvmguest closed.
```

- user@workstation:~\$
- As well as at the console:

```
ubuntu-server login:
```

```
[[0;1;31mFAILED[0m] Failed unmounting [0;1;39m/cdrom[0m.
[ 169.161139] sd-umoun[15600]: Failed to unmount /oldroot: Device or resource busy
```

```
[ 169.161550] sd-umoun[15601]: Failed to unmount /oldroot/cdrom: Device or resource busy
[ 169.168118] shutdown[1]: Failed to finalize file systems, loop devices, ignoring
Total: 282 Selected: 0
```

Command:

```
• ...and that the post-install reboot got triggered:
```

```
Message
Mounting [0;1;39mKernel Configuration File System[0m...
Starting [0;1;39mApply Kernel Variables[0m...
[[0;32m OK [Om] Finished [0;1;39mRemount Root and Kernel File Systems[Om.
[[0;32m OK [Om] Finished [0;1;39mUncomplicated firewall[Om.
[[0;32m OK [Om] Mounted [0;1;39mFUSE Control File System[Om.
[[0;32m OK [Om] Mounted [0;1;39mKernel Configuration File System[Om.
[ 35.378928] cloud-init[2565]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 runnin
g 'modules:final' at Fri, 26 Jun 2020 11:10:44 +0000. Up 35.29 seconds.
  35.378978] cloud-init[2565]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 finish
[
ed at Fri, 26 Jun 2020 11:10:44 +0000. Datasource DataSourceNone. Up 35.37 seconds
   35.379008] cloud-init[2565]: 2020-06-26 11:10:44,359 - cc_final_message.py[W
ſ
ARNING]: Used fallback datasource
[[0;32m OK [Om] Finished [0;1;39mExecute cloud user/final scripts[Om.
[[0;32m OK [Om] Reached target [0;1;39mCloud-init target[Om.
```

zvmguest login:

• With the completion of the reboot the autoinstall is finished and the z/VM guest is ready to use:

```
user@workstation:~$ ssh-keygen -f "/home/user/.ssh/known_hosts" -R "zvmguest"
# Host zvmguest found: line 163
/home/user/.ssh/known_hosts updated.
Original contents retained as /home/user/.ssh/known_hosts.old
user@workstation:~$ ssh ubuntu@zvmguest
The authenticity of host 'zvmguest (10.11.12.23)' can't be established.
ECDSA key fingerprint is SHA256:iGtCArEg+ZnoZlgtOkvmyy0gPY8UEI+f7zoISOF+m/0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'zvmguest,10.11.12.23' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-39-generic s390x)
```

```
* Documentation: https://help.ubuntu.com
```

- * Management: https://landscape.canonical.com
- * Support: https://ubuntu.com/advantage

System information as of Fri 26 Jun 2020 11:12:23 AM UTC

System load: 0.21Memory usage: 3%Processes:189Usage of /: 28.2% of 30.88GBSwap usage: 0%Users logged in: 0

10 updates can be installed immediately.0 of these updates are security updates.To see these additional updates run: apt list --upgradable

The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details.

```
ubuntu@zvmguest:~$ uptime
11:12:35 up 2 min, 1 user, load average: 0.18, 0.17, 0.08
```

```
ubuntu@zvmguest:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:
                Ubuntu 20.04 LTS
Release:
            20.04
Codename:
            focal
ubuntu@zvmquest:~$ uname -a
Linux zvmguest 5.4.0-39-generic #43-Ubuntu SMP Fri Jun 19 10:27:17
UTC 2020 s390x s390x s390x
GNU/Linux
ubuntu@zvmguest:~$ lszdev | grep yes
dasd-eckd
            0.0.1f00
                                                              yes yes
             0.0.0600:0.0.0601:0.0.0602
                                                                         enc600
geth
                                                              yes yes
ubuntu@zvmguest:~$ exit
logout
Connection to zvmguest closed.
user@workstation:~$
```

- Some closing notes:
 - It's always best to use the latest installer and autoinstall components, hence either make sure the installer get's updated to the latest level, or just use a current daily live-server image
 - The ISO image that is specified with the kernel parameters need to fit to the boot folder and it's kernel and initrd that is specified in the Load from Removable Media and Server task at the HMC
 - In addition to activating disk storage resources in 'early-commands', other devices like OSA/qeth can be added and activated there, too - but this is not needed for the basic network device, specified in the kernel parameters that is used for the installation (that one is automatically handled)
 - If everything is properly setup (ftp server for the image, http server for the autoinstall config files) and installation can be as quick as 2 to 3 minutes of course dependent on the complexity of the autoinstall yaml file
 - There is a simple way of generating a sample autoinstall yaml file, one can just perform an interactive subiquity installation and grab the file '/var/log/installer/autoinstall-user-data' and use it as an example but notice that the early-command entries to activate the s390x-specific devices need to be added by hand!

Non-interactive IBM Z LPAR (s390x) installation using autoinstall

This non-interactive installation uses 'autoinstall', which can be considered as successor of d-i / preseed on Ubuntu. This is a detailed step-by-step guide, incl. output and logs, that are partially a bit shortened - indicated by '...' - to limit the size of this document.

The example LPAR here uses zFCP storage and is connected to a VLAN network.

For a DASD and a non-VLAN network example, please see the Non-interactive IBM z/VM (s390x) installation using autoinstall.

• Start with the preparation of the (ftp) install server (if it not already exists).

```
user@local:~$ ssh admin@installserver.local
admin@installserver:~$ mkdir -p /srv/ftp/ubuntu-daily-live-server-20.04
admin@installserver:~$ wget http://cdimage.ubuntu.com/ubuntu-server/focal/daily-live/current/focal-
live-server-s390x.iso --directory-prefix=/srv/ftp/ubuntu-daily-live-server-20.04
--2020-06-26 12:18:48-- http://cdimage.ubuntu.com/ubuntu-server/focal/daily-live/current/focal-live-
server-s390x.iso
Resolving cdimage.ubuntu.com (cdimage.ubuntu.com)... 2001:67c:1560:8001::1d, 2001:67c:1360:8001::28, 2001:67c:1360
 . . .
Connecting to cdimage.ubuntu.com (cdimage.ubuntu.com)|2001:67c:1560:8001::1d|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 700952576 (668M) [application/x-iso9660-image]
Saving to: 'focal-live-server-s390x.iso'
in 4m 54s
2020-06-26 12:23:42 (2.27 MB/s) - 'focal-live-server-s390x.iso' saved [700952576/700952576]
admin@installserver:~$
```

• The ISO image needs to be extracted now. Since files in it's boot folder need to be modified, loopback mount is not an option here:

```
admin@installserver:~$ cd /srv/ftp/ubuntu-daily-live-server-20.04
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ mkdir iso
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ sudo mount -o loop ./focal-live-server-
s390x.iso ./iso
[sudo] password for admin:
mount: /home/user/iso-test/iso: WARNING: device write-protected, mounted read-only.
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ ls -l
total 684530
-rw-rw-r-- 1 user user 700952576 Jun 26 10:12 focal-live-server-s390x.iso
dr-xr-xr-x 10 root
                     root
                                  2048 Jun 26 10:12 iso
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ rsync -rtvz ./iso/ . && sync
sending incremental file list
skipping non-regular file "ubuntu"
skipping non-regular file "ubuntu-ports"
./
README.diskdefines
boot.catalog
md5sum.txt
ubuntu.ins
skipping non-regular file "dists/stable"
skipping non-regular file "dists/unstable"
.disk/
.disk/base_installable
.disk/casper-uuid-generic
.disk/cd type
.disk/info
hoot/
boot/README.boot
boot/initrd.off
boot/initrd.siz
boot/initrd.ubuntu
boot/kernel.ubuntu
boot/parmfile.ubuntu
boot/ubuntu.exec
boot/ubuntu.ikr
boot/ubuntu.ins
casper/
  . . .
sent 681,509,758 bytes received 1,857 bytes 22,344,643.11 bytes/sec
total size is 700,317,941 speedup is 1.03
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ ls -l
total 684578
dr-xr-xr-x 2 user user
                          4096 Jun 26 10:12 boot
-r--r--r-- 1 user user
                            2048 Jun 26 10:12 boot.catalog
dr-xr-xr-x 3 user user
                            4096 Jun 26 10:12 casper
dr-xr-xr-x 3 user user
                            4096 Jun 26 10:11 dists
-rw-rw-r-- 1 user user 700952576 Jun 26 10:12 focal-live-server-s390x.iso
dr-xr-xr-x 2 user user 4096 Jun 26 10:11 install
                                  2048 Jun 26 10:12 iso
dr-xr-xr-x 10 root root
                            4944 Jun 26 10:12 md5sum.txt
-r--r--r-- 1 user user
dr-xr-xr-x 2 user user
                            4096 Jun 26 10:11 pics
dr-xr-xr-x 3 user user
                            4096 Jun 26 10:11 pool
dr-xr-xr-x 2 user user
                            4096 Jun 26 10:11 preseed
-r--r-- 1 user user
                             236 Jun 26 10:11 README.diskdefines
-r--r-- 1 user user
                             185 Jun 26 10:12 ubuntu.ins
```

• Now create *ins* and *parm* files dedicated to the LPAR that will be installed (here *zlinlpar*), based on the default *ins* and *parm* files that are shipped with the ISO image:

```
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ chmod -R +rw ./boot
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ cp ./boot/ubuntu.ins ./boot/ubuntu_zlinlpar.ins
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ cp ./boot/parmfile.ubuntu ./boot/parmfile.zlinlpar
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$
```

```
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ vi ./boot/ubuntu_zlinlpar.ins
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ cat ./boot/ubuntu_zlinlpar.ins
* Ubuntu for z Series (default kernel)
kernel.ubuntu 0x00000000
initrd.off 0x0001040c
initrd.siz 0x00010414
parmfile.zlinlpar 0x00010480
initrd.ubuntu 0x01000000
admin@installserver:~$
admin@installserver:~$
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ vi ./boot/parmfile.zlinlpar
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ vi ./boot/parmfile.zlinlpar
```

ip=10.11.12.42::10.11.12.1:255.255.255.0:zlinlpar:encc000.4711:none:10.11.12.1 vlan=encc000.4711:encc000 url=http://ir daily-live-server-20.04/focal-live-server-s390x.iso autoinstall ds=nocloud-net;s=http://installserver.local:80/autoins -- quiet

- Now make sure an ftp server is running in the *installserver* with /srv/ftp as ftp-server root (as used in this example).
- Now prepare an *autoinstall* (http) server, that hosts the configuration data for the non-interactive installation.

```
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ mkdir -p /srv/www/autoinstall/zlinlpar
admin@installserver:/srv/ftp/ubuntu-daily-live-server-20.04$ cd /srv/www/autoinstall/zlinlpar
admin@installserver:/srv/www/autoinstall/zlinlpar$
admin@installserver:/srv/www/autoinstall/zlinlpar$ echo "instance-id: $(uuidgen || openssl rand -
base64 8)" > meta-data
admin@installserver:/srv/www/autoinstall/zlinlpar$ cat meta-data
instance-id: 2c2215fb-6a38-417f-b72f-376b1cc44f01
admin@installserver:/srv/www/autoinstall/zlinlpar$
admin@installserver:/srv/www/autoinstall/zlinlpar$ vi user-data
admin@installserver:/srv/www/autoinstall/zlinlpar$ cat user-data
#cloud-config
autoinstall:
 version: 1
  refresh-installer:
    update: yes
  reporting:
    builtin:
      type: print
 apt:
    preserve_sources_list: false
    primary:
    - arches: [amd64, i386]
     uri: http://archive.ubuntu.com/ubuntu
    - arches: [default]
     uri: http://ports.ubuntu.com/ubuntu-ports
  keyboard:
    layout: en
    variant: us
  locale: en_US
  identity:
    hostname: zlinlpar
    password:
"$6$ebJ1f8wxED22bTL4F46P0"
     username: ubuntu
    user-data:
      timezone: America/Boston
      users:
        - name: ubuntu
          password:
"$6$KwuxED22bTL4F46P0"
          lock_passwd: false
    early-commands:
      - touch /tmp/lets_activate_the_s390x_devices
```

```
- chzdev zfcp -e e000
    - chzdev zfcp -e e100
    - chzdev zfcp-lun -e --online
    - touch /tmp/s390x_devices_activation_done
 network:
    ethernets:
      encc000: {}
    version: 2
    vlans:
      encc000.4711:
        addresses: [10.11.12.42/24]
        gateway4: 10.11.12.1
        id: 4711
        link: encc000
        nameservers:
          addresses: [10.11.12.1]
 ssh:
    install-server: true
    allow-pw: true
    authorized-keys: ['ssh-rsa meQwtZ user@workstation # ssh-import-id lp:user']
admin@installserver:~$
```

• For s390x installations the *early-commands* section is the interesting part:

```
early-commands:
```

- touch /tmp/lets_activate_the_s390x_devices
- chzdev zfcp -e e000
- chzdev zfcp -e e100
- chzdev zfcp-lun -e --online
- touch /tmp/s390x_devices_activation_done

The first and last *early-commands* are of course optional and only frame and indicate the real s390x command activation.

In this particular example two zfcp hosts (host-bus-adapters) are enabled via it's addresses e000 (*chzdev zfcp -e* e000) and e100 (*chzdev zfcp -e* e000). These have certain LUNs assigned that are all automatically discovered and activated by *chzdev zfcp-lun -e --online*.

An activation of a DASD would look like this: $chzdev \ dasd \ -e \ 1f00$ and a QETH device activation $chzdev \ qeth \ -e \ c000$ - as usual.

- For more details about the autoinstall config options, please have a look at the autoinstall reference and autoinstall schema page.
- Now make sure a http server is running with /srv/www as web-server root (in this particular example).
- Next steps need to be done at the hardware management console *HMC*. So connect to the HMC and proceed with the 'Load From Removable Media and Server' task
- Start the 'Load from Removable Media or Server' task under 'Recovery' --> 'Load from Removable Media or Server' on your specific LPAR that you are going to install and fill out he fields like this (it will be of course different on your system):

```
FTP Source
Host computer: installserver.local
User ID: ftpuser
Password: *******
Account (optional):
File location (optional): ubuntu-daily-live-server-20.04/boot
```

- Now confirm the entered data and click 'Ok'.
- At the 'Load from Removable Media or Server Select Software to Install' screen, pick and choose the LPAR that is going to be installed, here: 'ubuntu-daily-live-server-20.04/boot/ubuntu_zlinlpar.ins Ubuntu for z Series (default kernel)'.
- Confirm again with 'Ok'.
- And another confirmation about the 'Load will cause jobs to be cancelled'.

- And even again a 'Yes', understanding that it's an disruptive task: 'Disruptive Task Confirmation : Load from Removable Media or Server'.
- Now monitor the 'Load from Removable media or Server Progress' screen and confirm it once again when the status changed from 'Please wait while the image is being loaded.' to 'Success'.
- Then navigate to 'Daily' --> 'Operating System Messages' to monitor the IPL of the install system ...

```
Message
chzdev: Unknown device type or device ID format: c000.4711
Use 'chzdev --help' for more information
QETH device 0.0.c000:0.0.c001:0.0.c002 configured
IP-Config: encc000.4711 hardware address 1a:3c:99:55:2a:ef mtu 1500
IP-Config: encc000.4711 guessed broadcast address 10.11.12.255
IP-Config: encc000.4711 complete:
address: 10.11.12.42
                      broadcast: 10.11.12.255 netmask: 255.255.255.0
gateway: 10.11.12.1 dns0 : 10.11.12.1 dns1 : 0.0.0.0
host : zlinlpar
rootserver: 0.0.0.0 rootpath:
filename :
Connecting to installserver.local:80 (installserver.local:80)
                                                       | 39.9M 0:00:15 ETA
focal-live-server-s3 5% |*
focal-live-server-s3 22% |******
                                                       | 147M 0:00:07 ETA
254M 0:00:04 ETA
                                                       | 355M 0:00:03 ETA

      focal-live-server-s3
      67%
      | *********************
      | 453M
      0:00:02
      ETA

      focal-live-server-s3
      81%
      | *******************
      | 545M
      0:00:01
      ETA

mount: mounting /cow on /root/cow failed: No such file or directory
Connecting to plymouth: Connection refused
passwd: password expiry information changed.
Using CD-ROM mount point /cdrom/
Identifying... [5d25356068b713167814807dd678c261-2]
Scanning disc for index files...
Found 2 package indexes, 0 source indexes, 0 translation indexes and 1 signature
Found label 'Ubuntu-Server 20.04 LTS _Focal Fossa_ - Release s390x (20200616)'
This disc is called:
'Ubuntu-Server 20.04 LTS _Focal Fossa_ - Release s390x (20200616)'
. . .
[ 61.190916] cloud-init[2076]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 running
'modules:final' at Fri, 26 Jun 2020 11:02:01 +0000. Up 61.09 seconds.
[ 61.191002] cloud-init[2076]: ci-info: no authorized SSH keys fingerprints fo
und for user installer.
[ 61.191071] cloud-init[2076]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 finished at Fri, 26 Jun 2020 11:02:01 +0000
Datasource DataSourceNoCloudNet [seed=cmdline,
/var/lib/cloud/seed/nocloud,
http://installserver.local:80/autoinstall/zlinlpar/]
[dsmode=net]. Up 61.18 seconds
   61.191136] cloud-init[2076]: Welcome to Ubuntu Server Installer!
[
   61.191202] cloud-init[2076]: Above you will find SSH host keys and a random
[
password set for the `installer` user. You can use these credentials to ssh-in
and complete the installation. If you provided SSH keys in the cloud-init datasource,
they were also provisioned to the installer user.
   61.191273] cloud-init[2076]: If you have access to the graphical console,
[
like TTY1 or HMC ASCII terminal you can complete the installation there too.
It is possible to connect to the installer over the network, which
might allow the use of a more capable terminal.
To connect, SSH to installer@10.11.12.42.
```

The password you should use is ''i7UFdP8fhiVVMme3qqH8''.

The host key fingerprints are:

RSA SHA256:rBXLeUke3D4gKdsruKEajHjocxc9hr3PI ECDSA SHA256:KZZYFswtKxFXWQPuQS9QpOBUoS6RHswis ED25519 SHA256:s+5tZfagx0zffC6gYRGW3t1KcBH6f+Vt0

Ubuntu 20.04 LTS ubuntu-server sclp_line0

• On short notice, you can even login to the system with the user 'installer' and the temporary password that was given at the end of the boot-up process (see above) of the installation system:

user@workstation:~\$ ssh installer@zlinlpar The authenticity of host 'zlinlpar (10.11.12.42)' can't be established. ECDSA key fingerprint is SHA256:0/dU/D8jJAEGQcbqKGE9La24IRxUPLpzzs5li9F6Vvk. Are you sure you want to continue connecting (yes/no/[fingerprint])? yes Warning: Permanently added 'zlinlpar,10.11.12.42' (ECDSA) to the list of known hosts. installer@zlinlpar's password: Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-37-generic s390x)

- * Documentation: https://help.ubuntu.com
- * Management: https://landscape.canonical.com
- * Support: https://ubuntu.com/advantage

System information as of Fri Jun 26 11:08:18 UTC 2020

System load:1.25Memory usage:4%Processes:192Usage of /home:unknownSwap usage:0%Users logged in:0

0 updates can be installed immediately.0 of these updates are security updates.

```
The list of available updates is more than a week old.
To check for new updates run: sudo apt update
```

The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

the installer running on /dev/tty1 will perform the autoinstall

press enter to start a shell

• Please notice that it informs about a running autoinstall process:

the installer running on /dev/tty1 will perform the autoinstall

• Nevertheless, we can quickly check some things - of course only until the autoinstall process got finished and the post-install reboot got triggered:

root@ubuntu-server:/# ls -l /tmp/lets_activate_the_s390x_devices
-rw-r--r-- 1 root root 0 Jun 26 11:08 /tmp/lets_activate_the_s390x_devices
-rw-r--r-- 1 root root 0 Jun 26 11:09 /tmp/s390x devices activation done

root@ubuntu-server:/# lszdev | grep yes

zfcp-host	0.0.e000	yes	yes	
zfcp-host	0.0.e100	yes	yes	
zfcp-lun	0.0.e000:0x50050763060b16b6:0x402640020000000	yes	yes	sdb sgl
zfcp-lun	0.0.e000:0x50050763061b16b6:0x402640020000000	yes	yes	sda sg0
zfcp-lun	0.0.e100:0x50050763060b16b6:0x402640020000000	yes	yes	sdd sg3

 zfcp-lun
 0.0.e100:0x50050763061b16b6:0x402640020000000
 yes
 yes
 sdc sg2

 qeth
 0.0.c000:0.0.c001:0.0.c002
 yes
 no
 encc000

 root@ubuntu-server:/#

• If you wait long enough you'll recognize that the remote session get's closed:

```
root@ubuntu-server:/# Connection to zlinlpar closed by remote host.
Connection to zlinlpar closed.
user@workstation:~$
```

• As well as at the console:

ubuntu-server login:

Total: 282 Selected: 0

[[0;1;31mFAILED[0m] Failed unmounting [0;1;39m/cdrom[0m. [169.161139] sd-umoun[15600]: Failed to unmount /oldroot: Device or resource busy [169.161550] sd-umoun[15601]: Failed to unmount /oldroot/cdrom: Device or resource busy [169.168118] shutdown[1]: Failed to finalize file systems, loop devices, ignoring

Command:

• ...and that the post-install reboot got triggered:

```
Message
Mounting [0;1;39mKernel Configuration File System[0m...
Starting [0;1;39mApply Kernel Variables[0m...
[[0;32m OK [0m] Finished [0;1;39mRemount Root and Kernel File Systems[0m.
[[0;32m OK [Om] Finished [0;1;39mUncomplicated firewall[Om.
[[0;32m OK [Om] Mounted [0;1;39mFUSE Control File System[Om.
[[0;32m OK [Om] Mounted [0;1;39mKernel Configuration File System[Om.
   35.378928] cloud-init[2565]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 runnin
ſ
g 'modules:final' at Fri, 26 Jun 2020 11:10:44 +0000. Up 35.29 seconds.
   35.378978] cloud-init[2565]: Cloud-init v. 20.1-10-g71af48df-Oubuntu5 finish
[
ed at Fri, 26 Jun 2020 11:10:44 +0000. Datasource DataSourceNone. Up 35.37 seconds
   35.379008] cloud-init[2565]: 2020-06-26 11:10:44,359 - cc_final_message.py[W
[
ARNING]: Used fallback datasource
[[0;32m OK [Om] Finished [0;1;39mExecute cloud user/final scripts[Om.
[[0;32m OK [Om] Reached target [0;1;39mCloud-init target[Om.
```

zlinlpar login:

• With the completion of the reboot the autoinstall is finished and the LPAR is ready to use:

```
user@workstation:~$ ssh-keygen -f "/home/user/.ssh/known_hosts" -R "zlinlpar"
# Host zlinlpar found: line 163
/home/user/.ssh/known_hosts updated.
Original contents retained as /home/user/.ssh/known hosts.old
user@workstation:~$ ssh ubuntu@zlinlpar
The authenticity of host 'zlinlpar (10.11.12.42)' can't be established.
ECDSA key fingerprint is SHA256:iGtCArEg+ZnoZlgt0kvmyy0gPY8UEI+f7zoIS0F+m/0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'zlinlpar, 10.11.12.42' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-39-generic s390x)
* Documentation: https://help.ubuntu.com
 * Management:
                   https://landscape.canonical.com
 * Support:
                   https://ubuntu.com/advantage
 System information as of Fri 26 Jun 2020 11:12:23 AM UTC
```

System load:0.21Memory usage:3%Processes:189Usage of /:28.2% of 30.88GBSwap usage:0%Users logged in:0

updates can be installed immediately.
 of these updates are security updates.

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY,
to the extent permitted by applicable law.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
ubuntu@zlinlpar:~$ uptime
11:12:35 up 2 min, 1 user, load average: 0.18, 0.17, 0.08
ubuntu@zlinlpar:~$ lsb release -a
No LSB modules are available.
Distributor ID: Ubuntu
                Ubuntu 20.04 LTS
Description:
Release:
            20.04
Codename:
            focal
ubuntu@zlinlpar:~$ uname -a
Linux zlinlpar 5.4.0-39-generic #43-Ubuntu SMP Fri Jun 19 10:27:17
UTC 2020 s390x s390x s390x
GNU/Linux
ubuntu@zlinlpar:~$ lszdev | grep yes
zfcp-host
            0.0.e000
                                                              ves ves
zfcp-host
             0.0.e100
                                                              yes
                                                                   yes
zfcp-lun
             0.0.e000:0x50050763060b16b6:0x402640020000000
                                                              yes
                                                                   ves
sdb sgl
zfcp-lun
             0.0.e000:0x50050763061b16b6:0x402640020000000
                                                              ves
                                                                  yes
sda sg0
zfcp-lun
             0.0.e100:0x50050763060b16b6:0x402640020000000
                                                              ves
                                                                   yes
sdc sg2
zfcp-lun
             0.0.e100:0x50050763061b16b6:0x402640020000000
                                                              ves
                                                                   ves
sdd sg3
qeth
             0.0.c000:0.0.c001:0.0.c002
                                                                  yes
                                                                         encc000
                                                              ves
ubuntu@zlinlpar:~$ exit
logout
Connection to zlinlpar closed.
user@workstation:~$
```

To see these additional updates run: apt list --upgradable

- Some closing notes:
 - It's always best to use the latest installer and autoinstall components, hence either make sure the installer get's updated to the latest level, or just use a current daily live-server image
 - The ISO image that is specified with the kernel parameters need to fit to the boot folder and it's kernel and initrd that is specified in the Load from Removable Media and Server task at the HMC
 - In addition to activating disk storage resources in 'early-commands', other devices like OSA/qeth can be added and activated there, too - but this is not needed for the basic network device, specified in the kernel parameters that is used for the installation (that one is automatically handled)
 - If everything is properly setup (ftp server for the image, http server for the autoinstall config files) and installation can be as quick as 2 to 3 minutes - of course dependent on the complexity of the autoinstall yaml file
 - There is a simple way of generating a sample autoinstall yaml file, one can just perform an interactive subiquity installation and grab the file '/var/log/installer/autoinstall-user-data' and use it as an example but notice that the early-command entries to activate the s390x-specific devices need to be added by hand!

Cloud Images

 $Canonical \ produces \ a \ variety \ of \ cloud-specific \ images, \ which \ are \ available \ directly \ on \ the \ cloud's \ themselves \ as \ well \ as \ on \ https://cloud-images.ubuntu.com.$

Public Clouds

Compute Offerings

Users can find Ubuntu images for virtual machines and bare-metal offerings published directly to the following clouds:

- Amazon Elastic Compute Cloud (EC2)
- Google Cloud Engine (GCE)
- IBM Cloud
- Microsoft Azure
- Oracle Cloud

Container Offerings

Ubuntu images are also produced for a number of container offerings:

- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)

Private Clouds

On cloud-images.ubuntu.com, users can find standard and minimal images for the following:

- Hyper-V
- KVM
- OpenStack
- Vagrant
- VMware

Release Support

Cloud images are published and supported throughout the lifecycle of an Ubuntu release. During this time images can receive all published security updates and bug fixes.

For users wanting to upgrade from one release to the next, the recommended path is to launch a new image with the desired release and then migrate whatever workload or data to the new image.

Some cloud image customization must be applied during image creation, these would be missing if an in-place upgrade were performed. For that reason in-place upgrades of cloud images are not recommended.

Amazon EC2

Amazon Web Service's Elastic Compute Cloud (EC2) provides a platform for deploying and running applications.

Images

On EC2, cloud images are referred to as Amazon Machine Images (AMIs). Canonical produces a wide variety of images to support numerous features found on EC2:

- Generally, all images utilize EBS storage and HVM virtualization types. Older releases may also support PV and instance-store, but users benefit from the newer storage and virtualization technologies.
- Standard server images as well as minimal images for amd64. As well, arm64 images for the standard server set.
- Daily (untested) and release images are published.

Find Images with SSM

The AWS Systems Manager (SSM) parameter store is used by Canonical to store the latest AMI release versions for EC2. This provides users with a programmatic method of querying for the latest AMI ID.

Canonical stores SSM parameters under /aws/service/canonical/. To find the latest AMI ID user's can use the AWS CLI:

```
aws ssm get-parameters --names \
    /aws/service/canonical/ubuntu/server/20.04/stable/current/amd64/hvm/ebs-gp2/ami-id
```

The path follows the following format:

ubuntu/\$PRODUCT/\$RELEASE/stable/current/\$ARCH/\$VIRT_TYPE/\$VOL_TYPE/ami-id

• PRODUCT: server or server-minimal

- RELEASE: focal, 20.04, bionic, 18.04, xenial, or 16.04
- ARCH: amd64 or arm64
- VIRT_TYPE: pv or hvm
- VOL_TYPE: ebs-gp2, ebs-io1, ebs-standard, or instance-store

The given serial for an image (e.g. 20210222) is also uploaded in place of current:

ubuntu/\$PRODUCT/\$RELEASE/stable/\$SERIAL/\$ARCH/\$VIRT TYPE/\$VOL TYPE/ami-id

For more details on SSM check out the following Discourse thread.

Ownership Verification

User's can verify that an AMI was published by Canonical by ensuring the OwnerId field of an image is 099720109477 (in the default partition). For the GovCloud partition, the OwnerId field is 513442679011. For the China partition, the OwnerId field is 837727238323. This ID is stored in SSM and is discoverable by running:

aws ssm get-parameters --names /aws/service/canonical/meta/publisher-id

With the value returned by that command, user's can then run the describe-images command against an AMI ID and verify the OwnerId field matches the above ID:

aws ec2 describe-images --image-ids \$AMI_ID

Note that listings on the AWS Marketplace will always show the OwnerId as Amazon (e.g. 679593333241). In these cases, users can verify the Amazon ID and look for aws-marketplace/ubuntu in the ImageLocation field.

Image Locator

Canonical also produces a Ubuntu Cloud Image Finder where users can filter down based on a variety of criteria (e.g. region, release, etc.).

AWS EKS

EKS is a managed Kubernetes service provided by AWS that lets users run Kubernetes applications in the cloud or on-premises.

Canonical provides minimized Ubuntu images customized for use with EKS. These are fully tested release images that cover all Kubernetes versions supported by the EKS service.

The latest EKS AMI ID can be found in the SSM parameter store:

aws ssm get-parameters --names /aws/service/canonical/ubuntu/eks/20.04/1.21/stable/current/amd64/hvm/ebsgp2/ami-id

The path follows the following format:

ubuntu/eks/\$RELEASE/\$K8S_VERSION/stable/current/\$ARCH/\$VIRT_TYPE/\$VOL_TYPE/ami-id

For newer Kubernetes versions (>= 1.20), there are also arm64 images available (in addition to the amd64 images)

AWS Marketplace

AWS Marketplace is a digital catalog with thousands of software listings from independent software vendors that make it easy to find, test, buy, and deploy software that runs on AWS.

Canonical maintains image listings for recent Ubuntu releases on AWS Marketplace, including images in minimal and arm64 flavors.

Those images can also be found in the SSM parameter store:

aws ssm get-parameter --name /aws/service/marketplace/<identifier>/latest

Customers can also use the AWS Marketplace to launch and subscribe to official Ubuntu Pro images that allow users to pay for additional support.

Thoses Ubuntu Pro images can also be found in the SSM parameter store:

Not all releases are in govcloud at this time. Below is a list of available products. To query govcloud, be sure to use a govcloud region and credentials

aws --region us-gov-west-1 --profile govcloud ssm get-parameter --name /aws/service/marketplace/<identifier>/latest

Please use this chart to find the product identifier and availability:

Name	Architecture	identifier	GovCloud
Ubuntu 16.04 LTS	amd64	prod-aq7wy7l65auna	\checkmark
Ubuntu 16.04 LTS	arm64	prod-zjzktzfox3gz6	
Ubuntu 18.04 LTS	amd64	prod-43pfd7pfsijnm	
Ubuntu 18.04 LTS	$\operatorname{arm}64$	prod-nanaj5afkj4gq	
Minimal Ubuntu 18.04 LTS	amd64	prod-gv4aco7axss42	× /
Minimal Ubuntu 18.04 LTS	arm64	prod-jfs2osz6fmem4	
Ubuntu 20.04 LTS	amd64	prod-x7h6cigkuiul6	
Ubuntu 20.04 LTS	$\operatorname{arm}64$	prod-gprrntd234sfc	
Minimal Ubuntu 20.04 LTS	amd64	prod-df2jln3gjtwps	\checkmark

Name	Architecture	identifier	GovCloud
Minimal Ubuntu 20.04 LTS	$\operatorname{arm}64$	prod-emtsb6upxf6us	\checkmark
Ubuntu 21.10	amd64	prod-mu37nwxj3xvxo	
Ubuntu 21.10	$\operatorname{arm}64$	prod-krrmhsgexe346	
Minimal Ubuntu 21.10	amd64	prod-ijhcxejyskj4q	
Minimal Ubuntu 21.10	arm64	prod-hobewtetsuufg	
Ubuntu 22.04 LTS	amd64	prod-lfutkwiaknxsk	
Ubuntu 22.04 LTS	arm64	prod-amd2rg3s3i7tc	
Minimal Ubuntu 22.04 LTS	amd64	prod-smq32swynllqw	
Minimal Ubuntu 22.04 LTS	$\operatorname{arm}64$	prod-zvdejcufto6ps	\checkmark

Name	Architecture	identifier	GovCloud
Ubuntu Pro FIPS 16.04 LTS	amd64	prod-hykkbajyverq4	\checkmark
Ubuntu Pro FIPS 18.04 LTS	amd64	prod-7izp2xqnddwdc	
Ubuntu Pro FIPS 20.04 LTS	amd64	prod-k6fgbnayirmrc	
Ubuntu Pro 14.04 LTS	amd64	prod-7u42cjnp5pcuo	
Ubuntu Pro 16.04 LTS	amd64	prod-f6ogoaqs7lwre	
Ubuntu Pro 18.04 LTS	amd64	prod-jlgu4232gpnwa	
Ubuntu Pro 20.04 LTS	amd64	prod-3sk4unyn4iwqu	
Ubuntu Pro 22.04 LTS	amd64	prod-uwytposjsg3du	\checkmark

Google Compute Engine

Google Cloud Platform lets you build and host applications and websites, store data, and analyze data on Google's scalable infrastructure.

Images

On GCE, Canonical produces standard server and minimal images for all supported releases.
Finding Images

Users can find the latest Ubuntu images on the GCE UI by selecting "Ubuntu" as the Operating System under the Boot Disk settings.

For a programmatic method, users can use the gcloud command to find the latest, release images:

gcloud compute images list --filter ubuntu-os-cloud

Daily, untested, images are found under the ubuntu-os-cloud-devel project:

gcloud compute images --project ubuntu-os-cloud-devel list --filter ubuntu-os-cloud-devel

Image Locator

Canonical also produces a Ubuntu Cloud Image Finder where users can filter down based on a variety of criteria (e.g. region, release, etc.).

ROCK Images - Introduction

What are ROCKs?

Ordinary software packages can often be installed in a variety of different types of environments that satisfy the given packaging system. However, these environments can be quite varied, such as including versions of language runtimes, system libraries, and other library dependencies that the software was not well tested with.

Software containers address this by encapsulating both the software and the surrounding environment. Instead of installing and maintaining a collection of software packages, the user runs and maintains a single container, instantiated from a container image with the desired software already installed. The user relies on the provider of the container image to perform the necessary software testing and maintenance updates. There is a rich ecosystem of container providers thanks mainstream tools like Docker, and popular container registries like Docker Hub, Amazon ECR, etc., which make it easy for anyone to build and publish a container image. Unfortunately, with that freedom and flexibility invariably comes unreliability of maintenance and inconsistency of implementation.

The *Open Container Initiative* (OCI) establishes standards for constructing container images that can be reliably installed across a variety of compliant host environments.

Ubuntu's LTS Docker Image Portfolio provides OCI-compliant images that receive stable security updates and predictable software updates, thus ensuring consistency in both maintenance schedule and operational interfaces for the underlying software your software builds on.

Container Creation and Deletion

Over the course of this tutorial we'll explore deriving a customized Apache container, and then networking in a Postgres container backend for it. By the end you'll have a working knowledge of how to set up a container-based environment using Canonical's ROCKs.

First the absolute basics. Let's spin up a single container providing the Apache2 web server software:

```
$ sudo apt-get update
$ sudo apt-get -y install docker.io
$ sudo docker run -d --name my-apache2-container -p 8080:80 ubuntu/apache2:2.4-22.04_beta
Unable to find image 'ubuntu/apache2:2.4-22.04_beta' locally
2.4-22.04_beta: Pulling from ubuntu/apache2
13c61b50dd15: Pull complete
34dadde438e6: Pull complete
34dadde438e6: Pull complete
Digest: sha256:11647ce68a130540150dfebbb755ee79c908103fafbf805074eb6513e6b9df83
Status: Downloaded newer image for ubuntu/apache2:2.4-22.04_beta
4031e6ed24a6e08185efd1c60e7df50f8f60c21ed9961c858ca0cb6bb300a72a
```

This container, named my-apache2-container runs in an Ubuntu 22.04 LTS environment and can be accessed via local port 8080. Load the website up in your local web browser:

\$ firefox http://localhost:8080



apache2-container 867×490 64.8 KB

If you don't have firefox handy, curl can be used instead:

\$ curl -s http://localhost:8080 | grep "<title>"
<title>Apache2 Ubuntu Default Page: It works</title>

The run command had a number of parameters to it. The Usage section of Ubuntu's Docker hub page for Apache2 has a table with an overview of parameters specific to the image, and Docker itself has a formal reference of all available parameters, but lets go over what we're doing in this particular case:

\$ sudo docker run -d --name my-apache2-container -e TZ=UTC -p 8080:80 ubuntu/apache2:2.4-22.04_beta

The -d parameter causes the container to be detached so it runs in the background. If you omit this, then you'll want to use a different terminal window for interacting with the container. The --name parameter allows you to use a defined name; if it's omitted you can still reference the container by its Docker id. The -e option lets you set environment variables used when creating the container; in this case we're just setting the timezone (TZ) to universal time (UTC). The -p parameter allows us to map port 80 of the container to 8080 on localhost, so we can reference the service as http://localhost:8080. The last parameter indicates what software image we want.

A variety of other container images are provided on Ubuntu's Docker Hub and on Amazon ECR, including documentation of supported customization parameters and debugging tips. This lists the different major/minor versions of each piece of software, packaged on top of different Ubuntu LTS releases. So for example, in specifying our requested image as ubuntu/apache2:2.4-22.04_beta we used Apache2 version 2.4 running on a Ubuntu 22.04 environment.

Notice that the image version we requested has _beta appended to it. This is called a *Channel Tag.* Like most software, Apache2 provides incremental releases numbered like 2.4.51, 2.4.52, and 2.4.53. Some of these releases are strictly bugfix-only, or even just CVE security fixes; others may include new features or other improvements. If we think of the series of these incremental releases for Apache2 2.4 on Ubuntu 22.04 as running in a *Channel*, the *Channel Tags* point to the newest incremental release that's been confirmed to the given level of stability. So, if a new incremental release 2.4.54 becomes available, ubuntu/apache2:2.4-22.04_edge images would be updated to that version rapidly, then ubuntu/apache2:2.4-22.04_beta once it's received some basic testing; eventually, if no problems are found, it will also be available in ubuntu/apache2:2.4-22.04_candidate and then in ubuntu/apache2:2.4-22.04_stable once it's validated as completely safe.

For convenience there's also a latest tag and an edge tag which are handy for experimentation or where you don't care what version is used and just want the newest available. For example, to launch the latest version of Nginx, we can do so as before, but specifying latest instead of the version:

\$ sudo docker run -d --name my-nginx-container -e TZ=UTC -p 9080:80 ubuntu/nginx:latest
4dac8d77645d7ed695bdcbeb3409a8eda942393067dad49e4ef3b8b1bdc5d584

\$ curl -s http://localhost:9080 | grep "<title>"
<title>Welcome to nginx!</title>

We've also changed the port to 9080 instead of 8080 using the -p parameter, since port 8080 is still being used by our apache2 container. If we were to try to also launch Nginx (or another Apache2 container) on port 8080, we'd get an error message, Bind for 0.0.0.0:8080 failed: port is already allocated and then would need to remove the container and try again.

Speaking of removing containers, now that we know how to create generic default containers, let's clean up:

\$ sudo docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES d86e93c98e20 ubuntu/apache2:2.4-22.04_beta "apache2-foreground" 29 minutes ago Up 29 minutes 0.0.0.0:8080->80/tcp, :::8080->80/tcp my-apache2-container eed23be5f65d ubuntu/nginx:latest "/docker-entrypoint..." 18 minutes ago Up 18 minutes 0.0.0.0:9080->80/tcp, :::9080->80/tcp my-nginx-container \$ sudo docker stop my-apache2-container

```
$ sudo docker rm my-apache2-container
```

```
$ sudo docker stop my-nginx-container
```

```
$ sudo docker rm my-nginx-container
```

To be able to actually use the containers, we'll have to configure and customize them, which we'll look at next.

Software Containers - ROCK Customization with Docker

In the last section we looked at the basics of how to start and stop containers. Here we'll apply our own modifications to the images.

You'll recall we used the -p parameter to give the two containers we created different ports so they didn't conflict with each other. We can think of this type of customization as a *container configuration*, as opposed to an *image configuration* change defined in the **Dockerfile** settings for the image itself. From a single image definition we can create an arbitrary number of different containers with different ports (or other pre-defined aspects), which are all otherwise reliably identical. A third approach is to modify the running container after it has been launched, applying whatever arbitrary changes we wish as *runtime modifications*.

- **image configuration**: Done in **Dockerfile**, changes common to all container instances of that image. Requires rebuilding the image.
- container configuration: Done at container launch, allowing variation between instances of a given image. Requires re-launching the container to change.
- runtime modifications: Done dynamically after container launch. Does not require re-launching the container.

The second approach follows Docker's immutable infrastructure principle, and is what the ROCKs system intends for production environments. For the sake of this tutorial we'll use the third approach for introductory purposes, building on that later to show how to achieve the same with only configuration at container creation time.

Setting up a Development Environment

Speaking of doing things properly, let's prepare a virtual machine (VM) to do our tutorial work in.

While you can of course install the docker.io package directly on your desktop, as you may have done in the previous section of this tutorial, using it inside a VM has a few advantages. First, it encapsulates the system changes you want to experiment with, so that they don't affect your desktop; if anything gets seriously messed up you just can delete the VM and start over. Second, it facilitates experimenting with different versions of Ubuntu, Docker, or other tools than would be available from your desktop. Third, since "The Cloud" is built with VM's, developing in a VM from the start lets you more closely emulate likely types of environments you'll be deploying to.

There are a number of different VM technologies available, any of which will suit our purposes, but for this tutorial we'll set one up using Canonical's Multipass software, which you can install on Windows using a downloadable installer, or on macOS via brew, or any flavor of Linux via snapd.

Here's how to launch a Ubuntu 22.04 VM with a bit of extra resources, and log in:

host> multipass launch --cpus 2 --mem 4G --disk 10G --name my-vm daily:20.04 host> multipass shell my-vm

If later you wish to suspend or restart the VM, use the stop/start commands:

host> multipass stop my-vm
host> multipass start my-vm

Go ahead and set up your new VM devel environment with Docker, your preferred editor, and any other tools you like having on hand:

\$ sudo apt-get update
\$ sudo apt-get -y install docker.io

Data Customization

The most basic customization for a webserver would be the index page. Let's replace the default one with the typical hello world example:

\$ echo '<html><title>Hello Docker...</title><body>Hello Docker!</body></html>' > index.html

The technique we'll use to load this into the webserver container is called *bind mounting a volume*, and this is done with the -v (or -volume) flag to docker run (not to be confused with docker -v which of course just prints the docker version). A *volume* is a file or directory tree or other data on the host we wish to provide via the container. A *bind mount* means rather than copying the data *into* the container, we establish a linkage between the local file and the file in the container. Have a look at how this works:

\$ sudo docker run -d --name my-apache2-container -e TZ=UTC -p 8080:80 -v "\${HOME}/index.html:/var/www/html/index.html" ub
...

```
$ curl http://localhost:8080
<html><title>Hello Docker...</title></html>
```

```
$ sudo docker inspect -f "{{ .Mounts }}" my-apache2-container
[{bind /home/ubuntu/index.html /var/www/html/index.html true rprivate}]
```

Watch what happens when we change the index.html contents:

```
$ echo '<html><title>...good day!</title></html>' > index.html
```

\$ curl http://localhost:8080
<html><title>...good day</title></html>

This linkage is two-way, which means that the container itself can change the data. (We mentioned *runtime modifications* earlier – this would be an example of doing that.)

\$ sudo docker exec -ti my-apache2-container /bin/bash

```
root@abcd12345678:/# echo '<html><title>Hello, again</title></html>' > /var/www/html/index.html
root@abcd12345678:/# exit
exit
```

```
$ curl http://localhost:8080
<html><title>Hello, again</title></html>
```

What if we don't want that behavior, and don't want to grant the container the ability to do so? We can set the bind mount to be read-only by appending :ro:

```
$ sudo docker stop my-apache2-container
$ sudo docker rm my-apache2-container
$ sudo docker run -d --name my-apache2-container -e TZ=UTC -p 8080:80 -v ${HOME}/index.html:/var/www/html/index.html:ro u
```

\$ sudo docker exec -ti my-apache2-container /bin/bash

```
root@abcd12345678:/# echo '<html><title>good day, sir!</title></html>' > /var/www/html/index.html
bash: /var/www/html/index.html: Read-only file system
```

root@abcd12345678:/# exit

```
$ curl http://localhost:8080
<html><title>Hello, again</title></html>
```

However, the read-only mount still sees changes on the host side:

\$ echo '<html><title>I said good day!</title></html>' > ./index.html

```
$ curl http://localhost:8080
<html><title>I said good day!</title></html>
```

This same approach can be used to seed database containers:

```
$ echo 'CREATE DATABASE my_db;' > my-database.sql
$ sudo docker run -d --name my-database -e TZ=UTC \
    -e POSTGRES_PASSWORD=mysecret \
    -v $(pwd)/my-database.sql:/docker-entrypoint-initdb.d/my-database.sql:ro \
    ubuntu/postgres:latest
```

The docker-entrypoint-initdb.d/ directory we're using here is special in that files ending in the .sql extension (or .sql.gz or .sql.xz) will be executed to the database on container initialization. Bash scripts (.sh) can also be placed in this directory to perform other initialization steps.

Let's verify the database's creation:

\$ sudo docker exec -ti my-database su postgres --command "psql my_db --command 'SELECT * FROM pg_database WHERE datistempla oid | datname | datdba | encoding | datcollate | datctype | datistemplate | datallowconn | datconnlimit | datlastsysoid | -----+

```
13761 | postgres |
            10 | 6 | en_US.utf8 | en_US.utf8 | f
                                             | t
                                                       T
            727 | 1 |
      13760 I
                                 1663 |
1 |
16384 | my_db |
             10 |
                                             | t
                     6 | en_US.utf8 | en_US.utf8 | f
                                                       727 | 1 |
                                 1663 |
1 |
      13760 |
(2 rows)
```

Debugging Techniques

Most containers are configured to make pertinent status information (such as their error log) visible through Docker's logs command:

\$ sudo docker logs my-apache2-container

```
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'Server'...
```

Sometimes this isn't sufficient to diagnose a problem. In the previous example we shelled into our container to experiment with, via:

```
$ sudo docker exec -it my-apache2-container /bin/bash
root@abcd12345678:/# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-5.15.0-25-generic root=LABEL=cloudimg-rootfs ro console=tty1 console=ttyS0
```

This places you inside a bash shell inside the container; commands you issue will be executed within the scope of the container. While tinkering around inside the container isn't suitable for normal production operations, it can be a handy way to debug problems such as if you need to examine logs or system settings. For example, if you're trying to examine the network:

```
root@abcd12345678:/# apt-get update && apt-get install -y iputils-ping iproute2
root@abcd12345678:/# ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
root@abcd12345678:/# ping my-apache2-container
ping: my-apache2-container: Name or service not known
root@abcd12345678:/# ping -c1 172.17.0.1 | tail -n2
l packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.194/0.194/0.194/0.000 ms
```

```
root@abcd12345678:/# ping -c1 172.17.0.2 | tail -n2
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.044/0.044/0.044/0.000 ms
root@abcd12345678:/# ping -c1 172.17.0.3 | tail -n2
```

1 packets transmitted, 0 received, +1 errors, 100% packet loss, time Oms

We won't use this container any further, so can remove it:

```
$ sudo docker stop my-apache2-container
```

\$ sudo docker rm my-apache2-container

Network

IP addresses may be suitable for debugging purposes, but as we move beyond individual containers we'll want to refer to them by network hostnames. First we create the network itself:

\$ sudo docker network create my-network c1507bc90cfb6100fe0e696986eb99afe64985c7c4ea44ad319f8080e640616b

\$ sudo docker	network list		
NETWORK ID	NAME	DRIVER	SCOPE
7e9ce8e7c0fd	bridge	bridge	local
6566772ff02f	host	host	local
c1507bc90cfb	my-network	bridge	local
8b992742eb38	none	null	local

Now when creating containers we can attach them to this network:

\$ sudo docker run -d --name my-container-0 --network my-network ubuntu/apache2:latest
\$ sudo docker run -d --name my-container-1 --network my-network ubuntu/apache2:latest

\$ sudo docker exec -it my-container-0 /bin/bash root@abcd12345678:/# apt-get update && apt-get install -y iputils-ping bind9-dnsutils root@abcd12345678:/# ping my-container-1 -c 1| grep statistics -A1 --- my-container-1 ping statistics ---1 packets transmitted, 1 received, 0% packet loss, time 0ms

root@abcd12345678:/# dig +short my-container-0 my-container-1
172.18.0.2
172.18.0.3

root@abcd12345678:/# exit

\$ sudo docker stop my-container-0 my-container-1
\$ sudo docker rm my-container-0 my-container-1

A common use case for networked containers is load balancing. Docker's --network-alias option provides one means of setting up *round-robin* load balancing at the network level during container creation:

\$ sudo docker run -d --name my-container-0 --network my-network --network-alias my-website -e TZ=UTC p 8080:80 -v \${HOME}/index.html:/var/www/html/index.html:ro ubuntu/apache2:latest
\$ sudo docker run -d --name my-container-1 --network my-network --network-alias my-website -e TZ=UTC p 8081:80 -v \${HOME}/index.html:/var/www/html/index.html:ro ubuntu/apache2:latest
\$ sudo docker run -d --name my-container-2 --network my-network --network-alias my-website -e TZ=UTC p 8082:80 -v \${HOME}/index.html:/var/www/html/index.html:ro ubuntu/apache2:latest

\$ sudo docker ps PORTS CONTAINER ID IMAGE COMMAND CREATED STATUS NAMES "apache2-foreground" 665cf336ba9c Up 4 days 0.0.0.0:8082ubuntu/apache2:latest 4 days ago >80/tcp, :::8082->80/tcp my-container-2 fd952342b6f8 ubuntu/apache2:latest "apache2-foreground" 4 days ago Up 4 days 0.0.0.0:8081->80/tcp, :::8081->80/tcp my-container-1 0592e413e81d ubuntu/apache2:latest "apache2-foreground" Up 4 days 0.0.0.0:8080-4 days ago >80/tcp, :::8080->80/tcp mv-container-0

The my-website alias selects a different container for each request it handles, allowing load to be distributed across all of them.

\$ sudo docker exec -it my-container-0 /bin/bash root@abcd12345678:/# apt update; apt install -y bind9-dnsutils root@abcd12345678:/# dig +short my-website 172.18.0.3 172.18.0.2 172.18.0.4

Run that command several times, and the output should display in a different order each time.

```
root@abcd12345678:/# dig +short my-website
172.18.0.3
172.18.0.4
172.18.0.2
root@abcd12345678:/# dig +short my-website
172.18.0.2
172.18.0.3
172.18.0.4
root@abcd12345678:/# exit
$ sudo docker stop my-container-0 my-container-1 my-container-2
$ sudo docker rm my-container-0 my-container-1 my-container-2
```

Installing Software

By default Apache2 can serve static pages, but for more than that it's necessary to enable one or more of its modules. As we mentioned above, there are three approaches you could take: Set things up at runtime by logging into the container and running commands directly; configuring the container at creation time; or, customizing the image definition itself.

Ideally, we'd use the second approach to pass a parameter or setup.sh script to install software and run a2enmod <mod>, however the Apache2 image lacks the equivalent of Postgres' /docker-entrypoint-initdb.d/ directory and automatic processing of shell scripts. So for a production system you'd need to derive your own customized Apache2 image and build containers from that.

For the purposes of this tutorial, though, we can use the runtime configuration approach just for experimental purposes.

First, create our own config file that enables CGI support:

```
$ cat > ~/my-apache2.conf << 'EOF'</pre>
User ${APACHE RUN USER}
Group ${APACHE RUN GROUP}
ErrorLog ${APACHE_LOG_DIR}/error.log
ServerName localhost
HostnameLookups Off
LogLevel warn
Listen 80
# Include module configuration:
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
<Directory />
        AllowOverride None
        Require all denied
</Directory>
<Directory /var/www/html/>
        AllowOverride None
        Require all granted
</Directory>
<Directory /var/www/cgi-bin/>
        AddHandler cgi-script .cgi
        AllowOverride None
        Options +ExecCGI -MultiViews
        Require all granted
</Directory>
<VirtualHost *:80>
        DocumentRoot /var/www/html/
        ScriptAlias /cgi-bin/ /var/www/cgi-bin/
</VirtualHost>
EOF
Next, copy the following into a file named fortune.cgi.
```

```
$ cat > ~/fortune.cgi << 'EOF'</pre>
#!/usr/bin/env bash
echo -n -e "Content-Type: text/plain\n\n"
echo "Hello ${REMOTE_ADDR}, I am $(hostname -f) at ${SERVER_ADDR}"
echo "Today is $(date)"
if [ -x /usr/games/fortune ]; then
    /usr/games/fortune
fi
EOF
$ chmod a+x ~/fortune.cgi
Now create our container:
$ sudo docker run -d --name my-fortune-cgi -e TZ=UTC -p 9080:80 \
     -v $(pwd)/my-apache2.conf:/etc/apache2/apache2.conf:ro \
     -v $(pwd)/fortune.cgi:/var/www/cgi-bin/fortune.cgi:ro \
     ubuntu/apache2:latest
c3709dc03f24fbf862a8d9499a03015ef7ccb5e76fdea0dc4ac62a4c853597bf
Next, perform the runtime configuration steps:
$ sudo docker exec -it my-fortune-cgi /bin/bash
root@abcd12345678:/# apt-get update && apt-get install -y fortune
root@abcd12345678:/# a2enmod cgid
root@abcd12345678:/# service apache2 force-reload
Finally, restart the container so our changes take effect:
$ sudo docker restart my-fortune-cgi
my-fortune-cgi
Let's test it out:
$ curl http://localhost:9080/cgi-bin/fortune.cgi
Hello 172.17.0.1, I am 8ace48b71de7 at 172.17.0.2
Today is Wed Jun 1 16:59:40 UTC 2022
Q: Why is Christmas just like a day at the office?
   You do all of the work and the fat guy in the suit
A:
        gets all the credit.
Finally is cleanup, if desired:
```

```
Next
```

\$ sudo docker stop my-fortune-cgi
\$ sudo docker rm my-fortune-cgi

While it's interesting to be able to customize a basic container, how can we do this without resorting to runtime configuration? As well, a single container by itself is not terrible useful, so in the next section we'll practice setting up a database node to serve data to our webserver.

Software Containers - Multi-Node ROCK Configuration with Docker-Compose

The prior section explained the use of a single container for running a single software instance, but the principle benefit of using ROCKs is the ability to easily create and architecturally organize, or "orchestrate", them to operate together in a modular fashion.

If you set up a VM while following that section, you can continue to use that here, or if not feel free to create a new VM for this section, using those same directions.

Colors Web App

This section will demonstrate use of docker-compose to set up two nodes that inter-operate to implement a trivial CGI web app that lets the user select a background color from the standard rgb.txt color codes. Here's the table definition itself:

```
$ cat > ~/my-color-database.sql <<'EOF'
CREATE DATABASE my_color_db;
CREATE TABLE "color"
(
    id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    red INTEGER,
    green INTEGER,
    blue INTEGER,
    colorname VARCHAR NOT NULL
);</pre>
```

```
REVOKE ALL ON "color" FROM public;
GRANT SELECT ON "color" TO "postgres";
```

E0F

For the data, we'll scarf up X11's rgb.txt file, which should be readily at hand with most Ubuntu desktop installations:

```
$ sudo apt-get install x11-common
$ grep -v ^! /usr/share/X11/rgb.txt | \
  awk 'BEGIN{print "INSERT INTO color(red, green, blue, colorname) VALUES"}
     $1 != $2 || $2 != $3 {
                   (%d, %d, %d, '\''', $1, $2, $3);
         printf("
         for (i = 4; i <= NF; i++) {
            printf("%s", $i);
         }
         printf("'\''),\n");
    }
  END {print "
                  (0, 0, 0, '\''black'\'');"}' >> ~/my-color-database.sql
Here's the corresponding CGI script:
$ cat > ~/my-colors.cgi <<'EOF'</pre>
#!/usr/bin/env python3
import cgi
import psycopg2
# Get web form data (if any)
query_form = cgi.FieldStorage()
if 'bgcolor' in query_form.keys():
   bgcolor = query_form["bgcolor"].value
else:
   bgcolor = 'FFFFFF'
print("Content-Type: text/html\n\n");
# Head
body style = "body { background-color: #%s; }" %(bgcolor)
text_style = ".color-invert { filter: invert(1); mix-blend-mode: difference; }"
print(f"<html>\n<head><style>\n{body_style}\n{text_style}\n</style></head>\n")
print("<body>\n<h1 class=\"color-invert\">Pick a background color:</h1>\n")
print("\n")
print(" ColorNameCode
# Connect database
db = psycopg2.connect(host='examples_postgres_1', user='postgres', password='myS&cret')
# Display the colors
colors = db.cursor()
colors.execute("SELECT * FROM color;")
for row in colors.fetchall():
   code = ''.join('{:02X}'.format(a) for a in row[1:4])
   color = row[4]
```

```
print(f" \n")
print(f" <a href=\"my-colors.cgi?bgcolor={code}\">{color}\n")
print(f" {code}
```

```
# Foot
print("\n")
print("</body>\n</html>\n")
FOF
```

By default, Apache2 is configured to allow CGI scripts in the /usr/lib/cgi-bin system directory, but rather than installing the script there, let's use our own directory to serve from:

```
$ cat > ~/my-apache.conf <<'EOF'</pre>
User ${APACHE RUN USER}
Group ${APACHE_RUN GROUP}
ErrorLog ${APACHE_LOG_DIR}/error.log
ServerName localhost
HostnameLookups Off
LogLevel warn
Listen 80
# Include module configuration:
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
<Directory />
        AllowOverride None
        Require all denied
</Directory>
<Directory /var/www/html/>
        AllowOverride None
        Require all granted
</Directory>
<Directory /var/www/cgi-bin/>
        AddHandler cgi-script .cgi
        AllowOverride None
        Options +ExecCGI -MultiViews
        Require all granted
</Directory>
<VirtualHost *:80>
        DocumentRoot /var/www/html/
        ScriptAlias /cgi-bin/ /var/www/cgi-bin/
</VirtualHost>
E0F
```

Install Docker Compose

With our web app developed, we're ready to containerize it. We'll install Docker Compose, pull in the two base images for the database and web server, and create our own containers with our web app files and configuration layered on top.

First, install what we'll need:

```
$ sudo apt-get update
$ sudo apt-get install -y docker.io docker-compose
```

Create Database Container

Next, prepare the Postgres container. Each of Ubuntu's Docker Images has a git repository, referenced from the respective Docker Hub page. These repositories include some example content that we can build from:

```
$ git clone https://git.launchpad.net/~canonical-server/ubuntu-docker-images/+git/postgresql my-postgresql-
oci
```

\$ cd my-postgresql-oci/ \$ git checkout origin/14-22.04 -b my-postgresql-oci-branch \$ find ./examples/ -type f ./examples/README.md

./examples/postgres-deployment.yml

./examples/docker-compose.yml

./examples/config/postgresql.conf

Notice the two YAML files. The docker-compose.yml file lets us create a derivative container where we can insert our own customizations such as config changes and our own SQL data to instantiate our database. (The other YAML file is for Kubernetes-based deployments.)

```
$ mv -iv ~/my-color-database.sql ./examples/
renamed '/home/ubuntu/my-color-database.sql' -> './examples/my-color-database.sql'
$ git add ./examples/my-color-database.sql
```

Modify the services section of the file examples/docker-compose.yml to look like this:

services:

```
postgres:
    image: ubuntu/postgres:14-22.04_beta
    ports:
        - 5432:5432
    environment:
        - POSTGRES_PASSWORD=myS&cret
    volumes:
        - ./config/postgresql.conf:/etc/postgresql/postgresql.conf:ro
        - ./my-color-database.sql:/docker-entrypoint-initdb.d/my-color-database.sql:ro
```

The volumes section of the file lets us bind files from our local git repository into our new container. Things like the postgresql.conf configuration file get installed to the normal system as you'd expect.

But the /docker-entrypoint-initdb.d/ directory will look unusual – this is a special directory provided by Ubuntu's Postgres Docker container that will automatically run .sql (or .sql.gz or .sql.xz) and .sh files through the psql interpreter during initialization, in POSIX alphanumerical order. In our case we have a single .sql file that we want invoked during initialization.

Ubuntu's ROCKs are also built with environment variables to customize behavior; above we can see where we can specify our own password.

Commit everything so far to our branch:

so you can watch the service log info live.)

```
$ git commit -a -m "Add a color database definition"
[my-postgresql-oci-branch 0edeb20] Add a color database definition
2 files changed, 549 insertions(+)
create mode 100644 examples/my-color-database.sql
```

Now we're ready to create and start our application's database container:

```
$ cd ./examples/
$ sudo docker-compose up -d
Pulling postgres (ubuntu/postgres:edge)...
...
Creating examples_postgres_1 ... done
$ sudo docker-compose logs
...
postgres_1 | /usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/my-color-database.sql
...
postgres_1 | 2022-06-02 03:14:28.040 UTC [1] LOG: database system is ready to accept connections
The -d flag causes the container to run in the background (you might omit it if you want to run it in its own window
```

Note that if there is an error, such as a typo in your .sql file, you can't just re-run docker-compose up (or restart) because it'll attempt to re-attach and may appear successful at first glance:

```
...
postgres_1 | psql:/docker-entrypoint-initdb.d/my-color-database.sql:10: ERROR: type "sometypo" does not exist
postgres_1 | LINE 3: "id" SOMETYPO,
```

postgres_1 | ^
examples_postgres_1 exited with code 3

\$ sudo docker-compose up Starting examples_postgres_1 ... done Attaching to examples_postgres_1 postgres_1 | postgres_1 | PostgreSQL Database directory appears to contain a database; Skipping initialization ... postgres_1 | 2022-06-02 04:00:51.400 UTC [25] LOG: database system was not properly shut down; automatic recovery in prog ... postgres 1 | 2022-06-02 04:00:51.437 UTC [1] LOG: database system is ready to accept connections

However, while there is a live database, our data didn't load into it so it is invalid.

Instead, always issue a down command before attempting a restart when fixing issues:

\$ sudo docker-compose down; sudo docker-compose up

. . .

Note that in our environment docker-compose needs to be run with root permissions; if it isn't, you may see an error similar to this:

```
ERROR: Couldn't connect to Docker daemon at http+docker://localhost - is it running?
If it's at a non-standard location, specify the URL with the DOCKER_HOST environment variable.
```

At this point we could move on to the webserver container, but we can double-check our work so far by installing the Postgres client locally in the VM and running a sample query:

```
$ sudo apt-get install postgresql-client
$ psql -h localhost -U postgres
Password for user postgres: myS&cret
postgres=# \d
           List of relations
Schema |
           Name
                 | Type | Owner
| table
public | color
                          | postgres
public | color_id_seq | sequence | postgres
(2 rows)
postgres=# SELECT * FROM color WHERE id<4;</pre>
id | red | green | blue | colorname
1 | 255 | 250 | 250 | snow
 2 | 248 | 248 | 255 | ghostwhite
 3 | 248 | 248 | 255 | GhostWhite
(3 rows)
```

Create Webserver Docker Container

Now we do the same thing for the Apache2 webserver.

Get the example files from Canonical's Apache2 image repository via git:

```
$ cd ~
$ git clone https://git.launchpad.net/~canonical-server/ubuntu-docker-images/+git/apache2 my-apache2-oci
$ cd my-apache2-oci/
$ git checkout origin/2.4-22.04 -b my-apache2-oci-branch
$ find ./examples/ -type f
./examples/apache2-deployment.yml
./examples/apache2-deployment.yml
./examples/README.md
./examples/docker-compose.yml
./examples/config/apache2.conf
./examples/config/html/index.html
$ mv -ivf ~/my-apache2.conf ./examples/config/apache2.conf'
$ mv -iv ~/my-colors.cgi ./examples/
```

renamed '/home/ubuntu/my-colors.cgi' -> 'examples/my-colors.cgi'
\$ chmod a+x ./examples/my-colors.cgi
\$ git add ./examples/config/apache2.conf ./examples/my-colors.cgi

Modify the examples/docker-compose.yml file to look like this:

Commit everything to the branch:

\$ git commit -a -m "Add a color CGI web application"

Now launch the web server container:

\$ cd ./examples/

```
$ sudo docker-compose up -d
```

You will now be able to connect to the service:

\$ firefox http://localhost:8080/cgi-bin/my-colors.cgi?bgcolor=FFDEAD

a	Mozilla	Firefox					^	-	o x
10.170.67.210:8080/cgi-b× +									
< → C @	0 8 10.170.67.210:8	080/cgi-bin/my_c	פ ג	¥	lii\	9	А	»	=
Pick a background color:									
	longiouna								- 1
Color	Name	Code							
snow	#FFFAFA								
<u>ghostwhite</u>	#F8F8FF								
GhostWhite	#F8F8FF								
floralwhite	#FFFAF0								
FloralWhite	#FFFAF0								
oldlace	#FDF5E6								
OldLace	#FDF5E6								
linen	#FAF0E6								
antiquewhite	#FAEBD7								
AntiqueWhite	#FAEBD7								
<u>papayawhip</u>	#FFEFD5								
<u>PapayaWhip</u>	#FFEFD5								
blanchedalmond	#FFEBCD								
<u>BlanchedAlmond</u>	#FFEBCD								
<u>bisque</u>	#FFE4C4								
<u>peachpuff</u>	#FFDAB9								
<u>PeachPuff</u>	#FFDAB9								
<u>navajowhite</u>	#FFDEAD								
<u>NavajoWhite</u>	#FFDEAD								
<u>moccasin</u>	#FFE4B5								
<u>cornsilk</u>	#FFF8DC								
<u>ivory</u>	#FFFFF0								
lemonchiffon	#FFFACD								
LemonChiffon	#FFFACD								
<u>seashell</u>	#FFF5EE								
<u>honeydew</u>	#F0FFF0								
<u>mintcream</u>	#F5FFFA								
<u>MintCream</u>	#F5FFFA								
azure	#F0FFFF								
aliceblue	#F0F8FF								
AliceBlue	#F0F8FF								
lavender	#E6E6FA								
lavenderblush	#FFF0F5								

colors_screenshot_1717 $\times780$ 82.5 KB

Click on one of the colors to see the background color change:

•		Mozilla Firefox							^	-	o x
10 170 57 210 8080#	rality +										
10.170.07.210.8080/0	- Glick -										
< → ଫ 🏠	O 🏝 10.170.67.210:8	080/cgi-bin/my_colors.cgi?bgcolor=FFDEAD	습	⊌	坐	jit/	А	۲	8	6	*
Pick a ba	ckground co	lor:									
Color	Name	Code									
snow	FFFAFA										
ghostwhite	F8F8FF										
GhostWhite	F8F8FF										
floralwhite	FFFAF0										
FloralWhite	FFFAF0										
oldlace	FDF5E6										
OldLace	FDF5E6										
linen	FAF0E6										
antiquewhite	FAEBD7										
AntiqueWhite	FAEBD7										
papayawhip	FFEFD5										
PapayaWhip	FFEFD5										
blanchedalmond	FFEBCD										
BlanchedAlmond	FFEBCD										
bisque	FFE4C4										
peachpuff	FFDAB9										
PeachPuff	FFDAB9										
navajowhite	FFDEAD										
NavajoWhite	FFDEAD										
moccasin	FFE4B5										
cornsilk	FFF8DC										
ivory	FFFFF0										
lemonchitton	FFFACD										
LemonChitton	FFFACD										
seashell	FFF5EE										
honeydew	FOFFFO										
mintcream	FSFFFA										
MintCream	Forffa										
azure	FUFFFF										
anceblue	FUFSFF										
AllceBlue	FUFGFF										
lavender	EbE6FA										
lavenderblush	FFF0F5										

colors_screenshot_21094 $\times 780$ 84 KB

Once you're done, if you wish you can cleanup the containers as before, or if you used Multipass you can shutdown and delete the VM:

```
$ exit
host> multipass stop my-vm
host> multipass delete my-vm
```

Next Steps

As you can see, docker-compose makes it convenient to set up multi-container applications without needing to perform runtime changes to the containers. As you can imagine, this can permit building a more sophisticated management system to handle fail-over, load-balancing, scaling, upgrading old nodes, and monitoring status. But rather than needing to implement all of this directly on top of docker-container, you can next investigate Kubernetes-style cluster management software such as microk8s.

Package Management

Ubuntu features a comprehensive package management system for installing, upgrading, configuring, and removing software. In addition to providing access to an organized base of over 60,000 software packages for your Ubuntu computer, the package management facilities also feature dependency resolution capabilities and software update checking.

Several tools are available for interacting with Ubuntu's package management system, from simple command-line utilities which may be easily automated by system administrators, to a graphical interface which is easy to use by those new to Ubuntu.

Introduction

Ubuntu's package management system is derived from the same system used by the Debian GNU/Linux distribution. The package files contain all of the necessary files, meta-data, and instructions to implement a particular functionality or software application on your Ubuntu computer.

Debian package files typically have the extension .deb, and usually exist in *repositories* which are collections of packages found online or on physical media, such as CD-ROM discs. Packages are normally in a pre-compiled binary format; thus installation is quick and requires no compiling of software.

Many packages use *dependencies*. Dependencies are additional packages required by the principal package in order to function properly. For example, the speech synthesis package festival depends upon the package alsa-utils, which is a package supplying the ALSA sound library tools needed for audio playback. In order for festival to function, it and all of its dependencies must be installed. The software management tools in Ubuntu will do this automatically.

\mathbf{Apt}

The apt command is a powerful command-line tool, which works with Ubuntu's Advanced Packaging Tool (APT) performing such functions as installation of new software packages, upgrade of existing software packages, updating of the package list index, and even upgrading the entire Ubuntu system.

Some examples of popular uses for the apt utility:

• **Install a Package**: Installation of packages using the apt tool is quite simple. For example, to install the nmap network scanner, type the following:

sudo apt install nmap

• **Remove a Package**: Removal of a package (or packages) is also straightforward. To remove the package installed in the previous example, type the following:

sudo apt remove nmap

Tip

Multiple Packages: You may specify multiple packages to be installed or removed, separated by spaces.

Notice

Scripting: While apt is a command-line tool, it is intended to be used interactively, and not to be called from non-interactive scripts. The apt-get command should be used in scripts (perhaps with the --quiet flag). For basic commands the syntax of the two tools is identical.

Also, adding the --purge option to apt remove will remove the package configuration files as well. This may or may not be the desired effect, so use with caution.

• Update the Package Index: The APT package index is essentially a database of available packages from the repositories defined in the /etc/apt/sources.list file and in the /etc/apt/sources.list.d directory. To update the local package index with the latest changes made in the repositories, type the following:

sudo apt update

• **Upgrade Packages**: Over time, updated versions of packages currently installed on your computer may become available from the package repositories (for example security updates). To upgrade your system, first, update your package index as outlined above, and then type:

sudo apt upgrade

For information on upgrading to a new Ubuntu release see Upgrading.

Actions of the apt command, such as installation and removal of packages, are logged in the /var/log/dpkg.log log file.

For further information about the use of APT, read the comprehensive APT User's Guide or type:

apt help

Aptitude

Launching Aptitude with no command-line options will give you a menu-driven, text-based front-end to the *Advanced Packaging Tool* (APT) system. Many of the common package management functions, such as installation, removal, and upgrade, can be performed in Aptitude with single-key commands, which are typically lowercase letters.

Aptitude is best suited for use in a non-graphical terminal environment to ensure proper functioning of the command keys. You may start the menu-driven interface of Aptitude as a normal user by typing the following command at a terminal prompt:

sudo aptitude

When Aptitude starts, you will see a menu bar at the top of the screen and two panes below the menu bar. The top pane contains package categories, such as *New Packages* and *Not Installed Packages*. The bottom pane contains information related to the packages and package categories.

Using Aptitude for package management is relatively straightforward, and the user interface makes common tasks simple to perform. The following are examples of common package management functions as performed in Aptitude:

- **Install Packages**: To install a package, locate the package via the *Not Installed Packages* package category, by using the keyboard arrow keys and the ENTER key. Highlight the desired package, then press the + key. The package entry should turn *green*, indicating that it has been marked for installation. Now press g to be presented with a summary of package actions. Press g again, and downloading and installation of the package will commence. When finished, press ENTER, to return to the menu.
- **Remove Packages**: To remove a package, locate the package via the *Installed Packages* package category, by using the keyboard arrow keys and the ENTER key. Highlight the desired package you wish to remove, then press the key. The package entry should turn *pink*, indicating it has been marked for removal. Now press g to be presented with a summary of package actions. Press g again, and removal of the package will commence. When finished, press ENTER, to return to the menu.
- **Update Package Index**: To update the package index, simply press the u key. Updating of the package index will commence.
- **Upgrade Packages**: To upgrade packages, perform the update of the package index as detailed above, and then press the U key to mark all packages with updates. Now press g whereby you'll be presented with a summary of package actions. Press g again, and the download and installation will commence. When finished, press ENTER, to return to the menu.

The first column of the information displayed in the package list in the top pane, when actually viewing packages lists the current state of the package, and uses the following key to describe the state of the package:

- i: Installed package
- c: Package not installed, but package configuration remains on the system
- **p**: Purged from system
- v: Virtual package
- B: Broken package
- u: Unpacked files, but package not yet configured
- C: Half-configured Configuration failed and requires fix
- H: Half-installed Removal failed and requires a fix

To exit Aptitude, simply press the q key and confirm you wish to exit. Many other functions are available from the Aptitude menu by pressing the F10 key.

Command Line Aptitude

You can also use Aptitude as a command-line tool, similar to apt. To install the nmap package with all necessary dependencies, as in the apt example, you would use the following command:

sudo aptitude install nmap

To remove the same package, you would use the command:

sudo aptitude remove nmap

Consult the man pages for more details of command-line options for Aptitude.

dpkg

dpkg is a package manager for *Debian*-based systems. It can install, remove, and build packages, but unlike other package management systems, it cannot automatically download and install packages or their dependencies. Apt and Aptitude are newer, and layer additional features on top of dpkg. This section covers using dpkg to manage locally installed packages:

• To list all packages in the system's package database, including all packages, installed and uninstalled, from a terminal prompt type:

dpkg -l

• Depending on the number of packages on your system, this can generate a large amount of output. Pipe the output through grep to see if a specific package is installed:

dpkg -l | grep apache2

Replace apache2 with any package name, part of a package name, or a regular expression.

• To list the files installed by a package, in this case the ufw package, enter:

dpkg -L ufw

• If you are not sure which package installed a file, dpkg -S may be able to tell you. For example:

```
dpkg -S /etc/host.conf
base-files: /etc/host.conf
```

The output shows that the /etc/host.conf belongs to the base-files package.

Note

Many files are automatically generated during the package install process, and even though they are on the filesystem, dpkg -S may not know which package they belong to.

• You can install a local .deb file by entering:

sudo dpkg -i zip_3.0-4_amd64.deb

Change zip_3.0-4_amd64.deb to the actual file name of the local .deb file you wish to install.

• Uninstalling a package can be accomplished by:

sudo dpkg -r zip

Caution

Uninstalling packages using dpkg, in most cases, is *NOT* recommended. It is better to use a package manager that handles dependencies to ensure that the system is in a consistent state. For example, using dpkg -r zip will remove the zip package, but any packages that depend on it will still be installed and may no longer function correctly.

For more dpkg options see the man page: man dpkg.

APT Configuration

Configuration of the *Advanced Packaging Tool* (APT) system repositories is stored in the /etc/apt/sources.list file and the /etc/apt/sources.list.d directory. An example of this file is referenced here, along with information on adding or removing repository references from the file.

You may edit the file to enable repositories or disable them. For example, to disable the requirement of inserting the Ubuntu CD-ROM whenever package operations occur, simply comment out the appropriate line for the CD-ROM, which appears at the top of the file:

```
# no more prompting for CD-ROM please
# deb cdrom:[DISTRO-APT-CD-NAME - Release i386 (20111013.1)]/ DISTRO-SHORT-CODENAME main restricted
```

Extra Repositories

In addition to the officially supported package repositories available for Ubuntu, there exist additional communitymaintained repositories which add thousands more packages for potential installation. Two of the most popular are the *universe* and *multiverse* repositories. These repositories are not officially supported by Ubuntu, but because they are maintained by the community they generally provide packages which are safe for use with your Ubuntu computer.

Note

Packages in the *multiverse* repository often have licensing issues that prevent them from being distributed with a free operating system, and they may be illegal in your locality.

Warning

Be advised that neither the *universe* or *multiverse* repositories contain officially supported packages. In particular, there may not be security updates for these packages.

Many other package sources are available, sometimes even offering only one package, as in the case of package sources provided by the developer of a single application. You should always be very careful and cautious when using nonstandard package sources, however. Research the source and packages carefully before performing any installation, as some package sources and their packages could render your system unstable or non-functional in some respects.

By default, the *universe* and *multiverse* repositories are enabled but if you would like to disable them edit /etc/apt/sources.list and comment the following lines:

deb http://archive.ubuntu.com/ubuntu DISTRO-SHORT-CODENAME universe multiverse
deb-src http://archive.ubuntu.com/ubuntu DISTRO-SHORT-CODENAME universe multiverse

deb http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME universe deb-src http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME universe deb http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME-updates universe deb-src http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME-updates universe

deb http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME multiverse deb-src http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME multiverse deb http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME-updates multiverse deb-src http://us.archive.ubuntu.com/ubuntu/ DISTRO-SHORT-CODENAME-updates multiverse

deb http://security.ubuntu.com/ubuntu DISTRO-SHORT-CODENAME-security universe deb-src http://security.ubuntu.com/ubuntu DISTRO-SHORT-CODENAME-security universe deb http://security.ubuntu.com/ubuntu DISTRO-SHORT-CODENAME-security multiverse deb-src http://security.ubuntu.com/ubuntu DISTRO-SHORT-CODENAME-security multiverse

Automatic Updates

The unattended-upgrades package can be used to automatically install updated packages and can be configured to update all packages or just install security updates. First, install the package by entering the following in a terminal:

sudo apt install unattended-upgrades

To configure unattended-upgrades, edit /etc/apt/apt.conf.d/50unattended-upgrades and adjust the following to fit your needs:

```
Unattended-Upgrade::Allowed-Origins {
    "${distro_id}:${distro_codename}";
    "${distro_id}:${distro_codename}-security";
// "${distro_id}:${distro_codename}-updates";
// "${distro_id}:${distro_codename}-proposed";
// "${distro_id}:${distro_codename}-backports";
};
```

Certain packages can also be excluded and therefore will not be automatically updated. To block a package, add it to the list:

Unattended-Upgrade::Package-Blacklist {

```
// "vim";
// "libc6";
// "libc6-dev";
// "libc6-i686";
};
```

Note

The double "//" serve as comments, so whatever follows "//" will not be evaluated.

To enable automatic updates, edit /etc/apt/apt.conf.d/20auto-upgrades and set the appropriate apt configuration options:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Download-Upgradeable-Packages "1";
APT::Periodic::AutocleanInterval "7";
APT::Periodic::Unattended-Upgrade "1";
```

The above configuration updates the package list, downloads, and installs available upgrades every day. Those actions are triggered by timer units at a set time but with a random delay: apt-daily.timer and apt-daily-upgrade.timer. These timers activate the correspondent services that run /usr/lib/apt/apt.systemd.daily script.

However, it may happen that if the server is off at the time the timer unit elapses, the timer will be triggered immediately at the next startup. As a result, they will often run on system startup and thereby cause immediate activity and hold the apt-lock.

In many cases that is beneficial, but in some cases it might be counter-productive. Examples are administrators with many shut down machines or VM images that are only started for some quick action that then is delayed or even blocked by the unattended upgrades. To adapt this behaviour, we can change/override the configuration of both apt's timer units [apt-daily-upgrade.timer, apt-daily.timer]. To do so use systemctl edit <timer_unit> and override the *Persistent* attribute like Persistent=delay (example of such an override):

[Timer]

Persistent=delay

The local download archive is cleaned every week. On servers upgraded to newer versions of Ubuntu, depending on your responses, the file listed above may not be there. In this case, creating a new file of this name should also work.

Note

You can read more about apt Periodic configuration options in the apt.conf(5) manpage and in the /usr/lib/apt/apt.systemd.daily script header.

The results of unattended-upgrades will be logged to /var/log/unattended-upgrades.

Notifications

Configuring Unattended-Upgrade::Mail in /etc/apt/apt.conf.d/50unattended-upgrades will enable unattended-upgrades to email an administrator detailing any packages that need upgrading or have problems.

Another useful package is apticron. apticron will configure a cron job to email an administrator information about any packages on the system that have updates available, as well as a summary of changes in each package.

To install the apticron package, in a terminal enter:

sudo apt install apticron

Once the package is installed edit /etc/apticron/apticron.conf, to set the email address and other options:

EMAIL="root@example.com"

References

Most of the material covered in this chapter is available in man pages, many of which are available online.

- The InstallingSoftware Ubuntu wiki page has more information.
- For more dpkg details see the dpkg man page.
- The APT User's Guide and apt man page contain useful information regarding apt usage.
- For more info about systemd timer units (and systemd in general), visit systemd man page and systemd.timer man page.
- See the aptitude user's manual for more aptitude options.
- The Adding Repositories HOWTO (Ubuntu Wiki) page contains more details on adding repositories.

Software Bill of Materials (SBOM)

A Software Bill of Materials (SBOM) in the context of Ubuntu is a list of software components, applications, libraries, and modules that are installed by the operating system. Ubuntu consists of several packages that depend on other packages such as shared libraries. These are shipped on different forms such as debs and snaps. The following sections show how to get the list for each type of packages to assist you in keeping your systems up-to-date.

List of deb packages

To see the package dependencies in Ubuntu use:

\$ dpkg -l

The output is a list of packages similarly to the following.

Desired=Unknown/Install/Remove/Purge/Hold

| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend

// Err?=(none)/Reinst-required (Status,Err: uppercase=bad)

/ Name	Version		Architecture Description			
+++-===================================			============================	=== -		
ii apparmor	3.0.3-0ubun	tul	amd64	user-		
space parser utility for AppArmor						
rc apparmor-utils	3.0.0-0ubuntu7.1	amd64	utilities for contr	olling AppArmor		
management						
ii apt	2.3.9	amd64	commandline package	manager		

List of snap packages

\$ snap list

This outputs a list such as the one below.

Name	Version	Rev	Tracking	Publisher	Notes
bare	1.0	5	latest/stable	canonical∕	base
canonical-livepatch	10.1.2	126	latest/stable	canonical√	-

Upgrading

The following details how to upgrade an Ubuntu Server or Ubuntu cloud image to the next release.

Upgrade paths

Ubuntu supports the ability to upgrade from one LTS to the next LTS in sequential order. For example, a user on Ubuntu 16.04 LTS can upgrade to Ubuntu 18.04 LTS, but cannot jump directly to Ubuntu 20.04 LTS. To do this, the user would need to upgrade twice: once to Ubuntu 18.04 LTS, and then upgrade again to Ubuntu 20.04 LTS.

It is recommended that users run an LTS release as it provides 5 years of standard support and security updates. After the initial standard support, an extended support period is available via an Ubuntu Advantage subscription.

For a complete list of releases and current support status see the Ubuntu Wiki Releases page.

Upgrade checklist

To ensure a successful upgrade, please review the following items:

- Check the release notes for the new release for any known issues or important changes. Release notes for each release are found on the Ubuntu Wiki Releases page.
- Fully update the system. The upgrade process works best when the current system has all the latest updates installed. Users should confirm that these commands complete successfully and that no further updates are available. It is also suggested that users reboot the system after all the updates are applied to verify a user is running the latest kernel. To upgrade run the following commands:

sudo apt update
sudo apt upgrade

- Users should check that there is sufficient free disk space for the upgrade. Upgrading a system will at least download hundreds of new packages. Systems with additional software installed may require a few gigabytes of disk space.
- The upgrade process takes time to complete. Users should have dedicated time to participate in the upgrade process.
- Third-party software repositories and PPAs are disabled during the upgrade. However, any software installed from these repositories is not removed or downgraded. Software installed from these repositories is the single most common cause of upgrade issues.
- Backup any and all data. Upgrades are normally safe, however, there is always the chance that something may go wrong. It is extremely important that the data is safely copied to a backup location to allow restoration if there are any problems or complications during the upgrade process.

Upgrade

It is recommended to upgrade the system using the do-release-upgrade command on server edition and cloud images. This command can handle system configuration changes that are sometimes needed between releases.

do-release-upgrade

To begin this process run the following command:

```
sudo do-release-upgrade
```

Upgrading to a development release of Ubuntu is available using the -d flag. However, using the development release or this flag is not recommended for production environments.

Upgrades from one LTS to the next LTS release are only available after the first point release. For example, Ubuntu 18.04 LTS will only upgrade to Ubuntu 20.04 LTS after the 20.04.1 point release. If users wish to update before the point release (e.g. on a subset of machines to evaluate the LTS upgrade) users can force the upgrade via the -d flag.

Pre-upgrade summary

Before making any changes the command will first do some checks to verify the system is ready to update. The user will get prompted with a summary of the upgrade before proceeding. If the user accepts the changes, the process will begin to update the system's packages:

Do you want to start the upgrade?

```
5 installed packages are no longer supported by Canonical. You can still get support from the community.
```

4 packages are going to be removed. 117 new packages are going to be installed. 424 packages are going to be upgraded.

```
You have to download a total of 262 M. This download will take about 33 minutes with a 1Mbit DSL connection and about 10 hours with a 56k modem.
```

Fetching and installing the upgrade can take several hours. Once the download has finished, the process cannot be canceled.

Continue [yN] Details [d]

Configuration changes

It is possible during the upgrade process the user gets presented with a message to make decisions about package updates. These prompts occur when there are existing configuration files edited by the user and the new package configuration file are different. Below is an example prompt:

```
Configuration file '/etc/ssh/ssh_config'
==> Modified (by you or by a script) since installation.
==> Package distributor has shipped an updated version.
What would you like to do about it ? Your options are:
Y or I : install the package maintainer's version
N or 0 : keep your currently-installed version
D : show the differences between the versions
Z : start a shell to examine the situation
The default action is to keep your current version.
```

*** ssh_config (Y/I/N/0/D/Z) [default=N] ?

Users should look at the differences between the files and decide what to do. The default response is to keep the current version of the file. There are situations where accepting the new version, like with /boot/grub/menu.lst, is required for the system to boot correctly with the new kernel.

Package removal

After all packages are updated the user will again remove any obsolete, no longer needed, packages:

30 packages are going to be removed.

Continue [yN] Details [d]

Reboot

Finally, when the upgrade is complete the user is prompted to reboot the system. The system is not considered upgraded until a reboot occurs:

System upgrade is complete.

Restart required

To finish the upgrade, a restart is required. If you select 'y' the system will be restarted.

Continue [yN]

Reporting Bugs in Ubuntu Server

The Ubuntu Project, and thus Ubuntu Server, uses Launchpad as its bug tracker. In order to file a bug, you will need a Launchpad account. Create one here if necessary.

Reporting Bugs With apport-cli

The preferred way to report a bug is with the apport-cli command. It must be invoked on the machine affected by the bug because it collects information from the system on which it is being run and publishes it to the bug report on Launchpad. Getting that information to Launchpad can, therefore, be a challenge if the system is not running a desktop environment in order to use a browser (common with servers) or if it does not have Internet access. The steps to take in these situations are described below.

Note

The commands apport-cli and ubuntu-bug should give the same results on a CLI server. The latter is actually a symlink to apport-bug which is intelligent enough to know whether a desktop environment is in use and will choose apport-cli if not. Since server systems tend to be CLI-only apport-cli was chosen from the outset in this guide.

Bug reports in Ubuntu need to be filed against a specific software package, so the name of the package (source package or program name/path) affected by the bug needs to be supplied to apport-cli:

apport-cli PACKAGENAME

Once apport-cli has finished gathering information you will be asked what to do with it. For instance, to report a bug in vim:

\$ apport-cli vim

*** Collecting problem information

The collected information can be sent to the developers to improve the application. This might take a few minutes.

*** Send problem report to the developers?

After the problem report has been sent, please fill out the form in the automatically opened web browser.

What would you like to do? Your options are:

```
S: Send report (2.8 KB)
```

V: View report

K: Keep report file for sending later or copying to somewhere else

I: Cancel and ignore future crashes of this program version

C: Cancel

Please choose (S/V/K/I/C):

The first three options are described below:

• Send: submits the collected information to Launchpad as part of the process of filing a new bug report. You will be given the opportunity to describe the bug in your own words.

*** Uploading problem information

```
The collected information is being sent to the bug tracking system.
This might take a few minutes.
94%
```

*** To continue, you must visit the following URL:

https://bugs.launchpad.net/ubuntu/+source/vim/+filebug/09b2495a-e2ab-11e3-879b-68b5996a96c8?

You can launch a browser now, or copy this URL into a browser on another computer.

Choices: 1: Launch a browser now C: Cancel Please choose (1/C): 1

The browser that will be used when choosing '1' will be the one known on the system as www-browser via the Debian alternatives system. Examples of text-based browsers to install include links, elinks, lynx, and w3m. You can also manually point an existing browser at the given URL.

- View: displays the collected information on the screen for review. This can be a lot of information. Press 'Enter' to scroll by a screenful. Press 'q' to quit and return to the choice menu.
- Keep: writes the collected information to disk. The resulting file can be later used to file the bug report, typically after transferring it to another Ubuntu system.

What would you like to do? Your options are:

```
S: Send report (2.8 KB)
V: View report
K: Keep report file for sending later or copying to somewhere else
I: Cancel and ignore future crashes of this program version
C: Cancel
Please choose (S/V/K/I/C): k
Problem report file: /tmp/apport.vim.1pg92p02.apport
```

To report the bug, get the file onto an Internet-enabled Ubuntu system and apply apport-cli to it. This will cause the menu to appear immediately (the information is already collected). You should then press 's' to send:

apport-cli apport.vim.1pg92p02.apport

To directly save a report to disk (without menus) you can do:

apport-cli vim --save apport.vim.test.apport

Report names should end in .apport.

Note

If this Internet-enabled system is non-Ubuntu/Debian, apport-cli is not available so the bug will need to be created manually. An apport report is also not to be included as an attachment to a bug either so it is completely useless in this scenario.

Reporting Application Crashes

The software package that provides the apport-cli utility, apport, can be configured to automatically capture the state of a crashed application. This is enabled by default (in /etc/default/apport).

After an application crashes, if enabled, apport will store a crash report under /var/crash:

-rw-r---- 1 peter whoopsie 150K Jul 24 16:17 _usr_lib_x86_64-linux-gnu_libmenu-cache2_libexec_menucached.1000.crash

Use the apport-cli command without arguments to process any pending crash reports. It will offer to report them one by one.

apport-cli
*** Send problem report to the developers?
After the problem report has been sent, please fill out the form in the
automatically opened web browser.
What would you like to do? Your options are:
 S: Send report (153.0 KB)
 V: View report
 K: Keep report file for sending later or copying to somewhere else
 I: Cancel and ignore future crashes of this program version
 C: Cancel

Please choose (S/V/K/I/C): s

If you send the report, as was done above, the prompt will be returned immediately and the /var/crash directory will then contain 2 extra files:

```
-rw-r---- 1 peter whoopsie 150K Jul 24 16:17 _usr_lib_x86_64-linux-gnu_libmenu-cache2_libexec_menu-
cached.1000.crash
-rw-rw-r-- 1 peter whoopsie 0 Jul 24 16:37 _usr_lib_x86_64-linux-gnu_libmenu-cache2_libexec_menu-
cached.1000.upload
-rw------ 1 whoopsie whoopsie 0 Jul 24 16:37 _usr_lib_x86_64-linux-gnu_libmenu-cache2_libexec_menu-
cached.1000.uploaded
```

Sending in a crash report like this will not immediately result in the creation of a new public bug. The report will be made private on Launchpad, meaning that it will be visible to only a limited set of bug triagers. These triagers will then scan the report for possible private data before creating a public bug.

Resources

- See the Reporting Bugs Ubuntu wiki page.
- Also, the Apport page has some useful information. Though some of it pertains to using a GUI.

Kernel Crash Dump

Introduction

A Kernel Crash Dump refers to a portion of the contents of volatile memory (RAM) that is copied to disk whenever the execution of the kernel is disrupted. The following events can cause a kernel disruption :

- Kernel Panic
- Non Maskable Interrupts (NMI)
- Machine Check Exceptions (MCE)
- Hardware failure
- Manual intervention

For some of those events (panic, NMI) the kernel will react automatically and trigger the crash dump mechanism through *kexec*. In other situations a manual intervention is required in order to capture the memory. Whenever one of the above events occurs, it is important to find out the root cause in order to prevent it from happening again. The cause can be determined by inspecting the copied memory contents.

Kernel Crash Dump Mechanism

When a kernel panic occurs, the kernel relies on the *kexec* mechanism to quickly reboot a new instance of the kernel in a pre-reserved section of memory that had been allocated when the system booted (see below). This permits the existing memory area to remain untouched in order to safely copy its contents to storage.

Installation

The kernel crash dump utility is installed with the following command:

sudo apt install linux-crashdump

Note

Starting with 16.04, the kernel crash dump mechanism is enabled by default. During the installation, you will be prompted with the following dialogs.

```
|-----| Configuring kexec-tools |-----|
|
|
| |
| If you choose this option, a system reboot will trigger a restart into a
| kernel loaded by kexec instead of going through the full system boot
| loader process.
|
| Should kexec-tools handle reboots (sysvinit only)?
|
|
```

Select Yes to hook up kexec-tools for all reboots.

```
|-----| Configuring kdump-tools |-----|
|
| If you choose this option, the kdump-tools mechanism will be enabled. A
| reboot is still required in order to enable the crashkernel kernel
| parameter.
|
| Should kdump-tools be enabled be default?
|
|
```

Yes should be selected here as well, to enable kdump-tools.

If you ever need to manually enable the functionality, you can use the dpkg-reconfigure kexec-tools and dpkg-reconfigure kdump-tools commands and answer Yes to the questions. You can also edit /etc/default/kexec and set parameters directly:

Load a kexec kernel (true/false)
LOAD KEXEC=true

As well, edit /etc/default/kdump-tools to enable kdump by including the following line:

USE_KDUMP=1

If a reboot has not been done since installation of the linux-crashdump package, a reboot will be required in order to activate the crashkernel= boot parameter. Upon reboot, kdump-tools will be enabled and active.

If you enable kdump-tools after a reboot, you will only need to issue the kdump-config load command to activate the kdump mechanism.

You can view the current status of kdump via the command kdump-config show. This will display something like this:

DUMP_MODE: kdump USE_KDUMP: 1 KDUMP_SYSCTL: kernel.panic_on_oops=1 KDUMP_COREDIR: /var/crash crashkernel addr: /var/lib/kdump/vmlinuz kdump initrd: /var/lib/kdump/initrd.img current state: ready to kdump kexec command: /sbin/kexec -p --command-line="..." --initrd=...

This tells us that we will find core dumps in /var/crash.

Configuration

In addition to local dump, it is now possible to use the remote dump functionality to send the kernel crash dump to a remote server, using either the SSH or NFS protocols.

Local Kernel Crash Dumps

Local dumps are configured automatically and will remain in use unless a remote protocol is chosen. Many configuration options exist and are thoroughly documented in the /etc/default/kdump-tools file.

Remote Kernel Crash Dumps using the SSH protocol

To enable remote dumps using the SSH protocol, the /etc/default/kdump-tools must be modified in the following manner :

------# Remote dump facilities: # SSH - username and hostname of the remote server that will receive the dump and dmesg files. # # SSH KEY - Full path of the ssh private key to be used to login to the remote server. use kdump-config propagate to send the public key to the # remote server # # HOSTTAG - Select if hostname of IP address will be used as a prefix to the timestamped directory when sending files to the remote server. # 'ip' is the default. # SSH="ubuntu@kdump-netcrash"

The only mandatory variable to define is SSH. It must contain the username and hostname of the remote server using the format {username}@{remote server}.

SSH_KEY may be used to provide an existing private key to be used. Otherwise, the kdump-config propagate command will create a new keypair. The HOSTTAG variable may be used to use the hostname of the system as a prefix to the remote directory to be created instead of the IP address.

The following example shows how kdump-config propagate is used to create and propagate a new keypair to the remote server :

sudo kdump-config propagate
Need to generate a new ssh key...
The authenticity of host 'kdump-netcrash (192.168.1.74)' can't be established.
ECDSA key fingerprint is SHA256:iMp+5Y28qhbd+tevFCWrEXykDd4dI3yN40Vlu3CBBQ4.
Are you sure you want to continue connecting (yes/no)? yes
ubuntu@kdump-netcrash's password:
propagated ssh key /root/.ssh/kdump_id_rsa to server ubuntu@kdump-netcrash

The password of the account used on the remote server will be required in order to successfully send the public key to the server

The kdump-config show command can be used to confirm that kdump is correctly configured to use the SSH protocol :

```
kdump-config show
DUMP_MODE:
                  kdump
USE KDUMP:
                  1
KDUMP SYSCTL:
                  kernel.panic_on_oops=1
KDUMP COREDIR:
                  /var/crash
crashkernel addr: 0x2c000000
   /var/lib/kdump/vmlinuz: symbolic link to /boot/vmlinuz-4.4.0-10-generic
kdump initrd:
   /var/lib/kdump/initrd.img: symbolic link to /var/lib/kdump/initrd.img-4.4.0-10-generic
SSH:
                  ubuntu@kdump-netcrash
SSH KEY:
                  /root/.ssh/kdump id rsa
```

HOSTTAG: ip current state: ready to kdump

Remote Kernel Crash Dumps using the NFS protocol

To enable remote dumps using the *NFS* protocol, the /etc/default/kdump-tools must be modified in the following manner :

```
# NFS - Hostname and mount point of the NFS server configured to receive
# the crash dump. The syntax must be {HOSTNAME}:{MOUNTPOINT}
# (e.g. remote:/var/crash)
```

#

NFS="kdump-netcrash:/var/crash"

As with the SSH protocol, the HOSTTAG variable can be used to replace the IP address by the hostname as the prefix of the remote directory.

The kdump-config show command can be used to confirm that kdump is correctly configured to use the NFS protocol :

```
kdump-config show
DUMP MODE:
                  kdump
USE_KDUMP:
                  1
KDUMP_SYSCTL:
                  kernel.panic_on_oops=1
KDUMP_COREDIR:
                  /var/crash
crashkernel addr: 0x2c000000
   /var/lib/kdump/vmlinuz: symbolic link to /boot/vmlinuz-4.4.0-10-generic
kdump initrd:
   /var/lib/kdump/initrd.img: symbolic link to /var/lib/kdump/initrd.img-4.4.0-10-generic
NES:
                  kdump-netcrash:/var/crash
HOSTTAG:
                  hostname
current state:
                  ready to kdump
```

Verification

To confirm that the kernel dump mechanism is enabled, there are a few things to verify. First, confirm that the *crashkernel* boot parameter is present (note: The following line has been split into two to fit the format of this document:

cat /proc/cmdline

```
BOOT_IMAGE=/vmlinuz-3.2.0-17-server root=/dev/mapper/PreciseS-root ro crashkernel=384M-2G:64M,2G-:128M
```

The *crashkernel* parameter has the following syntax:

crashkernel=<rangel>:<sizel>[,<range2>:<size2>,...][@offset]
 range=start-[end] 'start' is inclusive and 'end' is exclusive.

So for the crashkernel parameter found in /proc/cmdline we would have :

crashkernel=384M-2G:64M,2G-:128M

The above value means:

- if the RAM is smaller than 384M, then don't reserve anything (this is the "rescue" case)
- if the RAM size is between 386M and 2G (exclusive), then reserve 64M
- if the RAM size is larger than 2G, then reserve 128M

Second, verify that the kernel has reserved the requested memory area for the kdump kernel by doing: $dmesg \mid grep -i \ crash$

... [

0.000000] Reserving 64MB of memory at 800MB for crashkernel (System RAM: 1023MB)

Finally, as seen previously, the ${\tt kdump-config}$ show command displays the current status of the kdump-tools configuration :

kexec command:

/sbin/kexec -p --command-line="BOOT_IMAGE=/vmlinuz-4.4.0-10-generic root=/dev/mapper/VividS--vgroot ro debug break=init console=ttyS0,115200 irqpoll maxcpus=1 nousb systemd.unit=kdump-tools.service" -initrd=/var/lib/kdump/initrd.img /var/lib/kdump/vmlinuz

Testing the Crash Dump Mechanism

Warning

Testing the Crash Dump Mechanism will cause *a system reboot*. In certain situations, this can cause data loss if the system is under heavy load. If you want to test the mechanism, make sure that the system is idle or under very light load.

Verify that the SysRQ mechanism is enabled by looking at the value of the /proc/sys/kernel/sysrq kernel parameter :

cat /proc/sys/kernel/sysrq

If a value of θ is returned the dump and then reboot feature is disabled. A value greater than 1 indicates that a sub-set of sysrq features is enabled. See /etc/sysctl.d/10-magic-sysrq.conf for a detailed description of the options and the default value. Enable dump then reboot testing with the following command :

sudo sysctl -w kernel.sysrq=1

Once this is done, you must become root, as just using sudo will not be sufficient. As the *root* user, you will have to issue the command echo c > /proc/sysrq-trigger. If you are using a network connection, you will lose contact with the system. This is why it is better to do the test while being connected to the system console. This has the advantage of making the kernel dump process visible.

A typical test output should look like the following :

```
sudo -s
[sudo] password for ubuntu:
# echo c > /proc/sysrq-trigger
[
    31.659002] SysRq : Trigger a crash
   31.659749] BUG: unable to handle kernel NULL pointer dereference at
                                                                                    (null)
[
   31.662668] IP: [<ffffff8139f166>] sysrq_handle_crash+0x16/0x20
Γ
   31.662668] PGD 3bfb9067 PUD 368a7067 PMD 0
[
    31.662668] Oops: 0002 [#1] SMP
ſ
[
    31.662668] CPU 1
. . . .
```

The rest of the output is truncated, but you should see the system rebooting and somewhere in the log, you will see the following line :

Begin: Saving vmcore from kernel crash ...

Once completed, the system will reboot to its normal operational mode. You will then find the Kernel Crash Dump file, and related subdirectories, in the /var/crash directory :

ls /var/crash

201809240744 kexec_cmd linux-image-4.15.0-34-generic-201809240744.crash

If the dump does not work due to OOM (Out Of Memory) error, then try increasing the amount of reserved memory by editing /etc/default/grub.d/kdump-tools.cfg. For example, to reserve 512 megabytes :

GRUB_CMDLINE_LINUX_DEFAULT="\$GRUB_CMDLINE_LINUX_DEFAULT crashkernel=384M-:512M"

 run sudo $\operatorname{update-grub}$ and then reboot afterwards, and then test again.

Resources

Kernel Crash Dump is a vast topic that requires good knowledge of the linux kernel. You can find more information on the topic here :

- Kdump kernel documentation.
- Analyzing Linux Kernel Crash (Based on Fedora, it still gives a good walkthrough of kernel dump analysis)

Networking

Networks consist of two or more devices, such as computer systems, printers, and related equipment which are connected by either physical cabling or wireless links for the purpose of sharing and distributing information among the connected devices.

This section provides general and specific information pertaining to networking, including an overview of network concepts and detailed discussion of popular network protocols.

TCP/IP

The Transmission Control Protocol and Internet Protocol (TCP/IP) is a standard set of protocols developed in the late 1970s by the Defense Advanced Research Projects Agency (DARPA) as a means of communication between different types of computers and computer networks. TCP/IP is the driving force of the Internet, and thus it is the most popular set of network protocols on Earth.

TCP/IP Introduction

The two protocol components of TCP/IP deal with different aspects of computer networking. *Internet Protocol*, the "IP" of TCP/IP is a connectionless protocol which deals only with network packet routing using the *IP Datagram* as the basic unit of networking information. The IP Datagram consists of a header followed by a message. The *Transmission Control Protocol* is the "TCP" of TCP/IP and enables network hosts to establish connections which may be used to exchange data streams. TCP also guarantees that the data between connections is delivered and that it arrives at one network host in the same order as sent from another network host.

TCP/IP Configuration

The TCP/IP protocol configuration consists of several elements which must be set by editing the appropriate configuration files, or deploying solutions such as the Dynamic Host Configuration Protocol (DHCP) server which in turn, can be configured to provide the proper TCP/IP configuration settings to network clients automatically. These configuration values must be set correctly in order to facilitate the proper network operation of your Ubuntu system.

The common configuration elements of TCP/IP and their purposes are as follows:

- IP address The IP address is a unique identifying string expressed as four decimal numbers ranging from zero (0) to two-hundred and fifty-five (255), separated by periods, with each of the four numbers representing eight (8) bits of the address for a total length of thirty-two (32) bits for the whole address. This format is called *dotted quad notation*.
- Netmask The Subnet Mask (or simply, *netmask*) is a local bit mask, or set of flags which separate the portions of an IP address significant to the network from the bits significant to the *subnetwork*. For example, in a Class C network, the standard netmask is 255.255.255.0 which masks the first three bytes of the IP address and allows the last byte of the IP address to remain available for specifying hosts on the subnetwork.
- Network Address The Network Address represents the bytes comprising the network portion of an IP address. For example, the host 12.128.1.2 in a Class A network would use 12.0.0.0 as the network address, where twelve (12) represents the first byte of the IP address, (the network part) and zeroes (0) in all of the remaining three bytes to represent the potential host values. A network host using the private IP address 192.168.1.100 would in turn use a Network Address of 192.168.1.0, which specifies the first three bytes of the Class C 192.168.1 network and a zero (0) for all the possible hosts on the network.
- Broadcast Address The Broadcast Address is an IP address which allows network data to be sent simultaneously to all hosts on a given subnetwork rather than specifying a particular host. The standard general broadcast address for IP networks is 255.255.255.255.255, but this broadcast address cannot be used to send a broadcast message to every host on the Internet because routers block it. A more appropriate broadcast address is set to match

a specific subnetwork. For example, on the private Class C IP network, 192.168.1.0, the broadcast address is 192.168.1.255. Broadcast messages are typically produced by network protocols such as the Address Resolution Protocol (ARP) and the Routing Information Protocol (RIP).

- Gateway Address A Gateway Address is the IP address through which a particular network, or host on a network, may be reached. If one network host wishes to communicate with another network host, and that host is not located on the same network, then a *gateway* must be used. In many cases, the Gateway Address will be that of a router on the same network, which will in turn pass traffic on to other networks or hosts, such as Internet hosts. The value of the Gateway Address setting must be correct, or your system will not be able to reach any hosts beyond those on the same network.
- Nameserver Address Nameserver Addresses represent the IP addresses of Domain Name Service (DNS) systems, which resolve network hostnames into IP addresses. There are three levels of Nameserver Addresses, which may be specified in order of precedence: The *Primary* Nameserver, the *Secondary* Nameserver, and the *Tertiary* Nameserver. In order for your system to be able to resolve network hostnames into their corresponding IP addresses, you must specify valid Nameserver Addresses which you are authorized to use in your system's TCP/IP configuration. In many cases these addresses can and will be provided by your network service provider, but many free and publicly accessible nameservers are available for use, such as the Level3 (Verizon) servers with IP addresses from 4.2.2.1 to 4.2.2.6.

Tip

The IP address, Netmask, Network Address, Broadcast Address, Gateway Address, and Nameserver Addresses are typically specified via the appropriate directives in the file /etc/network/interfaces. For more information, view the system manual page for interfaces, with the following command typed at a terminal prompt:

Access the system manual page for interfaces with the following command:

man interfaces

IP Routing

IP routing is a means of specifying and discovering paths in a TCP/IP network along which network data may be sent. Routing uses a set of *routing tables* to direct the forwarding of network data packets from their source to the destination, often via many intermediary network nodes known as *routers*. There are two primary forms of IP routing: *Static Routing* and *Dynamic Routing*.

Static routing involves manually adding IP routes to the system's routing table, and this is usually done by manipulating the routing table with the route command. Static routing enjoys many advantages over dynamic routing, such as simplicity of implementation on smaller networks, predictability (the routing table is always computed in advance, and thus the route is precisely the same each time it is used), and low overhead on other routers and network links due to the lack of a dynamic routing protocol. However, static routing does present some disadvantages as well. For example, static routing is limited to small networks and does not scale well. Static routing also fails completely to adapt to network outages and failures along the route due to the fixed nature of the route.

Dynamic routing depends on large networks with multiple possible IP routes from a source to a destination and makes use of special routing protocols, such as the Router Information Protocol (RIP), which handle the automatic adjustments in routing tables that make dynamic routing possible. Dynamic routing has several advantages over static routing, such as superior scalability and the ability to adapt to failures and outages along network routes. Additionally, there is less manual configuration of the routing tables, since routers learn from one another about their existence and available routes. This trait also eliminates the possibility of introducing mistakes in the routing tables via human error. Dynamic routing is not perfect, however, and presents disadvantages such as heightened complexity and additional network overhead from router communications, which does not immediately benefit the end users, but still consumes network bandwidth.

TCP and UDP

TCP is a connection-based protocol, offering error correction and guaranteed delivery of data via what is known as *flow control*. Flow control determines when the flow of a data stream needs to be stopped, and previously sent data packets should to be re-sent due to problems such as *collisions*, for example, thus ensuring complete and accurate delivery of the data. TCP is typically used in the exchange of important information such as database transactions.

The User Datagram Protocol (UDP), on the other hand, is a *connectionless* protocol which seldom deals with the transmission of important data because it lacks flow control or any other method to ensure reliable delivery of the data. UDP is commonly used in such applications as audio and video streaming, where it is considerably faster than TCP due to the lack of error correction and flow control, and where the loss of a few packets is not generally catastrophic.

ICMP

The Internet Control Messaging Protocol (ICMP) is an extension to the Internet Protocol (IP) as defined in the Request For Comments (RFC) #792 and supports network packets containing control, error, and informational messages. ICMP is used by such network applications as the ping utility, which can determine the availability of a network host or device. Examples of some error messages returned by ICMP which are useful to both network hosts and devices such as routers, include *Destination Unreachable* and *Time Exceeded*.

Daemons

Daemons are special system applications which typically execute continuously in the background and await requests for the functions they provide from other applications. Many daemons are network-centric; that is, a large number of daemons executing in the background on an Ubuntu system may provide network-related functionality. Some examples of such network daemons include the *Hyper Text Transport Protocol Daemon* (httpd), which provides web server functionality; the *Secure SHell Daemon* (sshd), which provides secure remote login shell and file transfer capabilities; and the *Internet Message Access Protocol Daemon* (imapd), which provides E-Mail services.

Resources

- There are man pages for TCP and IP that contain more useful information.
- Also, see the TCP/IP Tutorial and Technical Overview IBM Redbook.
- Another resource is O'Reilly's TCP/IP Network Administration.

Network Configuration

Ubuntu ships with a number of graphical utilities to configure your network devices. This document is geared toward server administrators and will focus on managing your network on the command line.

Ethernet Interfaces

Ethernet interfaces are identified by the system using predictable network interface names. These names can appear as eno1 or enp0s25. However, in some cases an interface may still use the kernel eth# style of naming.

Identify Ethernet Interfaces

To quickly identify all available Ethernet interfaces, you can use the ip command as shown below.

```
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00 brd 00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
       link/ether 00:16:3e:e2:52:42 brd ff:ff:ff:ff:ff link-netnsid 0
       inet 10.102.66.200/24 brd 10.102.66.255 scope global dynamic eth0
       valid_lft 3257sec preferred_lft 3257sec
    inet6 fe80::216:3eff:fee2:5242/64 scope link
       valid_lft forever preferred_lft forever
```

Another application that can help identify all network interfaces available to your system is the lshw command. This command provides greater details around the hardware capabilities of specific adapters. In the example below, lshw shows a single Ethernet interface with the logical name of eth0 along with bus information, driver details and all supported capabilities.

```
sudo lshw -class network
 *-network
    description: Ethernet interface
    product: MT26448 [ConnectX EN 10GigE, PCIe 2.0 5GT/s]
    vendor: Mellanox Technologies
    physical id: 0
    bus info: pci@0004:01:00.0
    logical name: eth4
```

```
version: b0
serial: e4:1d:2d:67:83:56
slot: U78CB.001.WZS09KB-P1-C6-T1
size: 10Gbit/s
capacity: 10Gbit/s
width: 64 bits
clock: 33MHz
capabilities: pm vpd msix pciexpress bus_master cap_list ethernet physical fibre 10000bt-fd
configuration: autonegotiation=off broadcast=yes driver=mlx4_en driverversion=4.0-0 duplex=full firmware=2.9.1326 i
    resources: iomemory:24000-23fff irg:481 memory:3fe200000000-3fe2000fffff memory:24000000000-
```

240007fffff

Ethernet Interface Logical Names

Interface logical names can also be configured via a netplan configuration. If you would like control which interface receives a particular logical name use the *match* and *set-name* keys. The match key is used to find an adapter based on some criteria like MAC address, driver, etc. Then the set-name key can be used to change the device to the desired logial name.

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth_lan0:
    dhcp4: true
    match:
    macaddress: 00:11:22:33:44:55
    set-name: eth_lan0
```

Ethernet Interface Settings

ethtool is a program that displays and changes Ethernet card settings such as auto-negotiation, port speed, duplex mode, and Wake-on-LAN. The following is an example of how to view supported features and configured settings of an Ethernet interface.

```
sudo ethtool eth4
Settings for eth4:
    Supported ports: [ FIBRE ]
    Supported link modes:
                            10000baseT/Full
    Supported pause frame use: No
    Supports auto-negotiation: No
    Supported FEC modes: Not reported
    Advertised link modes: 10000baseT/Full
    Advertised pause frame use: No
    Advertised auto-negotiation: No
    Advertised FEC modes: Not reported
    Speed: 10000Mb/s
    Duplex: Full
    Port: FIBRE
    PHYAD: 0
    Transceiver: internal
    Auto-negotiation: off
    Supports Wake-on: d
    Wake-on: d
    Current message level: 0x00000014 (20)
                   link ifdown
    Link detected: yes
```

IP Addressing

The following section describes the process of configuring your systems IP address and default gateway needed for communicating on a local area network and the Internet.

Temporary IP Address Assignment

For temporary network configurations, you can use the ip command which is also found on most other GNU/Linux operating systems. The ip command allows you to configure settings which take effect immediately, however they are not persistent and will be lost after a reboot.

To temporarily configure an IP address, you can use the ip command in the following manner. Modify the IP address and subnet mask to match your network requirements.

sudo ip addr add 10.102.66.200/24 dev enp0s25

The ip can then be used to set the link up or down.

ip link set dev enp0s25 up

ip link set dev enp0s25 down

To verify the IP address configuration of enp0s25, you can use the ip command in the following manner.

```
ip address show dev enp0s25
```

```
10: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
link/ether 00:16:3e:e2:52:42 brd ff:ff:ff:ff:ff link-netnsid 0
inet 10.102.66.200/24 brd 10.102.66.255 scope global dynamic eth0
valid_lft 2857sec preferred_lft 2857sec
inet6 fe80::216:3eff:fee2:5242/64 scope link
valid lft forever preferred lft forever6
```

To configure a default gateway, you can use the ip command in the following manner. Modify the default gateway address to match your network requirements.

```
sudo ip route add default via 10.102.66.1
```

To verify your default gateway configuration, you can use the ip command in the following manner.

```
ip route show
default via 10.102.66.1 dev eth0 proto dhcp src 10.102.66.200 metric 100
10.102.66.0/24 dev eth0 proto kernel scope link src 10.102.66.200
10.102.66.1 dev eth0 proto dhcp scope link src 10.102.66.200 metric 100
```

If you require DNS for your temporary network configuration, you can add DNS server IP addresses in the file /etc/resolv.conf. In general, editing /etc/resolv.conf directly is not recommanded, but this is a temporary and non-persistent configuration. The example below shows how to enter two DNS servers to /etc/resolv.conf, which should be changed to servers appropriate for your network. A more lengthy description of the proper persistent way to do DNS client configuration is in a following section.

nameserver 8.8.8.8
nameserver 8.8.4.4

If you no longer need this configuration and wish to purge all IP configuration from an interface, you can use the ip command with the flush option as shown below.

```
ip addr flush eth0
```

Note

Flushing the IP configuration using the ip command does not clear the contents of /etc/resolv.conf. You must remove or modify those entries manually, or re-boot which should also cause /etc/resolv.conf, which is a symlink to /run/systemd/resolve/stub-resolv.conf, to be re-written.

Dynamic IP Address Assignment (DHCP Client)

To configure your server to use DHCP for dynamic address assignment, create a netplan configuration in the file /etc/netplan/99_config.yaml. The example below assumes you are configuring your first Ethernet interface identified as *enp3s0*.

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp3s0:
        dhcp4: true
```

The configuration can then be applied using the netplan command.

Static IP Address Assignment

To configure your system to use static address assignment, create a netplan configuration in the file /etc/netplan/99_config.yaml. The example below assumes you are configuring your first Ethernet interface identified as *eth0*. Change the *addresses*, *gateway4*, and *nameservers* values to meet the requirements of your network.

```
network:
```

```
version: 2
renderer: networkd
ethernets:
   eth0:
    addresses:
        - 10.10.10.2/24
   gateway4: 10.10.10.1
   nameservers:
        search: [mydomain, otherdomain]
        addresses: [10.10.10.1, 1.1.1]
```

The configuration can then be applied using the netplan command.

```
sudo netplan apply
```

Loopback Interface

The loopback interface is identified by the system as *lo* and has a default IP address of 127.0.0.1. It can be viewed using the ip command.

```
ip address show lo
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
```

Name Resolution

Name resolution as it relates to IP networking is the process of mapping IP addresses to hostnames, making it easier to identify resources on a network. The following section will explain how to properly configure your system for name resolution using DNS and static hostname records.

DNS Client Configuration

Traditionally, the file /etc/resolv.conf was a static configuration file that rarely needed to be changed or automatically changed via DCHP client hooks. Systemd-resolved handles name server configuration, and it should be interacted with through the systemd-resolve command. Netplan configures systemd-resolved to generate a list of nameservers and domains to put in /etc/resolv.conf, which is a symlink:

/etc/resolv.conf -> ../run/systemd/resolve/stub-resolv.conf

To configure the resolver, add the IP addresses of the nameservers that are appropriate for your network to the netplan configuration file. You can also add an optional DNS suffix search-lists to match your network domain names. The resulting file might look like the following:

```
network:
version: 2
renderer: networkd
ethernets:
enp0s25:
addresses:
- 192.168.0.100/24
gateway4: 192.168.0.1
nameservers:
search: [mydomain, otherdomain]
addresses: [1.1.1.1, 8.8.8.8, 4.4.4.4]
```

The *search* option can also be used with multiple domain names so that DNS queries will be appended in the order in which they are entered. For example, your network may have multiple sub-domains to search; a parent domain of example.com, and two sub-domains, sales.example.com and dev.example.com.

If you have multiple domains you wish to search, your configuration might look like the following:

```
network:
version: 2
renderer: networkd
ethernets:
    enp0s25:
    addresses:
        - 192.168.0.100/24
    gateway4: 192.168.0.1
    nameservers:
        search: [example.com, sales.example.com, dev.example.com]
        addresses: [1.1.1.1, 8.8.8.8, 4.4.4.4]
```

If you try to ping a host with the name of *server1*, your system will automatically query DNS for its Fully Qualified Domain Name (FQDN) in the following order:

- 1. server1.example.com
- $2. \ \texttt{server1.sales.example.com}$
- 3. server1.dev.example.com

If no matches are found, the DNS server will provide a result of *notfound* and the DNS query will fail.

Static Hostnames

Static hostnames are locally defined hostname-to-IP mappings located in the file /etc/hosts. Entries in the hosts file will have precedence over DNS by default. This means that if your system tries to resolve a hostname and it matches an entry in /etc/hosts, it will not attempt to look up the record in DNS. In some configurations, especially when Internet access is not required, servers that communicate with a limited number of resources can be conveniently set to use static hostnames instead of DNS.

The following is an example of a **hosts** file where a number of local servers have been identified by simple hostnames, aliases and their equivalent Fully Qualified Domain Names (FQDN's).

```
127.0.0.1 localhost
127.0.1.1 ubuntu-server
10.0.0.11 server1 server1.example.com vpn
10.0.0.12 server2 server2.example.com mail
10.0.0.13 server3 server3.example.com www
10.0.0.14 server4 server4.example.com file
```

Note

In the above example, notice that each of the servers have been given aliases in addition to their proper names and FQDN's. *Server1* has been mapped to the name *vpn*, *server2* is referred to as *mail*, *server3* as *www*, and *server4* as *file*.

Name Service Switch Configuration

The order in which your system selects a method of resolving hostnames to IP addresses is controlled by the Name Service Switch (NSS) configuration file /etc/nsswitch.conf. As mentioned in the previous section, typically static hostnames defined in the systems /etc/hosts file have precedence over names resolved from DNS. The following is an example of the line responsible for this order of hostname lookups in the file /etc/nsswitch.conf.

hosts: files mdns4_minimal [NOTFOUND=return] dns mdns4

- files first tries to resolve static hostnames located in /etc/hosts.
- mdns4_minimal attempts to resolve the name using Multicast DNS.
- **[NOTFOUND=return]** means that any response of *notfound* by the preceding *mdns4_minimal* process should be treated as authoritative and that the system should not try to continue hunting for an answer.
- dns represents a legacy unicast DNS query.
- mdns4 represents a Multicast DNS query.
To modify the order of the above mentioned name resolution methods, you can simply change the *hosts:* string to the value of your choosing. For example, if you prefer to use legacy Unicast DNS versus Multicast DNS, you can change the string in /etc/nsswitch.conf as shown below.

hosts: files dns [NOTFOUND=return] mdns4_minimal mdns4

Bridging

Bridging multiple interfaces is a more advanced configuration, but is very useful in multiple scenarios. One scenario is setting up a bridge with multiple network interfaces, then using a firewall to filter traffic between two network segments. Another scenario is using bridge on a system with one interface to allow virtual machines direct access to the outside network. The following example covers the latter scenario.

Configure the bridge by editing your netplan configuration found in /etc/netplan/:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp3s0:
    dhcp4: no
  bridges:
    br0:
    dhcp4: yes
    interfaces:
        - enp3s0
    Note
```

Enter the appropriate values for your physical interface and network.

Now apply the configuration to enable the bridge:

sudo netplan apply

The new bridge interface should now be up and running. The brctl provides useful information about the state of the bridge, controls which interfaces are part of the bridge, etc. See man brctl for more information.

networkd-dispatcher for pre-up, post-up, etc. hook scripts

Users of the former ifupdown may be familiar with using hook scripts (e.g pre-up, post-up, etc.) in their interfaces file. Netplan configuration does not currently support hook scripts in its configuration definition.

Instead to achieve this functionality with the networkd renderer, users can use networkd-dispatcher. The package provides users and packages hook points when specific network states are reached to aid in reacting to network state.

Note: If not on Ubuntu Server, but Desktop the network is driven by Network Manager - in that case you'd need NM Dispatcher scripts instead.

The Netplan FAQ has a great table that compares event timings between ifupdown/systemd-networkd/network-manager

It is important to be aware that those hooks run asychronous; that is they will not block transition into another state.

The Netplan FAQ also has an example on converting an old ifupdown hook to networkd-dispatcher.

Resources

- The Ubuntu Wiki Network page has links to articles covering more advanced network configuration.
- The netplan website has additional examples and documentation.
- The netplan man page has more information on netplan.
- The systemd-resolved man page has more information on systemd-resolved service.
- For more information on *bridging* see the netplan.io examples page and the Linux Foundation's Networking-Bridge page.

Dynamic Host Configuration Protocol (DHCP)

The Dynamic Host Configuration Protocol (DHCP) is a network service that enables host computers to be automatically assigned settings from a server as opposed to manually configuring each network host. Computers configured to be DHCP clients have no control over the settings they receive from the DHCP server, and the configuration is transparent to the computer's user.

The most common settings provided by a DHCP server to DHCP clients include:

- IP address and netmask
- IP address of the default-gateway to use
- IP adresses of the DNS servers to use

However, a DHCP server can also supply configuration properties such as:

- Host Name
- Domain Name
- Time Server
- Print Server

The advantage of using DHCP is that changes to the network, for example a change in the address of the DNS server, need only be changed at the DHCP server, and all network hosts will be reconfigured the next time their DHCP clients poll the DHCP server. As an added advantage, it is also easier to integrate new computers into the network, as there is no need to check for the availability of an IP address. Conflicts in IP address allocation are also reduced.

A DHCP server can provide configuration settings using the following methods:

• Manual allocation (MAC address)

This method entails using DHCP to identify the unique hardware address of each network card connected to the network and then continually supplying a constant configuration each time the DHCP client makes a request to the DHCP server using that network device. This ensures that a particular address is assigned automatically to that network card, based on it's MAC address.

• Dynamic allocation (address pool)

In this method, the DHCP server will assign an IP address from a pool of addresses (sometimes also called a range or scope) for a period of time or lease, that is configured on the server or until the client informs the server that it doesn't need the address anymore. This way, the clients will be receiving their configuration properties dynamically and on a "first come, first served" basis. When a DHCP client is no longer on the network for a specified period, the configuration is expired and released back to the address pool for use by other DHCP Clients. This way, an address can be leased or used for a period of time. After this period, the client has to renegociate the lease with the server to maintain use of the address.

• Automatic allocation

Using this method, the DHCP automatically assigns an IP address permanently to a device, selecting it from a pool of available addresses. Usually DHCP is used to assign a temporary address to a client, but a DHCP server can allow an infinite lease time.

The last two methods can be considered "automatic" because in each case the DHCP server assigns an address with no extra intervention needed. The only difference between them is in how long the IP address is leased, in other words whether a client's address varies over time. The DHCP server Ubuntu makes available is dhcpd (dynamic host configuration protocol daemon), which is easy to install and configure and will be automatically started at system boot.

Installation

At a terminal prompt, enter the following command to install dhcpd:

sudo apt install isc-dhcp-server

NOTE: dhcpd's messages are being sent to syslog. Look there for diagnostics messages.

Configuration

You will probably need to change the default configuration by editing /etc/dhcp/dhcpd.conf to suit your needs and particular configuration.

Most commonly, what you want to do is assign an IP address randomly. This can be done with settings as follows:

```
# minimal sample /etc/dhcp/dhcpd.conf
default-lease-time 600;
max-lease-time 7200;
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.150 192.168.1.200;
  option routers 192.168.1.254;
  option domain-name-servers 192.168.1.1, 192.168.1.2;
  option domain-name "mydomain.example";
}
```

This will result in the DHCP server giving clients an IP address from the range 192.168.1.150-192.168.1.200. It will lease an IP address for 600 seconds if the client doesn't ask for a specific time frame. Otherwise the maximum (allowed) lease will be 7200 seconds. The server will also "advise" the client to use 192.168.1.254 as the default-gateway and 192.168.1.1 and 192.168.1.2 as its DNS servers.

You also may need to edit /etc/default/isc-dhcp-server to specify the interfaces dhcpd should listen to.

INTERFACESv4="eth4"

After changing the config files you have to restart the dhcpd service:

sudo systemctl restart isc-dhcp-server.service

References

- The isc-dhcp-server Ubuntu Wiki page has more information.
- For more /etc/dhcp/dhcpd.conf options see the dhcpd.conf man page.
- ISC dhcp-server

Time Synchronization

NTP is a TCP/IP protocol for synchronizing time over a network. Basically a client requests the current time from a server, and uses it to set its own clock.

Behind this simple description, there is a lot of complexity - there are tiers of NTP servers, with the tier one NTP servers connected to atomic clocks, and tier two and three servers spreading the load of actually handling requests across the Internet. Also the client software is a lot more complex than you might think - it has to factor out communication delays, and adjust the time in a way that does not upset all the other processes that run on the server. But luckily all that complexity is hidden from you!

Ubuntu by default uses timedatectl / timesyncd to synchronize time and users can optionally use chrony to serve the Network Time Protocol.

Synchronizing your systems time

Since Ubuntu 16.04 timedatectl / timesyncd (which are part of systemd) replace most of ntpdate / ntp.

timesyncd is available by default and replaces not only ntpdate, but also the client portion of chrony (or formerly ntpd). So on top of the one-shot action that ntpdate provided on boot and network activation, now timesyncd by default regularly checks and keeps your local time in sync. It also stores time updates locally, so that after reboots monotonically advances if applicable.

If chrony is installed timedatectl steps back to let chrony do the time keeping. That shall ensure that no two time syncing services are fighting. While no more recommended to be used, this still also applies to ntpd being installed to retain any kind of old behavior/config that you had through an upgrade. But it also implies that on an upgrade from a former release ntp/ntpdate might still be installed and therefore renders the new systemd based services disabled.

ntpdate is considered deprecated in favor of timedatectl (or chrony) and thereby no more installed by default. timesyncd will generally do the right thing keeping your time in sync, and chrony will help with more complex cases. But if you had one of a few known special ntpdate use cases, consider the following:

- If you require a one-shot sync use: ${\sf chronyd}$ -q
- If you require a one-shot time check, without setting the time use: chronyd -Q

Configuring timedatectl and timesyncd

The current status of time and time configuration via timedatectl and timesyncd can be checked with timedatectl status.

```
$ timedatectl status
Local time: Fr 2018-02-23 08:47:13 UTC
Universal time: Fr 2018-02-23 08:47:13 UTC
RTC time: Fr 2018-02-23 08:47:13
Time zone: Etc/UTC (UTC, +0000)
System clock synchronized: yes
systemd-timesyncd.service active: yes
RTC in local TZ: no
```

If chrony is running it will automatically switch to:

[...]

systemd-timesyncd.service active: no

Via timedatectl an admin can control the timezone, how the system clock should relate to the hwclock and if permanent synronization should be enabled or not. See man timedatectl for more details.

timesyncd itself is still a normal service, so you can check its status also more in detail via.

```
$ systemctl status systemd-timesyncd
systemd-timesyncd.service - Network Time Synchronization
Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled; vendor preset: enabled)
Active: active (running) since Fri 2018-02-23 08:55:46 UTC; 10s ago
Docs: man:systemd-timesyncd.service(8)
Main PID: 3744 (systemd-timesyn)
Status: "Synchronized to time server 91.189.89.198:123 (ntp.ubuntu.com)."
Tasks: 2 (limit: 4915)
CGroup: /system.slice/systemd-timesyncd.service
|-3744 /lib/systemd/systemd-timesyncd
```

```
Feb 23 08:55:46 bionic-test systemd[1]: Starting Network Time Synchronization...
Feb 23 08:55:46 bionic-test systemd[1]: Started Network Time Synchronization.
Feb 23 08:55:46 bionic-test systemd-timesyncd[3744]: Synchronized to time server 91.189.89.198:123 (ntp.ubuntu.com).
```

The nameserver to fetch time for timedatectl and timesyncd from can be specified in /etc/systemd/timesyncd.conf and additional config files can be stored in /etc/systemd/timesyncd.conf.d/. The entries for NTP= and FallbackNTP= are space separated lists. See man timesyncd.conf for more.

Serve the Network Time Protocol

If in addition to synchronizing your system you also want to serve NTP information you need an NTP server. There are several options with chrony, ntpd and open-ntp. The recommended solution is chrony.

chrony(d)

The NTP daemon chronyd calculates the drift and offset of your system clock and continuously adjusts it, so there are no large corrections that could lead to inconsistent logs for instance. The cost is a little processing power and memory, but for a modern server this is usually negligible.

Installation

To install chrony, from a terminal prompt enter:

sudo apt install chrony

This will provide two binaries:

- chronyd the actual daemon to sync and serve via the NTP protocol
- chronyc command-line interface for chrony daemon

Chronyd Configuration

Edit /etc/chrony/chrony.conf to add/remove server lines. By default these servers are configured:

```
# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
pool 0.ubuntu.pool.ntp.org iburst
pool 1.ubuntu.pool.ntp.org iburst
pool 2.ubuntu.pool.ntp.org iburst
pool 3.ubuntu.pool.ntp.org iburst
```

See man chrony.conf for more details on the configuration options. After changing the any of the config file you have to restart chrony:

sudo systemctl restart chrony.service

Of the pool 2.ubuntu.pool.ntp.org as well as ntp.ubuntu.com also support ipv6 if needed. If one needs to force ipv6 there also is ipv6.ntp.ubuntu.com which is not configured by default.

Serving the NTP Protocol

You can install chrony (above) and configure special Hardware (below) for a local synchronization and as-installed that is the default to stay on the secure and conservative side. But if you want to serve NTP you need adapt your configuration.

To enable serving the NTP protocol you'll need at least to set the allow rule to which controls which clients/networks you want chrony to serve NTP to.

An example would be

allow 1.2.3.4

See the section "NTP server" in the man page for more details on how you can control and restrict access to your NTP server.

View status

Use chronyc to see query the status of the chrony daemon. For example to get an overview of the currently available and selected time sources.

> 29ms 50ms 18ms 19ms 29ms 53ms

chronyc sources

MS	Name/IP address	Stratum	Poll	Reach	LastR	x Last sample 	
^+	gamma.rueckgr.at	2	8	377	135	-1048us[-1048us] +/-	29ms
^_	2b.ncomputers.org	2	8	377	204	-1141us[-1124us] +/-	50ms
^+	www.kashra.com	2	8	377	139	+3483us[+3483us] +/-	18ms
^+	stratum2-4.NTP.TechFak.U	> 2	8	377	143	-2090us[-2073us] +/-	19ms
^_	zepto.mcl.gg	2	7	377	9	-774us[-774us] +/-	29ms
^_	mirrorhost.pw	2	7	377	78	-660us[-660us] +/-	53ms
^_	atto.mcl.gg	2	7	377	8	-823us[-823us] +/-	50ms

^- atto.mcl.gg	2	7	377	8	-823us[-823us] +/-	50ms
^- static.140.107.46.78.cli>	2	8	377	9	-1503us[-1503us] +/-	45ms
^- 4.53.160.75	2	8	377	137	-11ms[-11ms] +/-	117ms
^- 37.44.185.42	3	7	377	10	-3274us[-3274us] +/-	70ms
^- bagnikita.com	2	7	377	74	+3131us[+3131us] +/-	71ms
<pre>^- europa.ellipse.net</pre>	2	8	377	204	-790us[-773us] +/-	97ms
<pre>^- tethys.hot-chilli.net</pre>	2	8	377	141	-797us[-797us] +/-	59ms
^- 66-232-97-8.static.hvvc.>	2	7	377	206	+1669us[+1686us] +/-	133ms
^+ 85.199.214.102	1	8	377	205	+175us[+192us] +/-	12ms
^* 46-243-26-34.tangos.nl	1	8	377	141	-123us[-106us] +/-	10ms
<pre>^- pugot.canonical.com</pre>	2	8	377	21	-95us[-95us] +/-	57ms
^- alphyn.canonical.com	2	6	377	23	-1569us[-1569us] +/-	79ms
<pre>^- golem.canonical.com</pre>	2	7	377	92	-1018us[-1018us] +/-	31ms
^- chilipepper.canonical.com	2	8	377	21	-1106us[-1106us] +/-	27ms

chronyc sourcestats

210 Number of sources = 20							
Name/IP Address	NP	NR	Span	Frequency	Freq Skew	0ffset	Std Dev

gamma.rueckgr.at	25	15	32m	-0.007	0.142	-878us	106us
2b.ncomputers.org	26	16	35m	-0.132	0.283	-1169us	256us
www.kashra.com	25	15	32m	-0.092	0.259	+3426us	195us
<pre>stratum2-4.NTP.TechFak.U></pre>	25	14	32m	-0.018	0.130	-2056us	96us
zepto.mcl.gg	13	11	21m	+0.148	0.196	-683us	66us
mirrorhost.pw	6	5	645	+0.117	0.445	-591us	19us
atto.mcl.gg	21	13	25m	-0.069	0.199	-904us	103us
static.140.107.46.78.cli>	25	18	34m	-0.005	0.094	-1526us	78us
4.53.160.75	25	10	32m	+0.412	0.110	-11ms	84us
37.44.185.42	24	12	30m	-0.983	0.173	-3718us	122us
bagnikita.com	17	7	31m	-0.132	0.217	+3527us	139us
europa.ellipse.net	26	15	35m	+0.038	0.553	-473us	424us
tethys.hot-chilli.net	25	11	32m	-0.094	0.110	-864us	88us
66-232-97-8.static.hvvc.>	20	11	35m	-0.116	0.165	+1561us	109us
85.199.214.102	26	11	35m	-0.054	0.390	+129us	343us
46-243-26-34.tangos.nl	25	16	32m	+0.129	0.297	-307us	198us
pugot.canonical.com	25	14	34m	-0.271	0.176	-143us	135us
alphyn.canonical.com	17	11	1100	-0.087	0.360	-1749us	114us
golem.canonical.com	23	12	30m	+0.057	0.370	-988us	229us
chilipepper.canonical.com	25	18	34m	-0.084	0.224	-1116us	169us

Certain chronyc commands are privileged and can not be run via the network without explicitly allowing them. See section *Command and monitoring access* in man chrony.conf for more details. A local admin can use sudo as usually as this will grant him access to the local admin socket /var/run/chrony/chronyd.sock.

PPS Support

Chrony supports various PPS types natively. It can use kernel PPS API as well as PTP hardware clock. Most general GPS receivers can be leveraged via GPSD. The latter (and potentially more) can be accessed via *SHM* or via a *socket* (recommended). All of the above can be used to augment chrony with additional high quality time sources for better accuracy, jitter, drift, longer-or-short term accuracy (Usually each kind of clock type is good at one of those, but non-perfect at the others). For more details on configuration see some of the external PPS/GPSD resource listed below.

Note: at the release of 20.04 there was a bug which until fixed you might want to add this content to your /etc/apparmor.d/local/usr.sbin.gpsd.

Example configuration for GPSD to feed Chrony

For the setup you need \$ sudo apt install gpsd chrony

But you will want to test/debug your setup and especially the GPS reception, therefore also install \$ sudo apt install pps-tools gpsd-clients

GPS devices usually will communicate via serial interfaces, yet the most common type these days are USB GPS devices which have a serial converter behind USB. If you want to use this for PPS please be aware that the majority does not signal PPS via USB. Check the GPSD hardware list for details. The examples below were run with a Navisys GR701-W.

When plugging in such a device (or at boot time) dmesg should report serial connection of some sorts, example:

52.442199] usb 1-1.1: new full-speed USB device number 3 using xhci hcd [52.546639] usb 1-1.1: New USB device found, idVendor=067b, idProduct=2303, bcdDevice= 4.00 ſ 52.546654] usb 1-1.1: New USB device strings: Mfr=1, Product=2, SerialNumber=0 ſ [52.546665] usb 1-1.1: Product: USB-Serial Controller D 52.546675] usb 1-1.1: Manufacturer: Prolific Technology Inc. [52.602103] usbcore: registered new interface driver usbserial generic [52.602244] usbserial: USB Serial support registered for generic [[52.609471] usbcore: registered new interface driver pl2303 52.609503] usbserial: USB Serial support registered for pl2303 [[52.609564] pl2303 1-1.1:1.0: pl2303 converter detected 52.618366] usb 1-1.1: pl2303 converter now attached to ttyUSB0 ſ

We see above that it appeared as ttyUSB0. To later on have chrony accept being feeded time information by that we have to set it up in /etc/chrony.conf (Please replace USB0 to whatever applies to your setup):

refclock SHM 0 refid GPS precision 1e-1 offset 0.9999 delay 0.2 refclock SOCK /var/run/chrony.ttyUSB0.sock refid PPS

Then restart chrony to make the socket available and waiting. sudo systemctl restart chrony

Next one needs to tell gpsd which device to manager, therefore in /etc/default/gpsd we set: DEVICES="/dev/ttyUSB0"

Furthermore the *default* use-case of gpsd is, well for *gps position tracking*. Therefore it will normally not consume any cpu since it is not running the service but waiting on a *socket* for clients. Furthermore the client will then tell gpsd what it requests and gpsd will only do that.

For the use case of gpsd as PPS-providing-daemon you want to set the option to:

- immediately start even without a client connected, this can be set in GPSD_OPTIONS of /etc/default/gpsd:
 GPSD_OPTIONS="-n"
- enable the service itself and not wait for a client to reach the socket in the future:
 sudo systemctl enable /lib/system/system/gpsd.service

Restarting gpsd will now initialize the PPS from GPS and in dmesg you will see

```
pps_ldisc: PPS line discipline registered
pps pps0: new PPS source usbserial0
pps pps0: source "/dev/ttyUSB0" added
```

In case you have multiple PPS the tool ppsfind might be useful to identify which PPS belongs to which GPS. You could check that with:

```
$ sudo ppsfind /dev/ttyUSB0
pps0: name=usbserial0 path=/dev/ttyUSB0
```

To get any further you need your GPS to get a lock. Tools like cgps or gpsmon need to report a 3D Fix for the data being any good. Then you'd want to check to really have PPS data reported on that with ppstest

```
$ cgps
```

. . .

Status: 3D FIX (7 secs) ...

Next one might want to ensure that the pps device really submits PPS data, to do so run ppstest:

```
$ sudo ppstest /dev/pps0
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1588140739.099526246, sequence: 69 - clear 1588140739.999663721, sequence: 70
source 0 - assert 1588140740.099661485, sequence: 70 - clear 1588140739.999663721, sequence: 70
source 0 - assert 1588140740.099661485, sequence: 70 - clear 1588140740.999786664, sequence: 71
source 0 - assert 1588140741.099792447, sequence: 71 - clear 1588140740.999786664, sequence: 71
```

Ok, gpsd is now running, the GPS reception has found a fix and this is fed into chrony. Now lets check that from the point of view of chrony.

Initially (e.g. before gpsd has started or before it has a lock) those will be new and "untrusted" sources marked with an "?". If your devices stay in the "?" state and won't leave it even after quite some time then gpsd seems not to feed any data to chrony and you'd need to debug why.

ch 21(chronyc> sources 210 Number of sources = 10								
MS	Name/IP	address	Stratum	Poll	Reach	LastRx	Last samp	le 	
 #?	GPS			4	 0	-	+0ns[+0ns] +/-	0ns
#?	PPS		0	4	Θ	-	+0ns[+0ns] +/-	0ns

Over time chrony will classify them as good or bad.

In the example case the raw GPS had too much deviation but PPS is good.

chronyc> sources 210 Number of sources = 10 MS Name/IP address Stratum Poll Reach LastRx Last sample #x GPS 0 4 177 24 -876ms[-876ms] +/- 200ms
#- PPS 0 4 177 21 +916us[+916us] +/- 63us
^- chilipepper.canonical.com 2 6 37 53 +33us[+33us] +/- 33ms

And finally after a while it used the hardware PPS input as it was better:

chronyc> sources							
210 Number of sources = 10							
MS Name/IP address	Stratum	Poll	Reach	LastR×	Last sam	ple	
#x GPS	 0	===== 4	====== 377	====== 20	-884ms[-884ms] +/·	- 200ms
#* PPS	Θ	4	377	18	- +6677ns[+52us] +/-	. 58us
^- alphyn.canonical.com	2	6	377	20	-1303us[-	1258us] +/-	· 114ms
	. 1.		1.	1	• 1		

The PPS might also be ok but used in a combined way with e.g. the selected server. See man chronyc for more details about how all these combinations will look like.

```
chronyc> sources
```

210 Number of sources = 11

MS Name/IP address Stratum Poll Reach LastRx Last sample

===		====			======				
#?	GPS	0	4	0	-	+0ns[+0ns]	+/-	0ns
#+	PPS	0	4	377	22	+154us[+154us]	+/-	8561us
^*	chilipepper.canonical.com	2	6	377	50	-353us[-300us]	+/-	44ms

And if you wonder if your shm based GPS data is any good, you can check for that as well.

First of all chrony will not only tell you if it classified it as good or bad. Via sourcestats you can also check the details

```
chronyc> sourcestats
```

```
210 Number of sources = 10
```

Name/IP Address	NP	NR	Span	Frequency	Freq Skew	Offset	Std Dev
 GPS	20	9	 302	+1.993	11.501	-868ms	1208us
PPS	6	3	78	+0.324	5.009	+3365ns	41us
golem.canonical.com	15	10	783	+0.859	0.509	-750us	108us

You can also track the raw data that gpsd or other ntpd compliant refclocks are sending via shared memory by using ntpshmmon:

```
$ sudo ntpshmmon -o
ntpshmmon: version 3.20
#
      Name
                    Offset
                                 Clock
                                                       Real
                                                                            I Prc
                    0.000223854 1588265805.000223854 1588265805.000000000 0 -10
sample NTP1
                    0.125691783 1588265805.125999851 1588265805.000308068 0 -20
sample NTP0
sample NTP1
                    0.000349341 1588265806.000349341 1588265806.000000000 0 -10
sample NTP0
                    0.130326636 1588265806.130634945 1588265806.000308309 0 -20
                    0.000485216 1588265807.000485216 1588265807.000000000 0 -10
sample NTP1
```

NTS Support

In Chrony 4.0 (first appeared in Ubuntu 21.04 Hirsute) support for Network Time Security "NTS" as added.

NTS Server

To set up your server with NTS you'll need certificates so that the server can authenticate itself and based on that allow to encrypt and verify the NTP traffic.

In addition to the allow statement that any chrony working as NTP server needs there are two mandatory config entries that will be needed. Those for the certificates like

Example entries would look like:

ntsservercert /etc/chrony/fullchain.pem
ntsserverkey /etc/chrony/privkey.pem

It is important to note that for isolation reasons chrony by default runs as user and group _chrony. Therefore you need to grant access to the certificates for that user.

```
$ sudo chown _chrony:_chrony /etc/chrony/*.pem
```

Then restart chrony systemctl restart chrony and it will be ready to provide NTS based time service.

A running chrony server has various statistics, one also accounts the number of NTS connections that were established (or dropped):

\$ sudo chronyc -N serverstats NTP packets received : 213 NTP packets dropped : 0 Command packets received : 117 Command packets dropped : 0 Client log records dropped : 0 NTS-KE connections accepted: 2 NTS-KE connections dropped : 0 Authenticated NTP packets : 197

And there is also a per-client statistic which can be enabled by the -p option of the clients command.

<pre>\$ sudo chronyc -N clients -k</pre>									
Hostname	NTP	Drop	Int	IntL	Last	NTS-KE	Drop I	nt	Last
	=====							===	
10.172.196.173	197	Θ	10	-	595	2	Θ	5	48h

For more complex scenarios there are more advanced options to configure NTS documented in the man page.

A Note about certificate placement:

Chrony by default is isolated via apparmor and in addition uses all kind of protect* features of systemd. Due to that there are not many paths chrony can access for the certificates. But /etc/chrony/* is allowed as read-only and that is enough.

 $Check \verb|/etc/apparmor.d/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in \verb|/etc/apparmor.d/local/usr.sbin.chronyd if you want other paths or allow custom paths in bataa custom paths in bataa custom paths in bataa custom paths i$

NTS Client

The client needs to specify server as usual (pool directives do not work with NTS). As usual after the server adress options can be listed and there nts has to be added.

Example:

server <server-fqdn-or-IP> iburst nts

One can check the authdata of the connections the client established like

<pre>\$ sudo chronyc -N authdata</pre>									
Name/IP address	Mode	KeyID	Туре	KLen	Last	Atmp	NAK	Cook	CLen
	=====		=====	=====	=====		=====	:====	
<server-fqdn-or-ip></server-fqdn-or-ip>	NTS	1	15	256	48h	Θ	Θ	8	100

Again there are more advanced options documented in the man page. Common use cases are specifying an explicit trusted certificate.

Bad Clocks and secure time syncing - "A Chicken and Egg" Problem:

There is one problem with systems that have very bad clocks. NTS is based on TLS and that needs a a somewhat correct clock. Due to that an NTS based sync might fail. On Hardware affected by this one can consider using the nocerttimecheck option which allows to set a number of times time can be synced without checking validation and expiration.

References

- Chrony FAQ
- ntp.org: home of the Network Time Protocol project
- pool.ntp.org: project of virtual cluster of timeservers
- Freedesktop.org info on timedatectl
- Freedesktop.org info on systemd-timesyncd service
- Feeding chrony from GPSD
- See the Ubuntu Time wiki page for more information.

Data Plane Development Kit

The DPDK is a set of libraries and drivers for fast packet processing and runs mostly in Linux userland. It is a set of libraries that provide the so called "Environment Abstraction Layer" (EAL). The EAL hides the details of the environment and provides a standard programming interface. Common use cases are around special solutions for instance network function virtualization and advanced high-throughput network switching. The DPDK uses a run-to-completion model for fast data plane performance and accesses devices via polling to eliminate the latency of interrupt processing at the tradeoff of higher cpu consumption. It was designed to run on any processors. The first supported CPU was Intel x86 and it is now extended to IBM PPC64 and ARM64.

Ubuntu further provides some infrastructure to ease DPDKs usability.

Prerequisites

This package is currently compiled for the lowest possible CPU requirements allowed by upstream. Starting with DPDK 17.08 that means it requires at least SSE4_2 and anything else activated by -march=corei7 (in gcc) to be supported by the CPU.

The list of upstream DPDK supported network cards can be found at supported NICs. But a lot of those are disabled by default in the upstream Project as they are not yet in a stable state. The subset of network cards that DPDK has enabled in the package as available in Ubuntu 16.04 is:

DPDK has "userspace" drivers for the cards called PMDs.

The packages for these follow the pattern of librte-pmd-<type>-<version>. Therefore the example for an intel e1000 in 18.11 would be librte-pmd-e1000-18.11.

The more commonly used, tested and fully supported drivers are installed as dependencies of dpdk. But there are way more in universe that follow the same naming pattern.

Unassigning the default Kernel drivers

Cards have to be unassigned from their kernel driver and instead be assigned to uio_pci_generic of vfio-pci. uio_pci_generic is older and usually getting to work more easily, but also has less features and isolation.

The newer vfio-pci requires that you activate the following kernel parameters to enable iommu.

iommu=pt intel_iommu=on

Or on AMD

amd_iommu=pt

On top for vfio-pci you then have to configure and assign the iommu groups accordingly. That is mostly done in Firmware and by HW layout, you can check the group assignment the kernel probed in /sys/kernel/iommu_groups/.

Note: virtio is special, dpdk can directly work on those devices without vfio_pci/uio_pci_generic. But to avoid issues by kernel and DPDK managing the device you still have to unassign the kernel driver.

Manual configuration and status checks can be done via sysfs or with the tool ${\tt dpdk_nic_bind}$

dpdk_nic_bind.py --help

Usage:

dpdk-devbind.py [options] DEVICE1 DEVICE2

where DEVICE1, DEVICE2 etc, are specified via PCI "domain:bus:slot.func" syntax or "bus:slot.func" syntax. For devices bound to Linux kernel drivers, they may also be referred to by Linux interface name e.g. eth0, eth1, em0, em1, etc.

Options: --help, --usage: Display usage information and quit

-s, --status: Print the current status of all known network, crypto, event and mempool devices. For each device, it displays the PCI domain, bus, slot and function, along with a text description of the device. Depending upon whether the device is being used by a kernel driver, the igb_uio driver, or no

driver, other relevant information will be displayed: * the Linux interface name e.g. if=eth0 * the driver being used e.g. drv=igb_uio * any suitable drivers not currently using that device e.g. unused=igb uio NOTE: if this flag is passed along with a bind/unbind option, the status display will always occur after the other operations have taken place. --status-dev: Print the status of given device group. Supported device groups are: "net", "crypto", "event", "mempool" and "compress" -b driver, --bind=driver: Select the driver to use or "none" to unbind the device -u, --unbind: Unbind a device (Equivalent to "-b none") --force: By default, network devices which are used by Linux - as indicated by having routes in the routing table - cannot be modified. Using the --force flag overrides this behavior, allowing active links to be forcibly unbound. WARNING: This can lead to loss of network connection and should be used with caution. Examples: To display current device status: dpdk-devbind.py --status To display current network device status: dpdk-devbind.py --status-dev net To bind ethl from the current driver and move to use igb uio dpdk-devbind.py --bind=igb_uio eth1 To unbind 0000:01:00.0 from using any driver dpdk-devbind.py -u 0000:01:00.0

To bind 0000:02:00.0 and 0000:02:00.1 to the ixgbe kernel driver dpdk-devbind.py -b ixgbe 02:00.0 02:00.1

DPDK Device configuration

The package dpdk provides init scripts that ease configuration of device assignment and huge pages. It also makes them persistent across reboots.

The following is an example of the file /etc/dpdk/interfaces configuring two ports of a network card. One with uio_pci_generic and the other one with vfio-pci.

```
# <bus> Currently only "pci" is supported
# <id> Device ID on the specified bus
# <driver> Driver to bind against (vfio-pci or uio_pci_generic)
#
# Be aware that the two DPDK compatible drivers uio_pci_generic and vfio-pci are
# part of linux-image-extra-<VERSION> package.
# This package is not always installed by default - for example in cloud-images.
# So please install it in case you run into missing module issues.
#
# <bus> <id> <driver>
pci 0000:04:00.0 uio_pci_generic
```

pci 0000:04:00.1 vfio-pci

Cards are identified by their PCI-ID. If you are unsure you might use the tool dpdk_nic_bind.py to show the current available devices and the drivers they are assigned to.

```
dpdk_nic_bind.py --status
```

<none>

DPDK HugePage configuration

DPDK makes heavy use of huge pages to eliminate pressure on the TLB. Therefore hugepages have to be configured in your system.

The *dpdk* package has a config file and scripts that try to ease hugepage configuration for DPDK in the form of /etc/dpdk/dpdk.conf. If you have more consumers of hugepages than just DPDK in your system or very special requirements how your hugepages are going to be set up you likely want to allocate/control them by yourself. If not this can be a great simplification to get DPDK configured for your needs.

Here an example configuring 1024 Hugepages of 2M each and 4 1G pages.

NR_2M_PAGES=1024 NR_1G_PAGES=4

As shown this supports configuring 2M and the larger 1G hugepages (or a mix of both). It will make sure there are proper hugetlbfs mountpoints for DPDK to find both sizes no matter what your default huge page size is. The config file itself holds more details on certain corner cases and a few hints if you want to allocate hugepages manually via a kernel parameter.

It depends on your needs which size you want - 1G pages are certainly more effective regarding TLB pressure. But there were reports of them fragmenting inside the DPDK memory allocations. Also it can be harder to grab enough free space to set up a certain amount of 1G pages later in the life-cycle of a system.

Compile DPDK Applications

Currently there are not a lot consumers of the DPDK library that are stable and released. OpenVswitch-DPDK being an exception to that (see below) and more are appearing. But in general it might still happen that you might want to compile an app against the library.

You will often find guides that tell you to fetch the DPDK sources, build them to your needs and eventually build your application based on DPDK by setting values RTE_* for the build system. Since Ubuntu provides an already compiled DPDK for you can can skip all that.

DPDK provides a valid pkg-config file to simplify setting the proper variables and options.

sudo apt-get install dpdk-dev libdpdk-dev
gcc testdpdkprog.c \$(pkg-config --libs --cflags libdpdk) -o testdpdkprog

An example of a complex (autoconfigure) user of pkg-config of DPDK including fallbacks to older non pkg-config style can be seen in the OpenVswitch build system.

Depending on what you build it might be a good addition to install all of DPDK build dependencies before the make, which on Ubuntu can be done automatically with.

DPDK in KVM Guests

If you have no access to DPDK supported network cards you can still work with DPDK by using its support for virtio. To do so you have to create guests backed by hugepages (see above).

On top of that there it is required to have at least SSE3. The default CPU model qemu/libvirt uses is only up to SSE2. So you will have to define a model that passed the proper feature flags (or use host-passthrough). An example can be found in following snippet to your virsh xml (or the equivalent virsh interface you use).

<cpu mode='host-passthrough'>

Also virtio nowadays supports multiqueue which DPDK in turn can exploit for better speed. To modify a normal virtio definition to have multiple queues add the following to your interface definition. This is about enhancing a normal virtio nic to have multiple queues, to later on be consumed e.g. by DPDK in the guest.

<driver name="vhost" queues="4"/>

Use DPDK

Since DPDK on its own is only (massive) library you most likely might continue to OpenVswitch-DPDK as an example to put it to use.

Resources

- DPDK Documentation
- Release Notes matching the version packages in Ubuntu 16.04
- Linux DPDK User Getting Started
- EAL Command-line Options
- DPDK Api Documentation
- OpenVswitch DPDK installation
- Wikipedias definition of DPDK

OpenVswitch-DPDK

With DPDK being *just* a library it doesn't do a lot on its own, so it depends on emerging projects making use of it. One consumer of the library that already is part of Ubuntu is OpenVswitch with DPDK support in the package openvswitch-switch-dpdk.

Here an example how to install and configure a basic OpenVswitch using DPDK for later use via libvirt/qemu-kvm.

```
sudo apt-get install openvswitch-switch-dpdk
sudo update-alternatives --set ovs-vswitchd /usr/lib/openvswitch-switch-dpdk/ovs-vswitchd-dpdk
ovs-vsctl set Open_vSwitch . "other_config:dpdk-init=true"
# run on core 0 only
ovs-vsctl set Open_vSwitch . "other_config:dpdk-lcore-mask=0x1"
# Allocate 2G huge pages (not Numa node aware)
ovs-vsctl set Open_vSwitch . "other_config:dpdk-alloc-mem=2048"
# limit to one whitelisted device
ovs-vsctl set Open_vSwitch . "other_config:dpdk-extra=--pci-whitelist=0000:04:00.0"
sudo service openvswitch-switch restart
```

Please remember that you have to assign devices to DPDK compatible drivers see above at Network - DPDK before restarting.

Please note that the section dpdk-alloc-mem=2048 above is the most basic numa setup for a single socket system. If you have multiple sockets you might want to define how to split your memory among them. More details about these options are outlined in OpenVswitch setup.

Attaching DPDK ports to OpenVswitch

The OpenVswitch you now started supports all port types OpenVswitch usually does, plus DPDK port types. Here an example how to create a bridge and - instead of a normal external port - add an external DPDK port to it. When doing so you can specify the device that will be associated.

ovs-vsctl add-br ovsdpdkbr0 -- set bridge ovsdpdkbr0 datapath_type=netdev ovs-vsctl add-port ovsdpdkbr0 dpdk0 -- set Interface dpdk0 type=dpdk "options:dpdk-devargs=\${0VSDEV_PCIID}"

Further tuning can be applied by setting options:

ovs-vsctl set Interface dpdk0 "options:n_rxq=2"

OpenVswitch DPDK to KVM Guests

If you are not building some sort of SDN switch or NFV on top of DPDK it is very likely that you want to forward traffic to KVM guests. The good news is, that with the new qemu/libvirt/dpdk/openvswitch versions in Ubuntu this is no more about manually appending commandline string. This chapter covers a basic configuration how to connect a KVM guest to a OpenVswitch-DPDK instance.

The recommended way to get to a KVM guest is using vhost_user_client.

This will cause OVS-DPDK to create connect to a socket that qemu created. That way old issues like guest failures on OVS restart are avoided. Here an example how to add such a port to the bridge you created above.

ovs-vsctl add-port ovsdpdkbr0 vhost-user-1 -- set Interface vhost-user-1 type=dpdkvhostuserclient "options:vhostserver-path=/var/run/vhostuserclient/vhost-user-client-1"

This will connect to the specified path that has to be created by a guest listening on it.

To let libvirt/kvm consume this socket and create a guest virtio network device for it add a snippet like this to your guest definition as the network definition.

```
<interface type='vhostuser'>
<source type='unix'
path='/var/run/vhostuserclient/vhost-user-client-1'
mode='server'/>
<model type='virtio'/>
</interface>
```

Tuning Openvswitch-DPDK

DPDK has plenty of options - in combination with Openvswitch-DPDK the two most commonly used are:

```
ovs-vsctl set Open_vSwitch . other_config:n-dpdk-rxqs=2
ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x6
```

The first select how many rx queues are to be used for each DPDK interface, while the second controls how many and where to run PMD threads. The example above will utilize two rx queues and run PMD threads on CPU 1 and 2. See the referred links to "EAL Command-line Options" and "OpenVswitch DPDK installation" at the end of this document for more.

As usual with tunings you have to know your system and workload really well - so please verify any tunings with workloads matching your real use case.

Support and Troubleshooting

DPDK is a fast evolving project. In any case of a search for support and further guides it is highly recommended to first check if they apply to the current version.

Check if it is a known issue on:

- DPDK Mailing Lists
- For OpenVswitch-DPDK OpenStack Mailing Lists
- Known issues in DPDK Launchpad Area
- Join the IRC channels #DPDK or #openvswitch on freenode.

Issues are often due to missing small details in the general setup. Later on, these missing details cause problems which can be hard to track down to their root cause. A common case seems to be the "could not open network device dpdk0 (No such device)" issue. This occurs rather late when setting up a port in Open vSwitch with DPDK. But the root cause most of the time is very early in the setup and initialization. Here an example how a proper initialization of a device looks - this can be found in the syslog/journal when starting Open vSwitch with DPDK enabled.

ovs-ctl[3560]: EAL: PCI device 0000:04:00.1 on NUMA socket 0
ovs-ctl[3560]: EAL: probe driver: 8086:1528 rte_ixgbe_pmd
ovs-ctl[3560]: EAL: PCI memory mapped at 0x7f2140000000
ovs-ctl[3560]: EAL: PCI memory mapped at 0x7f2140200000

If this is missing, either by ignored cards, failed initialization or other reasons, later on there will be no DPDK device to refer to. Unfortunately the logging is spread across syslog/journal and the openvswitch log. To allow some cross checking here an example what can be found in these logs, relative to the entered command.

#Note: This log was taken with dpdk 2.2 and openvswitch 2.5 but still looks quite similar (a bit extended) these days Captions:

CMD: that you enter SYSLOG: (Inlcuding EAL and OVS Messages) OVS-LOG: (Openvswitch messages)

#PREPARATION

Bind an interface to DPDK UIO drivers, make Hugepages available, enable DPDK on OVS

CMD: sudo service openvswitch-switch restart

SYSLOG:

```
2016-01-22T08:58:31.372Z|00003|daemon_unix(monitor)|INF0|pid 3329 died, killed (Terminated), exiting
2016-01-22T08:58:33.377Z|00002|vlog|INF0|opened log file /var/log/openvswitch/ovs-vswitchd.log
2016-01-22T08:58:33.381Z|00003|ovs_numa|INF0|Discovered 12 CPU cores on NUMA node 0
2016-01-22T08:58:33.381Z|00004|ovs_numa|INF0|Discovered 1 NUMA nodes and 12 CPU cores
2016-01-22T08:58:33.381Z|00005|reconnect|INF0|unix:/var/run/openvswitch/db.sock: connecting...
2016-01-22T08:58:33.383Z|00006|reconnect|INF0|unix:/var/run/openvswitch/db.sock: connected
2016-01-22T08:58:33.386Z|00007|bridge|INF0|ovs-vswitchd (Open vSwitch) 2.5.0
OVS-LOG:
systemd[1]: Stopping Open vSwitch...
systemd[1]: Stopped Open vSwitch.
systemd[1]: Stopping Open vSwitch Internal Unit...
ovs-ctl[3541]: * Killing ovs-vswitchd (3329)
ovs-ctl[3541]: * Killing ovsdb-server (3318)
systemd[1]: Stopped Open vSwitch Internal Unit.
systemd[1]: Starting Open vSwitch Internal Unit...
ovs-ctl[3560]: * Starting ovsdb-server
ovs-vsctl: ovs/00001/vsctl/INF0/Called as ovs-vsctl --no-wait -- init -- set Open_vSwitch . db-version=7.12.1
ovs-vsctl: ovs/00001/vsctl/INF0/Called as ovs-vsctl --no-wait set Open_vSwitch . ovs-version=2.5.0 "external-
ids:system-id=\"e7c5ba80-bb14-45c1-b8eb-628f3ad03903\"" "system-type=\"Ubuntu\"" "system-version=\"16.04-
xenial\""
ovs-ctl[3560]: * Configuring Open vSwitch system IDs
ovs-ctl[3560]: 2016-01-22T08:58:31Z|00001|dpdk|INF0|No -vhost_sock_dir provided - defaulting to /var/run/openvswitch
ovs-vswitchd: ovs/00001/dpdk/INFO/No -vhost_sock_dir provided - defaulting to /var/run/openvswitch
ovs-ctl[3560]: EAL: Detected lcore 0 as core 0 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 1 as core 1 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 2 as core 2 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 3 as core 3 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 4 as core 4 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 5 as core 5 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 6 as core 0 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 7 as core 1 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 8 as core 2 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 9 as core 3 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 10 as core 4 on socket 0
ovs-ctl[3560]: EAL: Detected lcore 11 as core 5 on socket 0
ovs-ctl[3560]: EAL: Support maximum 128 logical core(s) by configuration.
ovs-ctl[3560]: EAL: Detected 12 lcore(s)
ovs-ctl[3560]: EAL: VFIO modules not all loaded, skip VFIO support...
ovs-ctl[3560]: EAL: Setting up physically contiguous memory...
ovs-ctl[3560]: EAL: Ask a virtual area of 0x100000000 bytes
ovs-ctl[3560]: EAL: Virtual area found at 0x7f2040000000 (size = 0x100000000)
ovs-ctl[3560]: EAL: Requesting 4 pages of size 1024MB from socket 0
```

ovs-ctl[3560]: EAL: TSC frequency is ~2397202 KHz ovs-vswitchd[3592]: EAL: TSC frequency is ~2397202 KHz ovs-vswitchd[3592]: EAL: Master lcore 0 is ready (tid=fc6cbb00;cpuset=[0]) ovs-vswitchd[3592]: EAL: PCI device 0000:04:00.0 on NUMA socket 0 ovs-vswitchd[3592]: EAL: probe driver: 8086:1528 rte_ixgbe_pmd ovs-vswitchd[3592]: EAL: Not managed by a supported kernel driver, skipped ovs-vswitchd[3592]: EAL: PCI device 0000:04:00.1 on NUMA socket 0 ovs-vswitchd[3592]: EAL: probe driver: 8086:1528 rte_ixgbe_pmd ovs-vswitchd[3592]: EAL: PCI memory mapped at 0x7f2140000000 ovs-vswitchd[3592]: EAL: PCI memory mapped at 0x7f2140200000 ovs-ctl[3560]: EAL: Master lcore 0 is ready (tid=fc6cbb00;cpuset=[0]) ovs-ctl[3560]: EAL: PCI device 0000:04:00.0 on NUMA socket 0 ovs-ctl[3560]: EAL: probe driver: 8086:1528 rte ixgbe pmd ovs-ctl[3560]: EAL: Not managed by a supported kernel driver, skipped ovs-ctl[3560]: EAL: PCI device 0000:04:00.1 on NUMA socket 0 ovs-ctl[3560]: EAL: probe driver: 8086:1528 rte_ixgbe_pmd ovs-ctl[3560]: EAL: PCI memory mapped at 0x7f2140000000 ovs-ctl[3560]: EAL: PCI memory mapped at 0x7f2140200000 ovs-vswitchd[3592]: PMD: eth_ixgbe_dev_init(): MAC: 4, PHY: 3 ovs-vswitchd[3592]: PMD: eth_ixgbe_dev_init(): port 0 vendorID=0x8086 deviceID=0x1528 ovs-ctl[3560]: PMD: eth_ixgbe_dev_init(): MAC: 4, PHY: 3 ovs-ctl[3560]: PMD: eth_ixgbe_dev_init(): port 0 vendorID=0x8086 deviceID=0x1528 ovs-ctl[3560]: Zone 0: name:<RG_MP_log_history>, phys:0x83fffdec0, len:0x2080, virt:0x7f213fffdec0, socket_id:0, flags:0 ovs-ctl[3560]: Zone 1: name:<MP_log_history>, phys:0x83fd73d40, len:0x28a0c0, virt:0x7f213fd73d40, socket_id:0, flags:0 ovs-ctl[3560]: Zone 2: name:<rte eth dev data>, phys:0x83fd43380, len:0x2f700, virt:0x7f213fd43380, socket id:0, flags:0 ovs-ctl[3560]: * Starting ovs-vswitchd ovs-ctl[3560]: * Enabling remote OVSDB managers systemd[1]: Started Open vSwitch Internal Unit. systemd[1]: Starting Open vSwitch... systemd[1]: Started Open vSwitch.

CMD: sudo ovs-vsctl add-br ovsdpdkbr0 -- set bridge ovsdpdkbr0 datapath_type=netdev

SYSLOG:

```
2016-01-22T08:58:56.344Z|00008|memory|INF0|37256 kB peak resident set size after 24.5 seconds
2016-01-22T08:58:56.346Z|00009|ofproto_dpif|INF0|netdev@ovs-netdev: Datapath supports recirculation
2016-01-22T08:58:56.346Z|00011|ofproto_dpif|INF0|netdev@ovs-netdev: MPLS label stack length probed as 3
2016-01-22T08:58:56.346Z|00012|ofproto_dpif|INF0|netdev@ovs-netdev: Datapath supports unique flow ids
2016-01-22T08:58:56.346Z|00012|ofproto_dpif|INF0|netdev@ovs-netdev: Datapath does not support ct_state
2016-01-22T08:58:56.346Z|00013|ofproto_dpif|INF0|netdev@ovs-netdev: Datapath does not support ct_zone
2016-01-22T08:58:56.346Z|00014|ofproto_dpif|INF0|netdev@ovs-netdev: Datapath does not support ct_mark
2016-01-22T08:58:56.346Z|00015|ofproto_dpif|INF0|netdev@ovs-netdev: Datapath does not support ct_label
2016-01-22T08:58:56.346Z|00016|bridge|INF0|netdev@ovs-netdev: Datapath does not support ct_label
2016-01-22T08:58:56.360Z|00016|bridge|INF0|netdev@ovs-netdev: Datapath does not support ct_label
2016-01-22T08:58:56.361Z|00016|bridge|INF0|bridge ovsdpdkbr0: added interface ovsdpdkbr0 on port 65534
2016-01-22T08:58:56.361Z|00017|bridge|INF0|bridge ovsdpdkbr0: using datapath ID 00005a4aled0a14d
2016-01-22T08:58:56.361Z|00018|connmgr|INF0|ovsdpdkbr0: added service controller "punix:/var/run/openvswitch/ovsdpdkbr0
```

OVS-LOG:

```
ovs-vsctl: ovs|00001|vsctl|INF0|Called as ovs-vsctl add-br ovsdpdkbr0 -- set bridge ovsdpdkbr0 datapath_type=netdev
systemd-udevd[3607]: Could not generate persistent MAC address for ovs-netdev: No such file or directory
kernel: [50165.886554] device ovs-netdev entered promiscuous mode
kernel: [50165.901261] device ovsdpdkbr0 entered promiscuous mode
```

CMD: sudo ovs-vsctl add-port ovsdpdkbr0 dpdk0 -- set Interface dpdk0 type=dpdk

SYSLOG:

```
2016-01-22T08:59:06.369Z|00019|memory|INF0|peak resident set size grew 155% in last 10.0 seconds, from 37256 kB to 95008 k
2016-01-22T08:59:06.369Z|00020|memory|INF0|handlers:4 ports:1 revalidators:2 rules:5
2016-01-22T08:59:30.989Z|00021|dpdk|INF0|Port 0: 8c:dc:d4:b3:6d:e9
2016-01-22T08:59:31.521Z|00022|dpdk|INF0|Port 0: 8c:dc:d4:b3:6d:e9
2016-01-22T08:59:31.521Z|00023|dpif_netdev|INF0|Created 1 pmd threads on numa node 0
2016-01-22T08:59:31.522Z|00001|dpif_netdev(pmd16)|INF0|Core 0 processing port 'dpdk0'
```

```
2016-01-22T08:59:31.522Z|00024|bridge|INF0|bridge ovsdpdkbr0: added interface dpdk0 on port 1
2016-01-22T08:59:31.522Z|00025|bridge|INF0|bridge ovsdpdkbr0: using datapath ID 00008cdcd4b36de9
2016-01-22T08:59:31.523Z|00002|dpif netdev(pmd16)|INF0|Core 0 processing port 'dpdk0'
OVS-LOG:
ovs-vsctl: ovs/00001/vsctl/INF0/Called as ovs-vsctl add-port ovsdpdkbr0 dpdk0 -- set Interface dpdk0 type=dpdk
ovs-vswitchd[3595]: PMD: ixgbe_dev_tx_queue_setup(): sw_ring=0x7f211a79ebc0 hw_ring=0x7f211a7a6c00 dma_addr=0x81a7a6c00
ovs-vswitchd[3595]: PMD: ixgbe_set_tx_function(): Using simple tx code path
ovs-vswitchd[3595]: PMD: ixgbe_set_tx_function(): Vector tx enabled.
ovs-vswitchd[3595]: PMD: ixgbe_dev_rx_queue_setup(): sw_ring=0x7f211a78a6c0 sw_sc_ring=0x7f211a786580 hw_ring=0x7f211a7
ovs-vswitchd[3595]: PMD: ixgbe_set_rx_function(): Vector rx enabled, please make sure RX burst size no less than 4 (port=0
ovs-vswitchd[3595]: PMD: ixgbe dev tx queue setup(): sw ring=0x7f211a79ebc0 hw ring=0x7f211a7a6c00 dma addr=0x81a7a6c00
. . .
CMD: sudo ovs-vsctl add-port ovsdpdkbr0 vhost-user-1 -- set Interface vhost-user-1 type=dpdkvhostuser
OVS-LOG:
2016-01-22T09:00:35.145Z|00026|dpdk|INF0|Socket
                                                 /var/run/openvswitch/vhost-user-1
                                                                                       created
                                                                                                       vhost-
                                                                                                 for
user port vhost-user-1
2016-01-22T09:00:35.145Z|00003|dpif_netdev(pmd16)|INF0|Core 0 processing port 'dpdk0'
2016-01-22T09:00:35.145Z|00004|dpif_netdev(pmd16)|INF0|Core 0 processing port 'vhost-user-1'
2016-01-22T09:00:35.145Z|00027|bridge|INF0|bridge ovsdpdkbr0: added interface vhost-user-1 on port 2
SYSLOG:
ovs-vsctl: ovs/00001/vsctl/INFO/Called as ovs-vsctl add-port ovsdpdkbr0 vhost-user-1 -- set Interface vhost-
user-1 type=dpdkvhostuser
ovs-vswitchd[3595]: VHOST_CONFIG: socket created, fd:46
ovs-vswitchd[3595]: VHOST CONFIG: bind to /var/run/openvswitch/vhost-user-1
Eventually we can see the poll thread in top
```

 PID USER
 PR NI
 VIRT
 RES
 SHR S
 %CPU %MEM
 TIME+ COMMAND

 3595 root
 10 -10 4975344 103936
 9916 S 100.0
 0.3
 33:13.56 ovs-vswitchd

Resources

- DPDK Documentation
- Release Notes matching the version packages in Ubuntu 16.04
- Linux DPDK User Getting Started
- EAL Command-line Options
- DPDK Api Documentation
- OpenVswitch DPDK installation
- Wikipedias definition of DPDK

Device Mapper Multipathing - Introduction

Device Mapper Multipath will be referred here as **multipath** only.

Multipath allows you to configure multiple I/O paths between server nodes and storage arrays into a single device. These I/O paths are physical SAN connections that can include separate cables, switches, and controllers.

Multipathing aggregates the I/O paths, creating a new device that consists of the aggregated paths. This chapter provides an introduction and a high-level overview of multipath.

Overview

Multipath can be used to provide:

• Redundancy

multipath can provide failover in an active/passive configuration. In an active/passive configuration, only half the paths are used at any time for I/O. If any element of an I/O path (the cable, switch, or controller) fails, multipath switches to an alternate path.

• Improved Performance

Multipath can be configured in active/active mode, where I/O is spread over the paths in a round-robin fashion. In some configurations, multipath can detect loading on the I/O paths and dynamically re-balance the load.

Storage Array Overview

It is a very good idea to consult your **storage vendor** *installation guide* for the recommended multipath configuration variables for your storage model. The default configuration will probably work but will likely need adjustments based on your storage setup.

Multipath Components

Component	Description
dm_multipath kernel module	Reroutes I/O and supports failover for paths and path groups.
multipath command multipathd daemon	Monitors paths; as paths fail and come back, it may initiate path group switches. I
kpartx command	Creates device mapper devices for the partitions on a device It is necessary to use t

Multipath Setup Overview

multipath includes compiled-in default settings that are suitable for common multipath configurations. Setting up multipath is often a simple procedure. The basic procedure for configuring your system with multipath is as follows:

- 1. Install the **multipath-tools** and **multipath-tools-boot** packages
- 2. Create an empty config file called /etc/multipath.conf
- 3. Edit the **multipath.conf** file to modify default values and save the updated file.
- 4. Start the multipath daemon
- 5. Update initial ramdisk

For detailed setup instructions for multipath configuration see DM-Multipath Configuration and DM-Multipath Setup.

Multipath Devices

Without multipath, each path from a server node to a storage controller is treated by the system as a separate device, even when the I/O path connects the same server node to the same storage controller. Multipath provides a way of organizing the I/O paths logically, by creating a single device on top of the underlying paths.

Multipath Device Identifiers

Each multipath device has a World Wide Identifier (WWID), which is guaranteed to be globally unique and unchanging. By default, the name of a multipath device is set to its WWID. Alternately, you can set the user_friendly_names option in multipath.conf, which causes multipath to use a node-unique alias of the form mpathn as the name.

For example, a node with two HBAs attached to a storage controller with two ports via a single unzoned FC switch sees four devices: /dev/sda, /dev/sdb, /dev/sdc, and /dev/sdd. Multipath creates a single device with a unique WWID that reroutes I/O to those four underlying devices according to the multipath configuration.

When the user_friendly_names configuration option is set to **yes**, the name of the multipath device is set to **mpathn**. When new devices are brought under the control of multipath, the new devices may be seen in two different places under the /dev directory: /dev/mapper/mpathn and /dev/dm-n.

• The devices in /dev/mapper are created early in the boot process. Use these devices to access the multipathed devices.

• Any devices of the form /dev/dm-n are for internal use only and should never be used directly.

You can also set the name of a multipath device to a name of your choosing by using the **alias** option in the **multipaths** section of the multipath configuration file.

For information on the multipath configuration defaults, including the user_friendly_names and alias configuration options, see DM-Multipath Configuration.

Consistent Multipath Device Names in a Cluster

When the user_friendly_names configuration option is set to yes, the name of the multipath device is unique to a node, but it is not guaranteed to be the same on all nodes using the multipath device. Similarly, if you set the alias option for a device in the multipaths section of /etc/multipath.conf, the name is not automatically consistent across all nodes in the cluster.

This should not cause any difficulties if you use LVM to create logical devices from the multipath device, but if you require that your multipath device names be consistent in every node it is recommended that you leave the **user_friendly_names** option set to **no** and that you not configure aliases for the devices.

If you configure an alias for a device that you would like to be consistent across the nodes in the cluster, you should ensure that the /etc/multipath.conf file is the same for each node in the cluster by following the same procedure:

- 1. Configure the aliases for the multipath devices in the in the multipath.conf file on one machine.
- 2. Disable all of your multipath devices on your other machines by running the following commands:
 - # systemctl stop multipath-tools.service
 # multipath -F
- 3. Copy the /etc/multipath.conf file from the first machine to all the other machines in the cluster.
- 4. Re-enable the multipath daemon on all the other machines in the cluster by running the following command:
 - # systemctl start multipath-tools.service

When you add a new device you will need to repeat this process.

Multipath Device attributes

In addition to the user_friendly_names and alias options, a multipath device has numerous attributes. You can modify these attributes for a specific multipath device by creating an entry for that device in the multipaths section of /etc/multipath.conf.

For information on the multipaths section of the multipath configuration file, see DM-Multipath Configuration.

Multipath Devices in Logical Volumes

After creating multipath devices, you can use the multipath device names just as you would use a physical device name when creating an LVM physical volume.

For example, if /dev/mapper/mpatha is the name of a multipath device, the following command will mark /dev/mapper/mpatha as a physical volume:

pvcreate /dev/mapper/mpatha

You can use the resulting LVM physical device when you create an LVM volume group just as you would use any other LVM physical device.

Note

If you attempt to create an LVM physical volume on a whole device on which you have configured partitions, the pvcreate command will fail.

When you create an LVM logical volume that uses active/passive multipath arrays as the underlying physical devices, you should include filters in the **lvm.conf** to exclude the disks that underlie the multipath devices. This is because if the array automatically changes the active path to the passive path when it receives I/O, multipath will failover and failback whenever LVM scans the passive path if these devices are not filtered.

For active/passive arrays that require a command to make the passive path active, LVM prints a warning message when this occurs. To filter all SCSI devices in the LVM configuration file (lvm.conf), include the following filter in the devices section of the file.

filter = ["r/block/", "r/disk/", "r/sd.*/", "a/.*/"]

After updating /etc/lvm.conf, it's necessary to update the **initrd** so that this file will be copied there, where the filter matters the most, during boot.

Perform:

```
update-initramfs -u -k all
```

Note

Every time either /etc/lvm.conf or /etc/multipath.conf is updated, the initrd should be rebuilt to reflect these changes. This is imperative when denylists and filters are necessary to maintain a stable storage configuration.

Device Mapper Multipathing - Configuration

Device Mapper Multipath will be referred here as **multipath** only.

Before moving on with this session it is recommended that you read: 1. Device Mapper Multipathing - Introduction

Multipath is usually able to work out-of-the-box with most common storages. This doesn't mean the default configuration variables should be used in production: they don't treat important parameters your storage might need.

Consult your **storage manufacturer's install guide** for the Linux Multipath configuration options. It is very common that storage vendors provide the most adequate options for Linux, including minimal kernel and multipath-tools versions required.

Default configuration values for DM-Multipath can be overridden by editing the /etc/multipath.conf file and restarting the multipathd service.

This chapter provides information on parsing and modifying the multipath.conf file and it is split into the following configuration file sections:

- Configuration File Overview
- Configuration File Defaults
- Configuration File Blacklist & Exceptions
- Configuration File Multipath Section
- Configuration File Devices Section

Configuration File Overview

The configuration file contains entries of the form:

The following keywords are recognized:

- **defaults** This section defines default values for attributes which are used whenever no values are given in the appropriate device or multipath sections.
- blacklist This section defines which devices should be excluded from the multipath topology discovery.
- **blacklist_exceptions** This section defines which devices should be included in the multipath topology discovery, despite being listed in the blacklist section.
- **multipaths** This section defines the multipath topologies. They are indexed by a World Wide Identifier(WWID).Attributes set in this section take **precedence over all others**.
- devices This section defines the device-specific settings. Devices are identified by vendor, product, and revision.

• **overrides** - This section defines values for attributes that should override the device-specific settings for all devices.

Configuration File Defaults

Currently, the multipath configuration file **ONLY** includes a minor **defaults** section that sets the **user_friendly_names** parameter to **yes**:

```
defaults {
    user_friendly_names yes
}
```

}

This overwrites the default value of the user_friendly_names parameter.

All the multipath attributes that can set in the **defaults** section of the multipath.conf file can be found HERE with an explanation of what they mean. The attributes are:

- verbosity
- polling_interval
- max_polling_interval
- reassign_maps
- multipath_dir
- $\bullet \ \, path_selector$
- path_grouping_policy
- uid_attrs
- uid_attribute
- getuid_callout
- prio
- prio_args
- features
- $\bullet \ \, path_checker$
- alias_prefix
- failback
- rr_min_io
- rr_min_io_rq
- max_fds
- rr_weight
- no_path_retry
- $\bullet \ queue_without_daemon$
- checker_timeout
- flush_on_last_del
- user_friendly_names
- fast_io_fail_tmo
- dev_loss_tmo
- bindings_file
- wwids_file
- prkeys_file
- log_checker_err
- reservation_key
- all_tg_pt
- $\bullet \ retain_attached_hw_handler$
- detect_prio
- $\bullet \ \ detect_checker$
- force_sync
- strict_timing
- deferred_remove
- partition_delimiter
- config_dir
- san_path_err_threshold
- san_path_err_forget_rate
- san_path_err_recovery_time
- marginal_path_double_failed_time
- marginal_path_err_sample_time
- $\bullet \ marginal_path_err_rate_threshold$
- marginal_path_err_recheck_gap_time

- delay_watch_checks
- delay wait checks
- marginal pathgroups
- find_multipaths
- find multipaths timeout
- uxsock timeout
- retrigger_tries
- retrigger_delay
- missing_uev_wait_timeout
- skip kpartx
- disable_changed_wwids
- remove retries
- max sectors kb
- ghost delay
- enable foreign

Previously the multipath-tools project used to provide a complete configuration file with all the most used options for each of the most used storage devices. Currently you can see all those default options by executing sudo multipath -t. This will dump used configuration file including all the embedded default options.

Configuration File Blacklist & Exceptions

The blacklist section is used to exclude specific devices from the multipath topology. It is most commonly used to exclude local disks, non-multipathed OR non-disk devices.

1. Blacklist by devnode

The default blacklist consists of the regular expressions "^(ram|zram|raw|loop|fd|md|dm-|sr|scd|st|dcssblk)[0-9]" and "^(td|hd|vd)[a-z]". This causes virtual devices, non-disk devices, and some other device types to be excluded from multipath handling by default.

```
blacklist {
   devnode "^(ram|zram|raw|loop|fd|md|dm-|sr|scd|st|dcssblk)[0-9]"
   devnode "^(td|hd|vd)[a-z]"
   devnode "^cciss!c[0-9]d[0-9]*"
}
```

2. Blacklist by wwid

Regular expression for the World Wide Identifier of a device to be excluded/included

3. Blacklist by device

Subsection for the device description. This subsection recognizes the **vendor** and **product** keywords. Both are regular expressions.

device { vendor "LENOVO" product "Universal Xport"

4. Blacklist by property

}

Regular expression for an udev property. All devices that have matching udev properties will be excluded/included. The handling of the property keyword is special, because devices must have at least one whitelisted udev property; otherwise they're treated as blacklisted, and the message "blacklisted, udev property missing" is displayed in the logs.

5. Blacklist by protocol

The protocol strings that multipath recognizes are scsi:fcp, scsi:spi, scsi:ssa, scsi:sbp, scsi:srp, scsi:scsi, scsi:sas, scsi:adt, scsi:ata, scsi:unspec, ccw, cciss, nvme, and undef. The protocol that a path is using can be viewed by running multipathd show paths form at "%d $\%\mathrm{P}$ "

6. Blacklist Exceptions

The blacklist_exceptions section is used to revert the actions of the blacklist section. This allows one to selectively include ("whitelist") devices which would normally be excluded via the blacklist section.

```
blacklist_exceptions {
    property "(SCSI_IDENT_|ID_WWN)"
}
```

A common usage is to blacklist "everything" using a catch-all regular expression, and create specific blacklist_exceptions entries for those devices that should be handled by multipath-tools.

Configuration File Multipath Section

The multipaths section allows setting attributes of **multipath maps**. The attributes that are set via the **multipaths section** (see list below) take precedence over all other configuration settings, including those from the overrides section.

The only recognized attribute for the multipaths section is the multipath subsection. If there are multiple multipath subsections matching a given WWID, the contents of these sections are merged, and settings from later entries take precedence.

The multipath subsection recognizes the following attributes:

- **wwid** = (Mandatory) World Wide Identifier. Detected multipath maps are matched agains this attribute. Note that, unlike the wwid attribute in the blacklist section, this is not a regular expression or a substring; WWIDs must match exactly inside the multipaths section.
- **alias** = Symbolic name for the multipath map. This takes precedence over a an entry for the same WWID in the bindings file.

The following attributes are optional; if not set the default values are taken from the overrides, devices, or **defaults** section:

- path_grouping_policy
- path_selector
- prio
- prio_args
- failback
- rr_weight
- no_path_retry
- rr_min_io
- rr_min_io_rq
- flush_on_last_del
- features
- reservation_key
- user_friendly_names
- deferred_remove
- san_path_err_threshold
- san_path_err_forget_rate
- san path err recovery time
- marginal path err sample time
- marginal_path_err_rate_threshold
- marginal_path_err_recheck_gap_time
- marginal_path_double_failed_time
- delay watch checks
- delay wait checks
- skip_kpartx
- max sectors kb
- ghost delay

Example:

```
multipaths {
```

```
multipath {
    wwid
                             3600508b4000156d70001200000b0000
    alias
                             vellow
    path_grouping_policy
                             multibus
    path_selector
                             "round-robin 0"
    failback
                            manual
    rr_weight
                             priorities
    no_path_retry
                             5
}
multipath {
```

```
wwid 1DEC___321816758474
alias red
}
```

```
}
```

Configuration File Devices Section

multipath-tools have a built-in device table with reasonable defaults for more than 100 known multipath-capable storage devices. The devices section can be used to override these settings. If there are multiple matches for a given device, the attributes of all matching entries are applied to it. If an attribute is specified in several matching device subsections, later entries take precedence.

The only recognized attribute for the devices section is the device subsection. Devices detected in the system are matched against the device entries using the vendor, product, and revision fields.

The vendor, product, and revision fields that multipath or multipathd detect for devices in a system depend on the device type. For SCSI devices, they correspond to the respective fields of the SCSI INQUIRY page. In general, the command 'multipathd show paths format "%d %s"' command can be used to see the detected properties for all devices in the system.

The device subsection recognizes the following attributes:

- 1. vendor
 - (Mandatory) Regular expression to match the vendor name.
- 2. product
 - (Mandatory) Regular expression to match the product name.
- 3. revision
 - Regular expression to match the product revision.
- 4. product_blacklist
 - Products with the given vendor matching this string are blacklisted.
- 5. alias_prefix

The user_friendly_names prefix to use for this device type, instead of the default "mpath".

- 6. hardware_handler
 - The hardware handler to use for this device type. The following hardware handler are implemented:
 - **1 emc** (Hardware-dependent) Hardware handler for DGC class arrays as CLARiiON CX/AX and EMC VNX and Unity families.
 - 1 rdac (Hardware-dependent) Hardware handler for LSI / Engenio / NetApp RDAC class as NetApp SANtricity E/EF Series, and OEM arrays from IBM DELL SGI STK and SUN.
 - 1 hp_sw (Hardware-dependent) Hardware handler for HP/COMPAQ/DEC HSG80 and MSA/HSV arrays with Active/Standby mode exclusively.
 - 1 alua (Hardware-dependent) Hardware handler for SCSI-3 ALUA compatible arrays.
 - 1 ana (Hardware-dependent) Hardware handler for NVMe ANA compatible arrays.

The following attributes are optional; if not set the default values are taken from the defaults section:

- path_grouping_policy
- uid_attribute
- getuid_callout
- path_selector
- path_checker
- prio
- prio_args
- features
- failback
- rr_weight
- no_path_retry
- rr_min_io
- rr_min_io_rq
- fast_io_fail_tmo
- dev_loss_tmo
- flush_on_last_del
- user_friendly_names
- retain_attached_hw_handler
- detect_prio
- detect checker
- deferred_remove

```
• san_path_err_threshold
```

- san_path_err_forget_rate
- $\bullet \ \ san_path_err_recovery_time$
- marginal_path_err_sample_time
- marginal_path_err_rate_threshold
- marginal_path_err_recheck_gap_time
- marginal_path_double_failed_time
- delay_watch_checks
- delay_wait_checks
- skip_kpartx
- max_sectors_kb
- ghost_delay
- all_tg_pt

Example:

devices {

}

```
device {
    vendor "3PARdata"
    product "VV"
    path_grouping_policy "group_by_prio"
    hardware_handler "1 alua"
    prio "alua"
    failback "immediate"
    no path retry 18
    fast io fail tmo 10
    dev loss tmo "infinity"
}
device {
    vendor "DEC"
    product "HSG80"
    path_grouping_policy "group_by_prio"
    path_checker "hp_sw"
    hardware handler "1 hp sw"
    prio "hp sw"
    no_path_retry "queue"
}
```

Device Mapper Multipathing - Setup

Device Mapper Multipath will be referred here as **multipath** only.

Before moving on with this session it is recommended that you read:

- 1. Device Mapper Multipathing Introduction
- 2. Device Mapper Multipathing Configuration

This section provides step-by-step example procedures for configuring multipath.

It includes the following procedures:

- Basic setup
 - Main **defaults** & **devices** attributes.
 - Shows how to **ignore disks** with blacklists
 - Shows how to **rename disks** using WWIDs
- Configuring active/active paths

Basic Setup

Before setting up multipath on your system, ensure that your system has been updated and includes the **multipath-tools** package. If boot from SAN is desired, then the **multipath-tools-boot** package is also required.

A very simple /etc/multipath.conf file exists, as explained in Device Mapper Multipathing - Configuration session. All the non-declared in multipath.conf attributes are taken from multipath-tools internal database and its internal blacklist.

The internal attributes database can be acquired by doing:

\$ sudo multipath -t

Multipath is usually able to work out-of-the-box with most common storages. This **does not** mean the default configuration variables should be **used in production**: they don't treat important parameters your storage might need.

With the **internal attributes**, described above, and the given example bellow, you will likely be able to create your /etc/multipath.conf file by squashing the code blocks bellow. Make sure to read every **defaults** section attribute comments and change it based on your environment needs.

• Example of a **defaults** section:

```
defaults {
    #
    # name
              : polling_interval
             : multipathd
    # scope
    # desc
              : interval between two path checks in seconds. For
    #
                properly functioning paths, the interval between checks
    #
                will gradually increase to (4 * polling_interval).
    # values : n > 0
    # default : 5
    #
    polling_interval 10
    #
    # name
              : path_selector
    # scope
              : multipath & multipathd
              : the default path selector algorithm to use
    # desc
                these algorithms are offered by the kernel multipath target
    #
     values : "round-robin 0" = Loop through every path in the path group,
    #
                                   sending the same amount of IO to each.
    #
    #
                "queue-length 0" = Send the next bunch of IO down the path
    #
                                   with the least amount of outstanding IO.
    #
                "service-time 0" = Choose the path for the next bunch of IO
    #
                                   based on the amount of outstanding IO to
    #
                                   the path and its relative throughput.
    # default : "service-time 0"
    #
    path_selector "round-robin 0"
    #
    # name
              : path_grouping_policy
              : multipath & multipathd
    # scope
    # desc
              : the default path grouping policy to apply to unspecified
    #
                multipaths
     values : failover
    #
                                   = 1 path per priority group
                                   = all valid paths in 1 priority group
    #
                multibus
                                   = 1 priority group per detected serial
    #
                group_by_serial
    #
                                     number
    #
                group_by_prio
                                   = 1 priority group per path priority
    #
                                     value
                group_by_node_name = 1 priority group per target node name
    #
    # default : failover
    #
    path grouping policy multibus
    #
    # name
             : uid_attribute
    # scope : multipath & multipathd
    # desc
              : the default udev attribute from which the path
    #
            identifier should be generated.
    # default : ID_SERIAL
```

```
uid_attribute "ID_SERIAL"
#
# name
        : getuid_callout
# scope : multipath & multipathd
         : the default program and args to callout to obtain a unique
# desc
           path identifier. This parameter is deprecated.
#
#
           This parameter is deprecated, superseded by uid_attribute
# default : /lib/udev/scsi_id --whitelisted --device=/dev/%n
#
getuid_callout "/lib/udev/scsi_id --whitelisted --device=/dev/%n"
#
# name
         : prio
# scope : multipath & multipathd
         : the default function to call to obtain a path
# desc
#
           priority value. The ALUA bits in SPC-3 provide an
#
           exploitable prio value for example.
# default : const
#
# prio "alua"
#
# name
         : prio args
# scope : multipath & multipathd
# desc : The arguments string passed to the prio function
#
           Most prio functions do not need arguments. The
#
       datacore prioritizer need one.
# default : (null)
#
# prio_args "timeout=1000 preferredsds=foo"
#
         : features
# name
# scope : multipath & multipathd
        : The default extra features of multipath devices.
# desc
           Syntax is "num[ feature_0 feature_1 ...]", where `num' is the
#
           number of features in the following (possibly empty) list of
#
#
           features.
# values : queue_if_no_path = Queue IO if no path is active; consider
#
                               using the `no_path_retry' keyword instead.
#
           no_partitions
                             = Disable automatic partitions generation via
                               kpartx.
#
# default : "0"
#
           "0"
features
           "1 queue if no path"
#features
            "1 no_partitions"
#features
#features
           "2 queue_if_no_path no_partitions"
#
        : path_checker, checker
# name
# scope : multipath & multipathd
# desc
         : the default method used to determine the paths' state
# values : readsector0|tur|emc_clariion|hp_sw|directio|rdac|cciss_tur
# default : directio
#
path_checker directio
#
# name
          : rr_min_io
```

```
171
```

scope

: multipath & multipathd

```
# desc
       : the number of IO to route to a path before switching
#
            to the next in the same path group for the bio-based
#
           multipath implementation. This parameter is used for
#
           kernels version up to 2.6.31; newer kernel version
#
           use the parameter rr_min_io_rq
# default : 1000
#
rr_min_io 100
#
# name
         : rr_min_io_rq
# scope : multipath & multipathd
        : the number of IO to route to a path before switching
# desc
#
           to the next in the same path group for the request-based
#
           multipath implementation. This parameter is used for
           kernels versions later than 2.6.31.
#
# default : 1
rr_min_io_rq 1
#
# name
        : flush_on_last_del
# scope : multipathd
         : If set to "yes", multipathd will disable queueing when the
# desc
#
           last path to a device has been deleted.
# values : yes|no
# default : no
#
flush_on_last_del yes
#
# name
         : max_fds
# scope : multipathd
       : Sets the maximum number of open file descriptors for the
# desc
#
           multipathd process.
# values : max|n > 0
# default : None
#
max_fds 8192
#
# name
         : rr_weight
# scope : multipath & multipathd
# desc : if set to priorities the multipath configurator will assign
#
           path weights as "path prio * rr min io"
# values : priorities|uniform
# default : uniform
#
rr_weight priorities
#
       : failback
# name
# scope : multipathd
         : tell the daemon to manage path group failback, or not to.
# desc
           0 means immediate failback, values >0 means deffered
#
           failback expressed in seconds.
#
# values : manual|immediate|n > 0
# default : manual
#
failback immediate
#
# name
          : no_path_retry
```

```
172
```

```
# scope : multipath & multipathd
        : tell the number of retries until disable queueing, or
# desc
#
            "fail" means immediate failure (no queueing),
#
           "queue" means never stop queueing
# values : queue|fail|n (>0)
# default : (null)
#
no_path_retry fail
#
# name
         : queue_without_daemon
# scope : multipathd
       : If set to "no", multipathd will disable queueing for all
# desc
#
           devices when it is shut down.
# values : yes|no
# default : yes
queue_without_daemon no
#
# name
       : user_friendly_names
# scope : multipath & multipathd
       : If set to "yes", using the bindings file
# desc
           /etc/multipath/bindings to assign a persistent and
#
#
           unique alias to the multipath, in the form of mpath<n>.
#
           If set to "no" use the WWID as the alias. In either case
#
           this be will be overriden by any specific aliases in this
#
           file.
# values : yes|no
# default : no
user_friendly_names yes
#
         : mode
# name
# scope : multipath & multipathd
# desc : The mode to use for the multipath device nodes, in octal.
# values : 0000 - 0777
# default : determined by the process
mode 0644
#
# name
         : uid
# scope : multipath & multipathd
# desc : The user id to use for the multipath device nodes. You
#
           may use either the numeric or symbolic uid
# values : <user id>
# default : determined by the process
uid 0
#
# name
        : gid
# scope : multipath & multipathd
       : The group id to user for the multipath device nodes. You
# desc
           may use either the numeric or symbolic gid
#
# values : <group_id>
# default : determined by the process
gid disk
#
# name
        : checker_timeout
# scope : multipath & multipathd
# desc
         : The timeout to use for path checkers and prioritizers
#
           that issue scsi commands with an explicit timeout, in
#
           seconds.
```

```
# values : n > 0
# default : taken from /sys/block/sd<x>/device/timeout
checker_timeout 60
#
# name : fast io fail tmo
# scope : multipath & multipathd
# desc : The number of seconds the scsi layer will wait after a
#
           problem has been detected on a FC remote port before failing
#
           IO to devices on that remote port.
# values : off | n >= 0 (smaller than dev_loss_tmo)
# default : determined by the OS
fast io fail tmo 5
#
       : dev_loss_tmo
# name
# scope : multipath & multipathd
# desc : The number of seconds the scsi layer will wait after a
#
         problem has been detected on a FC remote port before
#
          removing it from the system.
# values : infinity | n > 0
# default : determined by the OS
dev_loss_tmo 120
#
# name
       : bindings_file
# scope : multipath
# desc : The location of the bindings file that is used with
#
           the user_friendly_names option.
# values : <full_pathname>
# default : "/var/lib/multipath/bindings"
# bindings_file "/etc/multipath/bindings"
#
# name : wwids_file
# scope : multipath
# desc : The location of the wwids file multipath uses to
#
           keep track of the created multipath devices.
# values : <full_pathname>
# default : "/var/lib/multipath/wwids"
# wwids_file "/etc/multipath/wwids"
#
# name : reservation_key
# scope : multipath
# desc
         : Service action reservation key used by mpathpersist.
# values : <key>
# default : (null)
# reservation_key "mpathkey"
#
# name : force sync
# scope : multipathd
        : If set to yes, multipath will run all of the checkers in
# desc
#
           sync mode, even if the checker has an async mode.
# values : yes|no
# default : no
force_sync yes
#
# name
       : config_dir
# scope : multipath & multipathd
# desc
         : If not set to an empty string, multipath will search
```

```
this directory alphabetically for files ending in ".conf"
   #
   #
              and it will read configuration information from these
   #
              files, just as if it was in /etc/multipath.conf
   # values : "" or a fully qualified pathname
   # default : "/etc/multipath/conf.d"
   #
   # name
            : delay_watch_checks
   # scope : multipathd
   # desc
           : If set to a value greater than 0, multipathd will watch
              paths that have recently become valid for this many
   #
   #
              checks. If they fail again while they are being watched,
   #
              when they next become valid, they will not be used until
   #
              they have stayed up for delay wait checks checks.
   # values : no|<n> > 0
   # default : no
   delay_watch_checks 12
   #
   # name
            : delay_wait_checks
   # scope : multipathd
   # desc
           : If set to a value greater than 0, when a device that has
              recently come back online fails again within
   #
              delay_watch_checks checks, the next time it comes back
   #
   #
              online, it will marked and delayed, and not used until
              it has passed delay_wait_checks checks.
   #
   # values : no|<n> > 0
   # default : no
   delay_wait_checks 12
}
  • Example of a multipaths section.
    Note: You can obtain the WWIDs for your LUNs executing:
    $ multipath -ll
    after the service multipath-tools.service has been restarted.
multipaths {
   multipath {
       alias yellow
   }
   multipath {
       alias blue
   }
   multipath {
       alias red
   }
   multipath {
       alias green
   }
   multipath {
       alias purple
   }
}
  • Small example of a devices section:
# devices {
```

```
# device {
```

```
vendor "IBM"
#
#
          product "2107900"
#
          path_grouping_policy group_by_serial
#
      }
# }
#
   • Example of a blacklist section:
# name
          : blacklist
  scope
         : multipath & multipathd
#
          : list of device names to discard as not multipath candidates
#
  desc
#
# Devices can be identified by their device node name "devnode",
# their WWID "wwid", or their vender and product strings "device"
# default : fd, hd, md, dm, sr, scd, st, ram, raw, loop, dcssblk
# blacklist {
#
      wwid 26353900f02796769
#
      devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]\*"
#
      devnode "^hd[a-z]"
#
      devnode "^dcssblk[0-9]\*"
#
      device {
#
          vendor DEC.\*
#
          product MSA[15]00
#
      }
# }
   • Example of a blacklist exception section:
# name
          : blacklist exceptions
# scope
         : multipath & multipathd
          : list of device names to be treated as multipath candidates
#
 desc
            even if they are on the blacklist.
#
#
# Note: blacklist exceptions are only valid in the same class.
# It is not possible to blacklist devices using the devnode keyword
# and to exclude some devices of them using the wwid keyword.
# default : -
#
# blacklist_exceptions {
#
         devnode "^dasd[c-d]+[0-9]\*"
#
         wwid
               "IBM.75000000092461.4d00.34"
#
         wwid
                 "TBM. 75000000092461.4d00.35"
#
                "IBM.7500000092461.4d00.36"
         wwid
# }
```

Device Mapper Multipathing - Usage & Debug

Device Mapper Multipath will be referred here as **multipath** only.

Before moving on with this session it is recommended that you read:

- 1. Device Mapper Multipathing Introduction
- 2. Device Mapper Multipathing Configuration
- 3. Device Mapper Multipathing Setup

This section provides step-by-step example procedures for configuring multipath.

It includes the following procedures:

- **Resizing Online** Multipath Devices
- Moving root File System from a Single Path Device to a Multipath Device $% \mathcal{A}$
- The Multipath Daemon
- Issues with queue_if_no_path

- Multipath Command Output
- Multipath **Queries **with multipath Command
- Determining Device Mapper Entries with dmsetup Command
- Troubleshooting with the multipathd interactive console

Resizing Online Multipath Devices

First find all the paths to the LUN about to be resized:

```
$ sudo multipath -ll
mpathb (360014056eee8ec6e1164fcb959086482) dm-0 LIO-ORG,lun01
size=1.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:1 sde 8:64 active ready running
`-+- policy='service-time 0' prio=50 status=enabled
    `- 8:0:0:1 sdf 8:80 active ready running
mpatha (36001405e3c2841430ee4bf3871b1998b) dm-1 LIO-ORG,lun02
size=1.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:2 sdc 8:32 active ready running
`-+- policy='service-time 0' prio=50 status=enabled
    `- 8:0:0:2 sdd 8:48 active ready running
```

Now I'll reconfigure **mpathb** (with wwid = 360014056eee8ec6e1164fcb959086482) to have 2GB instead of just 1Gb and check if has changed:

```
$ echo 1 | sudo tee /sys/block/sde/device/rescan
1
$ echo 1 | sudo tee /sys/block/sdf/device/rescan
1
$ sudo multipath -ll
mpathb (360014056eee8ec6e1164fcb959086482) dm-0 LIO-ORG,lun01
size=1.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:1 sde 8:64 active ready running
`-+- policy='service-time 0' prio=50 status=enabled
  `- 8:0:0:1 sdf 8:80 active ready running
mpatha (36001405e3c2841430ee4bf3871b1998b) dm-1 LIO-ORG,lun02
size=1.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:2 sdc 8:32 active ready running
`-+- policy='service-time 0' prio=50 status=enabled
  `- 8:0:0:2 sdd 8:48 active ready running
```

Not yet! We still need to re-scan the multipath map:

```
$ sudo multipathd resize map mpathb
ok
```

And **then** we are good:

```
$ sudo multipath -ll
mpathb (360014056eee8ec6e1164fcb959086482) dm-0 LIO-ORG,lun01
size=2.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:1 sde 8:64 active ready running
.`-+- policy='service-time 0' prio=50 status=enabled
    `- 8:0:0:1 sdf 8:80 active ready running
mpatha (36001405e3c2841430ee4bf3871b1998b) dm-1 LIO-ORG,lun02
size=1.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:2 sdc 8:32 active ready running
`-+- policy='service-time 0' prio=50 status=enabled
    `- 8:0:0:2 sdd 8:48 active ready running
```

Make sure to run <code>resize2fs /dev/mapper/mpathb</code> to resize the filesystem.

Moving root File System from a Single Path Device to a Multipath Device

This is dramatically simplified by the use of UUIDs to identify devices as an intrinsic label. Simply install multipathtools-boot and reboot. This will rebuild the initial ramdisk and afford multipath the opportunity to build it's paths before the root filesystem is mounted by UUID.

Note

Whenever multipath.conf is updated, so should the initrd by executing: update-initramfs -u -k all The reason behind is multipath.conf is copied to the ramdisk and is integral to determining the available devices to map via it's denylist and devices sections.

The Multipath Daemon

If you find you have trouble implementing a multipath configuration, you should ensure the multipath daemon is running as described in Device Mapper Multipathing - Setup. The **multipathd** daemon must be running in order to use** multipath** devices.

Multipath Command Output

When you create, modify, or list a multipath device, you get a printout of the current device setup. The format is as follows. For each multipath device:

action_if_any: alias (wwid_if_different_from_alias) dm_device_name_if_known vendor,product size=size features='features' hwhandler='hardware handler' wp=write permission if known

For each path group:

```
-+- policy='scheduling_policy' prio=prio_if_known
status=path_group_status_if_known
```

For each path:

```
`- host:channel:id:lun devnode major:minor dm_status_if_known path_status online_status
```

For example, the output of a multipath command might appear as follows:

```
mpathb (360014056eee8ec6e1164fcb959086482) dm-0 LIO-ORG,lun01
size=2.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=50 status=active
| `- 7:0:0:1 sde 8:64 active ready running
`-+- policy='service-time 0' prio=50 status=enabled
   ` 8:0:0:1 sdf 8:80 active ready running
```

If the path is up and ready for I/O, the status of the path is **ready** or **ghost**. If the path is down, the status is **faulty** or **shaky**. The path status is updated periodically by the **multipathd** daemon based on the *polling interval* defined in the /etc/multipath.conf file.

The dm_status is similar to the path status, but from the kernel's point of view. The dm_status has two states: failed, which is analogous to faulty, and active, which covers all other path states. Occasionally, the path state and the dm state of a device will temporary not agree.

The possible values for **online__status** are **running** and **offline**. A status of **offline** means that the SCSI device has been disabled.

Multipath **Queries **with multipath Command

You can use the -l and -ll options of the multipath command to display the current multipath configuration. The -l option displays multipath topology gathered from information in sysfs and the device mapper. The -ll option displays the information the -l displays in addition to all other available components of the system.

When displaying the multipath configuration, there are three verbosity levels you can specify with the -v option of the multipath command. Specifying -v0 yields no output. Specifying^{**}-v1^{**} outputs the created or updated multipath names only, which you can then feed to other tools such as kpartx. Specifying -v2 prints all detected paths, multipaths, and device maps.

Note

The default **verbosity** level of multipath is **2** and can be globally modified by defining the **verbosity attribute** in the **defaults** section of multipath.conf.

The following example shows the output of a **multipath** -l command.

```
$ sudo multipath -l
mpathb (360014056eee8ec6e1164fcb959086482) dm-0 LIO-ORG,lun01
size=2.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=0 status=active
| `- 7:0:0:1 sde 8:64 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
    `- 8:0:0:1 sdf 8:80 active undef running
mpatha (36001405e3c2841430ee4bf3871b1998b) dm-1 LIO-ORG,lun02
size=1.0G features='0' hwhandler='1 alua' wp=rw
|-+- policy='service-time 0' prio=0 status=active
| `- 7:0:0:2 sdc 8:32 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
    `- 8:0:0:2 sdd 8:48 active undef running
```

Determining Device Mapper Entries with dmsetup Command

You can use the dmsetup command to find out which device mapper entries match the **multipathed** devices. The following command displays all the device mapper devices and their major and minor numbers. The minor numbers determine the name of the dm device. For example, a minor number of 1 corresponds to the multipath device /dev/dm-1.

```
$ sudo dmsetup ls
mpathb (253:0)
mpatha (253:1)

$ ls -lahd /dev/dm*
brw-rw---- 1 root disk 253, 0 Apr 27 14:49 /dev/dm-0
brw-rw---- 1 root disk 253, 1 Apr 27 14:47 /dev/dm-1
```

Troubleshooting with the multipathd interactive console

The **multipathd** -k command is an interactive interface to the **multipathd** daemon. Entering this command brings up an interactive multipath console. After entering this command, you can enter help to get a list of available commands, you can enter a interactive command, or you can enter **CTRL-D** to quit.

The multipath interactive console can be used to troubleshoot problems you may be having with your system. For example, the following command sequence displays the multipath configuration, including the defaults, before exiting the console.

```
$ sudo multipathd -k
> show config
> CTRL-D
```

The following command sequence ensures that multipath has picked up any changes to the multipath.conf,

```
$ sudo multipathd -k
> reconfigure
> CTRL-D
```

Use the following command sequence to ensure that the path checker is working properly.

```
$ sudo multipathd -k
> show paths
> CTRL-D
```

Commands can also be streamed into multipathd using STDIN like so:

```
$ echo 'show config' | sudo multipathd -k
```

Security

Security should always be considered when installing, deploying, and using any type of computer system. Although a fresh installation of Ubuntu is relatively safe for immediate use on the Internet, it is important to have a balanced understanding of your system's security posture based on how it will be used after deployment.

This chapter provides an overview of security-related topics as they pertain to Ubuntu Server Edition, and outlines simple measures you may use to protect your server and network from any number of potential security threats.

If instead of security related administration and concepts for Ubuntu Server please be aware that there is more. Depending on what you were looking for please consider those references:

- Further information about security at Ubuntu, have a look at Ubuntu Security
- Information about known vulnerabilities:
 - per CVE check out the CVE overview
 - per Package have a look at the Ubuntu Security Notices
- Reporting a security issue, have a look at the disclosure policy

User Management

User management is a critical part of maintaining a secure system. Ineffective user and privilege management often lead many systems into being compromised. Therefore, it is important that you understand how you can protect your server through simple and effective user account management techniques.

Where is root?

Ubuntu developers made a conscientious decision to disable the administrative root account by default in all Ubuntu installations. This does not mean that the root account has been deleted or that it may not be accessed. It merely has been given a password hash which matches no possible value, therefore may not log in directly by itself.

Instead, users are encouraged to make use of a tool by the name of 'sudo' to carry out system administrative duties. Sudo allows an authorized user to temporarily elevate their privileges using their own password instead of having to know the password belonging to the root account. This simple yet effective methodology provides accountability for all user actions, and gives the administrator granular control over which actions a user can perform with said privileges.

• If for some reason you wish to enable the root account, simply give it a password:

sudo passwd

Sudo will prompt you for your password, and then ask you to supply a new password for root as shown below:

[sudo] password for username: (enter your own password) Enter new UNIX password: (enter a new password for root) Retype new UNIX password: (repeat new password for root) passwd: password updated successfully

• To disable the root account password, use the following passwd syntax:

sudo passwd -l root

• You should read more on Sudo by reading the man page:

man sudo

By default, the initial user created by the Ubuntu installer is a member of the group sudo which is added to the file /etc/sudoers as an authorized sudo user. If you wish to give any other account full root access through sudo, simply add them to the sudo group.

Adding and Deleting Users

The process for managing local users and groups is straightforward and differs very little from most other GNU/Linux operating systems. Ubuntu and other Debian based distributions encourage the use of the 'adduser' package for account management.

• To add a user account, use the following syntax, and follow the prompts to give the account a password and identifiable characteristics, such as a full name, phone number, etc.

sudo adduser username

• To delete a user account and its primary group, use the following syntax:

sudo deluser username

Deleting an account does not remove their respective home folder. It is up to you whether or not you wish to delete the folder manually or keep it according to your desired retention policies.

Remember, any user added later on with the same UID/GID as the previous owner will now have access to this folder if you have not taken the necessary precautions.
You may want to change these UID/GID values to something more appropriate, such as the root account, and perhaps even relocate the folder to avoid future conflicts:

sudo chown -R root:root /home/username/ sudo mkdir /home/archived_users/ sudo mv /home/username /home/archived_users/

• To temporarily lock or unlock a user password, use the following syntax, respectively:

sudo passwd -l username sudo passwd -u username

• To add or delete a personalized group, use the following syntax, respectively:

sudo addgroup groupname
sudo delgroup groupname

• To add a user to a group, use the following syntax:

sudo adduser username groupname

User Profile Security

When a new user is created, the adduser utility creates a brand new home directory named /home/username. The default profile is modeled after the contents found in the directory of /etc/skel, which includes all profile basics.

If your server will be home to multiple users, you should pay close attention to the user home directory permissions to ensure confidentiality. By default, user home directories in Ubuntu are created with world read/execute permissions. This means that all users can browse and access the contents of other users home directories. This may not be suitable for your environment.

• To verify your current user home directory permissions, use the following syntax:

ls -ld /home/username

The following output shows that the directory /home/username has world-readable permissions:

drwxr-xr-x 2 username username 4096 2007-10-02 20:03 username

• You can remove the world readable-permissions using the following syntax:

sudo chmod 0750 /home/username

Note

Some people tend to use the recursive option (-R) indiscriminately which modifies all child folders and files, but this is not necessary, and may yield other undesirable results. The parent directory alone is sufficient for preventing unauthorized access to anything below the parent.

A much more efficient approach to the matter would be to modify the adduser global default permissions when creating user home folders. Simply edit the file /etc/adduser.conf and modify the DIR_MODE variable to something appropriate, so that all new home directories will receive the correct permissions.

DIR_MODE=0750

• After correcting the directory permissions using any of the previously mentioned techniques, verify the results using the following syntax:

ls -ld /home/username

The results below show that world-readable permissions have been removed:

drwxr-x--- 2 username username 4096 2007-10-02 20:03 username

Password Policy

A strong password policy is one of the most important aspects of your security posture. Many successful security breaches involve simple brute force and dictionary attacks against weak passwords. If you intend to offer any form of remote access involving your local password system, make sure you adequately address minimum password complexity requirements, maximum password lifetimes, and frequent audits of your authentication systems.

Minimum Password Length

By default, Ubuntu requires a minimum password length of 6 characters, as well as some basic entropy checks. These values are controlled in the file /etc/pam.d/common-password, which is outlined below.

password [success=1 default=ignore] pam_unix.so obscure sha512

If you would like to adjust the minimum length to 8 characters, change the appropriate variable to min=8. The modification is outlined below.

password [success=1 default=ignore] pam_unix.so obscure sha512 minlen=8

Note

Basic password entropy checks and minimum length rules do not apply to the administrator using sudo level commands to setup a new user.

Password Expiration

When creating user accounts, you should make it a policy to have a minimum and maximum password age forcing users to change their passwords when they expire.

- To easily view the current status of a user account, use the following syntax:
 - sudo chage -l username

The output below shows interesting facts about the user account, namely that there are no policies applied:

Last password change	:	Jan 20, 2015
Password expires	:	never
Password inactive	:	never
Account expires	:	never
Minimum number of days between password change	:	0
Maximum number of days between password change	:	99999
Number of days of warning before password expires	:	7

• To set any of these values, simply use the following syntax, and follow the interactive prompts:

sudo chage username

The following is also an example of how you can manually change the explicit expiration date (-E) to 01/31/2015, minimum password age (-m) of 5 days, maximum password age (-M) of 90 days, inactivity period (-I) of 30 days after password expiration, and a warning time period (-W) of 14 days before password expiration:

sudo chage -E 01/31/2015 -m 5 -M 90 -I 30 -W 14 username

• To verify changes, use the same syntax as mentioned previously:

sudo chage -l username

The output below shows the new policies that have been established for the account:

Last password change	:	Jan 20, 2015
Password expires	:	Apr 19, 2015
Password inactive	:	May 19, 2015
Account expires	:	Jan 31, 2015
Minimum number of days between password change	:	5
Maximum number of days between password change	:	90
Number of days of warning before password expires	:	14

Other Security Considerations

Many applications use alternate authentication mechanisms that can be easily overlooked by even experienced system administrators. Therefore, it is important to understand and control how users authenticate and gain access to services and applications on your server.

SSH Access by Disabled Users

Simply disabling/locking a user password will not prevent a user from logging into your server remotely if they have previously set up SSH public key authentication. They will still be able to gain shell access to the server, without the need for any password. Remember to check the users home directory for files that will allow for this type of authenticated SSH access, e.g. /home/username/.ssh/authorized_keys.

Remove or rename the directory .ssh/ in the user's home folder to prevent further SSH authentication capabilities.

Be sure to check for any established SSH connections by the disabled user, as it is possible they may have existing inbound or outbound connections. Kill any that are found.

who | grep username (to get the pts/# terminal)
sudo pkill -f pts/#

Restrict SSH access to only user accounts that should have it. For example, you may create a group called "sshlogin" and add the group name as the value associated with the AllowGroups variable located in the file /etc/ssh/sshd_config.

AllowGroups sshlogin

Then add your permitted SSH users to the group "sshlogin", and restart the SSH service.

sudo adduser username sshlogin
sudo systemctl restart sshd.service

External User Database Authentication

Most enterprise networks require centralized authentication and access controls for all system resources. If you have configured your server to authenticate users against external databases, be sure to disable the user accounts both externally and locally. This way you ensure that local fallback authentication is not possible.

Smart card authentication

Among some of the popular uses for smart cards is the ability to control access to computer systems. To operate the owner must have the smart card and they must know the PIN to unlock the card. This provides a higher degree of security than single-factor authentication such as just using a password.

The following sections describe how to enable smart card authentication on Ubuntu. They apply to Ubuntu 18.04 and 20.04.

Requirements

Software

The following packages must be installed to obtain a smart card configuration on Ubuntu.

- pcscd: contains the drivers needed to communicate with the CCID smart card readers.
- opensc-pkcs11: contains the smart card drivers, such as PIV or CAC.
- libpam-pkcs11: contains the PAM module to allow X.509 certificate logins via smart cards.

To install:

```
$ sudo apt update
$ sudo apt install opensc-pkcs11 libpam-pkcs11 pcscd
```

Hardware

Any PIV or CAC smart card with the corresponding reader should be sufficient. USB smart cards like Yubikey embed the reader, and work like regular PIV cards.

Each smart card is expected to contain an X.509 certificate and the corresponding private key to be used for authentication.

PAM configuration

The pam_pkcs11 module allows PAM supported systems to use X.509 certificates to authenticate logins. The module relies on a PKCS#11 library, such as opensc-pkcs11 to access the smart card for the credentials it will need.

When enabled, the pam_pkcs11 login process is as follows:

- 1. Enter login
- 2. Enter PIN
- 3. Validate the X.509 certificate
- 4. Map the certificate to a user
- 5. Verify the login and match

To enable that process we have to configure the pam_pkcs11 module and add the relevant certificate authorities, add pam_pkcs11 to PAM configuration and set the mapping of certificate names to logins.

Configure the pam_pkcs11 module

\$ cd /etc/pam_pkcs11

\$ sudo cp /usr/share/doc/libpam-pkcs11/examples/pam_pkcs11.conf.example pkcs11.conf

Check the module, cert_policy, and use_pkcs11_module options defined within the pkcs11_module opensc {} entry in the pam_pkcs11.conf file. The module option should contain the absolute path of the <code>open-pkcs11.so</code> on the system. The cert_policy option should include oscp as one of its certificate verification policies.

In particular it should contain the following lines in Ubuntu 20.04.

```
use_pkcsll_module = opensc;
module = /usr/lib/x86_64-linux-gnu/opensc-pkcsll.so;
cert_policy = ca,signature,oscp_on;
```

Leave debug = true until everything is setup and is operating as desired.

Map certificate names to login

This PAM module allows certificates to be used for login, though our Linux system needs to know the username. The pam_pkcs11 module provides a variety of cert mappers to do this. Each cert mapper uses specific information from the certificate to map to a user on the system. The different cert mappers may even be stacked. In other words, if the first defined mapper fails to map to a user on the system, the next one will be tried, and so on until a user is found.

For the purposes of this guide, we will use the pwent mapper. This mapper uses the getpwent() system call to examine the pw_name and pw_gecos fields of every user for a match to the CN name. If either matches, the pw_name is returned as the login name. Next, it matches this result to the PAM login name to determine if a match was found or not. Set pwent as the mapper in the pam_pkcsll.conf file by modifying the existing entry:

use_mappers = pwent;

Set the Certificate Authority and CRLs

To validate the smart card certificates the pam_pkcs11 module needs to know the acceptable Certificate Authorities for signing user certificates and any available CRLs. You can add these in the following paths.

- Certificate Authorities: /etc/pam_pkcsll/cacerts
- CRLs: /etc/pam_pkcs11/crls

Assuming the Certificate Authority is in ca.crt, the following example sets it up.

```
$ sudo mkdir -p /etc/pam_pkcs11/cacerts
$ sudo cp ca.crt /etc/pam_pkcs11/cacerts
$ cd /etc/pam_pkcs11/cacerts
$ sudo pkcs11_make_hash_link
```

Similarly for the CRLs.

Add pam_pkcs11 to PAM

The next step includes the pam_pkcs11 module into the PAM stack. There are various ways to do this depending on your local policy. The following example enables smart card support for general authentication.

Edit /etc/pam.d/common-auth to include the pam_pkcs11 module as follows.

```
# require pkcsll smart card login
auth [success=2 default=ignore] pam_pkcsll.so
```

The above configuration will require the system to perform a smart card authentication only. If a user fails to authenticate with a smart card, then the login will fail. All the PAM services in the /etc/pam.d directory that include common-auth will require the smart card authentication.

Warning: A global configuration such as this requires a smart card for su and sudo authentication as well!

Configure the pwent mapper

Now that pam_pkcs11 and PAM have been configured for certificate logins, there is one more action. The pwent mapper requires the CN in the certificate to be in the /etc/passwd gecos field of the user. The CN must be extracted from the certificate on the smart card and added in passwd.

\$ sudo apt install gnutls-bin
\$ plltool --list-tokens

The command above will show all the available smart cards in the system and its associated PKCS#11 URI. Copy the URI of selected card in the following command.

This command will print all certificates that can be used for authentication and their associated PKCS#11 URI.

\$ p11tool --login --list-certs [TOKEN-URI]

Now, once the URI of the certificate that will be used for authentication is known, let's extract the Common Name from the certificate. In the example we are assuming that our certificate URI is pkcsl1:id=04.

\$ p11tool --login --export 'pkcs11:id=04' | openssl x509 -noout -subject subject=CN = PIVKey BA366DFE3722C7449EC906B9274C8BAC

The CN is 'PIVKey BA366DFE3722C7449EC906B9274C8BAC'. Edit the /etc/passwd file and add this CN to the gecos field of the user the certificate belongs to.

\$ sudo usermod -c "PIVKey BA366DFE3722C7449EC906B9274C8BAC" foo

The OS is now ready to do a smart card login for the user foo.

SSH authentication

See this page on SSH authentication with smart cards.

Smart card authentication with SSH

One of the authentication methods supported by the SSH protocol is public key authentication. A public key is copied to the SSH server where it is stored and marked as authorized. The owner of the corresponding private key in the smart card can then SSH login to the server.

We will use <code>opensc-pkcs11</code> on the client to access the smart card drivers, and we will copy the public key from the smart card to the SSH server to make the authentication work.

The following instructions apply to Ubuntu 18.04 later.

Server configuration

The SSH server and client must be configured to permit smart card authentication.

Configure the SSH server

The SSH server needs to allow public key authentication set in its configuration file and it needs the user's public key.

Ensure the server has the PubkeyAuthentication option set to 'yes' in its /etc/ssh/sshd_config file. In a default /etc/ssh/sshd config in Ubuntu, the

PubkeyAuthentication option is commented out. However, the default is 'yes'. To ensure the setting, edit the sshd_config file and set accordingly.

PubkeyAuthentication yes

Restart the SSH server

 ${\tt sudo \ systemctl \ restart \ sshd}$

Set the public key on the server

Extract the user's public key from the smart card on the SSH client. Use sshkeygen to read the public key from the smart card and into a format consumable for SSH.

ssh-keygen -D /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so > smartcard.pub

Copy this key to the SSH server.

ssh-copy-id -f -i smartcard.pub ubuntu@server-2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "smartcard.pub"
ubuntu@server-2's password:
Number of key(s) added: 1
Now try logging into the machine, with: "ssh 'ubuntu@server-2'"
and check to make sure that only the key(s) you wanted were added.

Client configuration

The SSH client needs to identify its PKCS#11 provider. To do that set the PKCS11Provider option in the $\sim/.ssh/config$ file of each user desiring to use SSH smart card login.

PKCS11Provider /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so

Use this method to enforce SSH smart card login on a per user basis.

After this step you can SSH into the server using the smart card for authentication.

AppArmor

AppArmor is a Linux Security Module implementation of name-based mandatory access controls. AppArmor confines individual programs to a set of listed files and posix 1003.1e draft capabilities.

AppArmor is installed and loaded by default. It uses *profiles* of an application to determine what files and permissions the application requires. Some packages will install their own profiles, and additional profiles can be found in the apparmor-profiles package.

To install the apparmor-profiles package from a terminal prompt:

sudo apt install apparmor-profiles

AppArmor profiles have two modes of execution:

- Complaining/Learning: profile violations are permitted and logged. Useful for testing and developing new profiles.
- Enforced/Confined: enforces profile policy as well as logging the violation.

Using AppArmor

The optional apparmor-utils package contains command line utilities that you can use to change the AppArmor execution mode, find the status of a profile, create new profiles, etc.

• apparmor_status is used to view the current status of AppArmor profiles.

sudo apparmor_status

• aa-complain places a profile into *complain* mode.

```
sudo aa-complain /path/to/bin
```

• aa-enforce places a profile into *enforce* mode.

```
sudo aa-enforce /path/to/bin
```

• The /etc/apparmor.d directory is where the AppArmor profiles are located. It can be used to manipulate the *mode* of all profiles.

Enter the following to place all profiles into complain mode:

sudo aa-complain /etc/apparmor.d/*

To place all profiles in enforce mode:

```
sudo aa-enforce /etc/apparmor.d/*
```

• apparmor_parser is used to load a profile into the kernel. It can also be used to reload a currently loaded profile using the *-r* option after modifying it to have the changes take effect. To reload a profile:

```
sudo apparmor_parser -r /etc/apparmor.d/profile.name
```

systemctl can be used to *reload* all profiles:

sudo systemctl reload apparmor.service

• The /etc/apparmor.d/disable directory can be used along with the apparmor_parser -R option to *disable* a profile.

sudo ln -s /etc/apparmor.d/profile.name /etc/apparmor.d/disable/ sudo apparmor_parser -R /etc/apparmor.d/profile.name

To *re-enable* a disabled profile remove the symbolic link to the profile in /etc/apparmor.d/disable/. Then load the profile using the -*a* option.

sudo rm /etc/apparmor.d/disable/profile.name
cat /etc/apparmor.d/profile.name | sudo apparmor parser -a

• AppArmor can be disabled, and the kernel module unloaded by entering the following:

```
sudo systemctl stop apparmor.service
sudo systemctl disable apparmor.service
```

• To re-enable AppArmor enter:

```
sudo systemctl enable apparmor.service
sudo systemctl start apparmor.service
```

Note

Replace *profile.name* with the name of the profile you want to manipulate. Also, replace /path/to/bin/ with the actual executable file path. For example for the ping command use /bin/ping

Profiles

AppArmor profiles are simple text files located in /etc/apparmor.d/. The files are named after the full path to the executable they profile replacing the "/" with ".". For example /etc/apparmor.d/bin.ping is the AppArmor profile for the /bin/ping command.

There are two main type of rules used in profiles:

- Path entries: detail which files an application can access in the file system.
- Capability entries: determine what privileges a confined process is allowed to use.

As an example, take a look at <code>/etc/apparmor.d/bin.ping</code>:

```
#include <tunables/global>
/bin/ping flags=(complain) {
    #include <abstractions/base>
    #include <abstractions/consoles>
    #include <abstractions/nameservice>
    capability net_raw,
    capability setuid,
    network inet raw,
    /bin/ping mixr,
    /etc/modules.conf r,
}
```

- *#include <tunables/global>:* include statements from other files. This allows statements pertaining to multiple applications to be placed in a common file.
- /bin/ping flags=(complain): path to the profiled program, also setting the mode to complain.
- capability net_raw,: allows the application access to the CAP_NET_RAW Posix.1e capability.
- /bin/ping mixr,: allows the application read and execute access to the file.

Note

After editing a profile file the profile must be reloaded. See above at Using AppArmor for details.

Creating a Profile

• *Design a test plan:* Try to think about how the application should be exercised. The test plan should be divided into small test cases. Each test case should have a small description and list the steps to follow.

Some standard test cases are:

- Starting the program.
- Stopping the program.
- Reloading the program.
- Testing all the commands supported by the init script.
- Generate the new profile: Use aa-genprof to generate a new profile. From a terminal:

sudo aa-genprof executable

For example:

sudo aa-genprof slapd

- To get your new profile included in the apparmor-profiles package, file a bug in *Launchpad* against the AppArmor package:
 - Include your test plan and test cases.
 - Attach your new profile to the bug.

Updating Profiles

When the program is misbehaving, audit messages are sent to the log files. The program aa-logprof can be used to scan log files for AppArmor audit messages, review them and update the profiles. From a terminal:

sudo aa-logprof

Further pre-existing Profiles

The packages apport-profiles and apparmor-profiles-extra ship some experimental profiles for AppArmor security policies.

Do not expect these profiles to work out-of-the-box, but they can give you a head start when trynig to create a new profile by starting off a base that exists.

These profiles are not considered mature enough to be shipped in enforce mode by default. Therefore they are shipped in complain mode so that users can test them, choose which are desired, and help improve them upstream if needed.

Some even more experimental profiles carried by the package are placed in /usr/share/doc/apparmor-profiles/extras/

Checking and debugging denies

You will see in 'dmesg' and any log that collects kernel messages if you have hit a deny. Bight away it is worth to know that this will cover any access that was denied because it was no

Right away it is worth to know that this will cover any access that was denied because it was not allowed, but explicit denies will put no message in your logs at all.

Examples might look like:

[1521056.552037] audit: type=1400 audit(1571868402.378:24425): apparmor="DENIED" operation="open" profile="/usr/sbin/cu browsed" name="/var/lib/libvirt/dnsmasq/" pid=1128 comm="cups-browsed" requested_mask="r" denied_mask="r" fsuid=0 ouid= [1482106.651527] audit: type=1400 audit(1571829452.330:24323): apparmor="DENIED" operation="sendmsg" profile="snap.lxd.

That follows a generic structure starting with a timestamp, an audit tag and the category apparmor="DENIED". From the following fields you can derive what was going on and why it was failing.

In the examples above that would be

First example:

- operation: open (program tried to open a file)
- profile: /usr/sbin/cups-browsed (you'll find /etc/apparmor.d/usr.bin.cups-browsed)
- name: /var/lib/libvirt/dnsmasq (what it wanted to access)
- pid/comm: the program that did trigger the access
- requested_mask/denied_mask/fsuid/ouid: parameters of that open call

Second example:

- operation: sendmsg (program tried send via network)
- profile: snap.lxd.lxc (snaps are special, you'll find /var/lib/snapd/apparmor/profiles/snap.lxd.lxc)
- pid/comm: the program that did trigger the access
- laddr/lport/faddr/fport/family/sock_type/protocol: parameters of that sendmsg call

That way you know in which profile and at what action you have to start if you consider either debugging or adapting the profiles.

Profile customization

Profiles are meant to provide security and thereby can't be all too open. But quite often a very special setup would work with a profile if it wold *just allow this one extra access*. To handle that there are three ways.

- modify the profile itself
 - always works, but has the drawback that profiles are in /etc and considered conffiles. So after modification
 on a related package update you might get a conffile prompt. Worst case depending on configuration
 automatic updates might even override it and your custom rule is gone.
- use tunables
 - those provide variables that can be used in templates, for example if you want a custom dir considered as it would be a home directory you could modify /etc/apparmor.d/tunables/home which defines the base path rules use for home directories
 - by design those variables will only influence profiles that use them
- modify a local override
 - to mitigate the drawbacks of above approaches *local includes* got introduced adding the ability to write arbitrary rules that will be used, and not get issues on upgrades that modify the packaged rule.
 - The files can be found in /etc/apparmor.d/local/ and exist for the packages that are known to sometimes need slight tweaks for special setups

References

- See the AppArmor Administration Guide for advanced configuration options.
- For details using AppArmor with other Ubuntu releases see the AppArmor Community Wiki page.
- The OpenSUSE AppArmor page is another introduction to AppArmor.
- (https://wiki.debian.org/AppArmor) is another introduction and basic howto for AppArmor.
- A great place to get involved with the Ubuntu Server community and to ask for AppArmor assistance is the *#ubuntu-server* IRC channel on Libera. The *#ubuntu-security* IRC channel may also be of use.

Firewall

Introduction

The Linux kernel includes the *Netfilter* subsystem, which is used to manipulate or decide the fate of network traffic headed into or through your server. All modern Linux firewall solutions use this system for packet filtering.

The kernel's packet filtering system would be of little use to administrators without a userspace interface to manage it. This is the purpose of iptables: When a packet reaches your server, it will be handed off to the Netfilter subsystem for acceptance, manipulation, or rejection based on the rules supplied to it from userspace via iptables. Thus, iptables is all you need to manage your firewall, if you're familiar with it, but many frontends are available to simplify the task.

ufw - Uncomplicated Firewall

The default firewall configuration tool for Ubuntu is ufw. Developed to ease iptables firewall configuration, ufw provides a user-friendly way to create an IPv4 or IPv6 host-based firewall.

ufw by default is initially disabled. From the ufw man page:

"ufw is not intended to provide complete firewall functionality via its command interface, but instead provides an easy way to add or remove simple rules. It is currently mainly used for host-based firewalls."

The following are some examples of how to use ufw:

• First, ufw needs to be enabled. From a terminal prompt enter:

sudo ufw enable

- To open a port (SSH in this example):
 - sudo ufw allow 22
- Rules can also be added using a *numbered* format:
 - sudo ufw insert 1 allow 80
- Similarly, to close an opened port:

sudo ufw deny 22

• To remove a rule, use delete followed by the rule:

sudo ufw delete deny 22

• It is also possible to allow access from specific hosts or networks to a port. The following example allows SSH access from host 192.168.0.2 to any IP address on this host:

sudo ufw allow proto tcp from 192.168.0.2 to any port 22

Replace 192.168.0.2 with 192.168.0.0/24 to allow SSH access from the entire subnet.

• Adding the *-dry-run* option to a *ufw* command will output the resulting rules, but not apply them. For example, the following is what would be applied if opening the HTTP port:

```
sudo ufw --dry-run allow http
*filter
:ufw-user-input - [0:0]
:ufw-user-output - [0:0]
:ufw-user-forward - [0:0]
:ufw-user-limit - [0:0]
:ufw-user-limit-accept - [0:0]
### RULES ###
### tuple ### allow tcp 80 0.0.0.0/0 any 0.0.0.0/0
-A ufw-user-input -p tcp --dport 80 -j ACCEPT
### END RULES ###
-A ufw-user-input -j RETURN
-A ufw-user-output -j RETURN
-A ufw-user-forward -j RETURN
-A ufw-user-limit -m limit --limit 3/minute -j LOG --log-prefix "[UFW LIMIT]: "
-A ufw-user-limit -j REJECT
-A ufw-user-limit-accept -j ACCEPT
COMMIT
Rules updated
```

• ufw can be disabled by:

sudo ufw disable

• To see the firewall status, enter:

sudo ufw status

• And for more verbose status information use:

sudo ufw status verbose

• To view the *numbered* format:

sudo ufw status numbered

Note

If the port you want to open or close is defined in /etc/services, you can use the port name instead of the number. In the above examples, replace 22 with ssh.

This is a quick introduction to using ufw. Please refer to the ufw man page for more information.

ufw Application Integration

Applications that open ports can include an ufw profile, which details the ports needed for the application to function properly. The profiles are kept in /etc/ufw/applications.d, and can be edited if the default ports have been changed.

• To view which applications have installed a profile, enter the following in a terminal:

sudo ufw app list

• Similar to allowing traffic to a port, using an application profile is accomplished by entering:

sudo ufw allow Samba

• An extended syntax is available as well:

ufw allow from 192.168.0.0/24 to any app Samba

Replace Samba and 192.168.0.0/24 with the application profile you are using and the IP range for your network.

Note

There is no need to specify the *protocol* for the application, because that information is detailed in the profile. Also, note that the *app* name replaces the *port* number.

• To view details about which ports, protocols, etc., are defined for an application, enter:

sudo ufw app info Samba

Not all applications that require opening a network port come with ufw profiles, but if you have profiled an application and want the file to be included with the package, please file a bug against the package in Launchpad.

ubuntu-bug nameofpackage

IP Masquerading

The purpose of IP Masquerading is to allow machines with private, non-routable IP addresses on your network to access the Internet through the machine doing the masquerading. Traffic from your private network destined for the Internet must be manipulated for replies to be routable back to the machine that made the request. To do this, the kernel must modify the *source* IP address of each packet so that replies will be routed back to it, rather than to the private IP address that made the request, which is impossible over the Internet. Linux uses *Connection Tracking* (conntrack) to keep track of which connections belong to which machines and reroute each return packet accordingly. Traffic leaving your private network is thus "masqueraded" as having originated from your Ubuntu gateway machine. This process is referred to in Microsoft documentation as Internet Connection Sharing.

ufw Masquerading

IP Masquerading can be achieved using custom ufw rules. This is possible because the current back-end for ufw is iptables-restore with the rules files located in /etc/ufw/*.rules. These files are a great place to add legacy iptables rules used without ufw, and rules that are more network gateway or bridge related.

The rules are split into two different files, rules that should be executed before ufw command line rules, and rules that are executed after ufw command line rules.

• First, packet forwarding needs to be enabled in ufw. Two configuration files will need to be adjusted, in /etc/default/ufw change the *DEFAULT_FORWARD_POLICY* to "ACCEPT":

DEFAULT_FORWARD_POLICY="ACCEPT"

Then edit /etc/ufw/sysctl.conf and uncomment:

net/ipv4/ip_forward=1

Similarly, for IPv6 forwarding uncomment:

net/ipv6/conf/default/forwarding=1

• Now add rules to the /etc/ufw/before.rules file. The default rules only configure the *filter* table, and to enable masquerading the *nat* table will need to be configured. Add the following to the top of the file just after the header comments:

```
# nat Table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Forward traffic from eth1 through eth0.
-A POSTROUTING -s 192.168.0.0/24 -o eth0 -j MASQUERADE
# don't delete the 'COMMIT' line or these nat table rules won't be processed
COMMIT
```

The comments are not strictly necessary, but it is considered good practice to document your configuration. Also, when modifying any of the *rules* files in /etc/ufw, make sure these lines are the last line for each table modified:

don't delete the 'COMMIT' line or these rules won't be processed COMMIT For each *Table* a corresponding *COMMIT* statement is required. In these examples only the *nat* and *filter* tables are shown, but you can also add rules for the *raw* and *mangle* tables.

Note

In the above example replace eth0, eth1, and 192.168.0.0/24 with the appropriate interfaces and IP range for your network.

• Finally, disable and re-enable ufw to apply the changes:

sudo ufw disable && sudo ufw enable

IP Masquerading should now be enabled. You can also add any additional FORWARD rules to the /etc/ufw/before.rules. It is recommended that these additional rules be added to the *ufw-before-forward* chain.

iptables Masquerading

iptables can also be used to enable Masquerading.

• Similar to ufw, the first step is to enable IPv4 packet forwarding by editing /etc/sysctl.conf and uncomment the following line:

```
net.ipv4.ip_forward=1
```

If you wish to enable IPv6 forwarding also uncomment:

net.ipv6.conf.default.forwarding=1

• Next, execute the sysctl command to enable the new settings in the configuration file:

```
sudo sysctl -p
```

• IP Masquerading can now be accomplished with a single iptables rule, which may differ slightly based on your network configuration:

sudo iptables -t nat -A POSTROUTING -s 192.168.0.0/16 -o ppp0 -j MASQUERADE

The above command assumes that your private address space is 192.168.0.0/16 and that your Internet-facing device is ppp0. The syntax is broken down as follows:

- -t nat the rule is to go into the nat table
- - A POSTROUTING the rule is to be appended (-A) to the POSTROUTING chain
- - s 192.168.0.0/16 the rule applies to traffic originating from the specified address space
- -o ppp0 the rule applies to traffic scheduled to be routed through the specified network device
- -j MASQUERADE traffic matching this rule is to "jump" (-j) to the MASQUERADE target to be manipulated as described above
- Also, each chain in the filter table (the default table, and where most or all packet filtering occurs) has a default *policy* of ACCEPT, but if you are creating a firewall in addition to a gateway device, you may have set the policies to DROP or REJECT, in which case your masqueraded traffic needs to be allowed through the FORWARD chain for the above rule to work:

```
sudo iptables -A FORWARD -s 192.168.0.0/16 -o ppp0 -j ACCEPT
sudo iptables -A FORWARD -d 192.168.0.0/16 -m state \
--state ESTABLISHED,RELATED -i ppp0 -j ACCEPT
```

The above commands will allow all connections from your local network to the Internet and all traffic related to those connections to return to the machine that initiated them.

• If you want masquerading to be enabled on reboot, which you probably do, edit /etc/rc.local and add any commands used above. For example add the first command with no filtering:

iptables -t nat -A POSTROUTING -s 192.168.0.0/16 -o ppp0 -j MASQUERADE

Logs

Firewall logs are essential for recognizing attacks, troubleshooting your firewall rules, and noticing unusual activity on your network. You must include logging rules in your firewall for them to be generated, though, and logging rules must come before any applicable terminating rule (a rule with a target that decides the fate of the packet, such as ACCEPT, DROP, or REJECT).

If you are using ufw, you can turn on logging by entering the following in a terminal:

sudo ufw logging on

To turn logging off in ufw, simply replace on with off in the above command.

If using iptables instead of ufw, enter:

```
sudo iptables -A INPUT -m state --state NEW -p tcp --dport 80 \
-j LOG --log-prefix "NEW HTTP CONN: "
```

A request on port 80 from the local machine, then, would generate a log in dmesg that looks like this (single line split into 3 to fit this document):

The above log will also appear in /var/log/messages, /var/log/syslog, and /var/log/kern.log. This behavior can be modified by editing /etc/syslog.conf appropriately or by installing and configuring ulogd and using the ULOG target instead of LOG. The ulogd daemon is a userspace server that listens for logging instructions from the kernel specifically for firewalls, and can log to any file you like, or even to a PostgreSQL or MySQL database. Making sense of your firewall logs can be simplified by using a log analyzing tool such as logwatch, fwanalog, fwlogwatch, or lire.

Other Tools

There are many tools available to help you construct a complete firewall without intimate knowledge of iptables. A command-line tool with plain-text configuration files:

• Shorewall is a very powerful solution to help you configure an advanced firewall for any network.

References

- The Ubuntu Firewall wiki page contains information on the development of ufw.
- Also, the ufw manual page contains some very useful information: man ufw.
- See the packet-filtering-HOWTO for more information on using iptables.
- The nat-HOWTO contains further details on masquerading.
- The IPTables HowTo in the Ubuntu wiki is a great resource.

Certificates

One of the most common forms of cryptography today is *public-key* cryptography. Public-key cryptography utilizes a *public key* and a *private key*. The system works by encrypting information using the public key. The information can then only be decrypted using the private key.

A common use for public-key cryptography is encrypting application traffic using a Secure Socket Layer (SSL) or Transport Layer Security (TLS) connection. One example: configuring Apache to provide *HTTPS*, the HTTP protocol over SSL/TLS. This allows a way to encrypt traffic using a protocol that does not itself provide encryption.

A *certificate* is a method used to distribute a *public key* and other information about a server and the organization who is responsible for it. Certificates can be digitally signed by a *Certification Authority*, or CA. A CA is a trusted third party that has confirmed that the information contained in the certificate is accurate.

Types of Certificates

To set up a secure server using public-key cryptography, in most cases, you send your certificate request (including your public key), proof of your company's identity, and payment to a CA. The CA verifies the certificate request and your identity, and then sends back a certificate for your secure server. Alternatively, you can create your own *self-signed* certificate.

Note

Note that self-signed certificates should not be used in most production environments.

Continuing the HTTPS example, a CA-signed certificate provides two important capabilities that a self-signed certificate does not:

• Browsers (usually) automatically recognize the CA signature and allow a secure connection to be made without prompting the user.

• When a CA issues a signed certificate, it is guaranteeing the identity of the organization that is providing the web pages to the browser.

Most of the software supporting SSL/TLS have a list of CAs whose certificates they automatically accept. If a browser encounters a certificate whose authorizing CA is not in the list, the browser asks the user to either accept or decline the connection. Also, other applications may generate an error message when using a self-signed certificate.

The process of getting a certificate from a CA is fairly easy. A quick overview is as follows:

- 1. Create a private and public encryption key pair.
- 2. Create a certificate signing request based on the public key. The certificate request contains information about your server and the company hosting it.
- 3. Send the certificate request, along with documents proving your identity, to a CA. We cannot tell you which certificate authority to choose. Your decision may be based on your past experiences, or on the experiences of your friends or colleagues, or purely on monetary factors.

Once you have decided upon a CA, you need to follow the instructions they provide on how to obtain a certificate from them.

- 4. When the CA is satisfied that you are indeed who you claim to be, they send you a digital certificate.
- 5. Install this certificate on your secure server, and configure the appropriate applications to use the certificate.

Generating a Certificate Signing Request (CSR)

Whether you are getting a certificate from a CA or generating your own self-signed certificate, the first step is to generate a key.

If the certificate will be used by service daemons, such as Apache, Postfix, Dovecot, etc., a key without a passphrase is often appropriate. Not having a passphrase allows the services to start without manual intervention, usually the preferred way to start a daemon.

This section will cover generating a key with a passphrase, and one without. The non-passphrase key will then be used to generate a certificate that can be used with various service daemons.

Warning

Running your secure service without a passphrase is convenient because you will not need to enter the passphrase every time you start your secure service. But it is insecure and a compromise of the key means a compromise of the server as well.

To generate the keys for the Certificate Signing Request (CSR) run the following command from a terminal prompt:

```
openssl genrsa -des3 -out server.key 2048
```

Generating RSA private key, 2048 bit long modulus

You can now enter your passphrase. For best security, it should at least contain eight characters. The minimum length when specifying -des3 is four characters. As a best practice it should include numbers and/or punctuation and not be a word in a dictionary. Also remember that your passphrase is case-sensitive.

Re-type the passphrase to verify. Once you have re-typed it correctly, the server key is generated and stored in the server.key file.

Now create the insecure key, the one without a passphrase, and shuffle the key names:

openssl rsa -in server.key -out server.key.insecure mv server.key server.key.secure mv server.key.insecure server.key

The insecure key is now named server.key, and you can use this file to generate the CSR without passphrase.

To create the CSR, run the following command at a terminal prompt:

openssl req -new -key server.key -out server.csr

It will prompt you enter the passphrase. If you enter the correct passphrase, it will prompt you to enter Company Name, Site Name, Email Id, etc. Once you enter all these details, your CSR will be created and it will be stored in the server.csr file.

You can now submit this CSR file to a CA for processing. The CA will use this CSR file and issue the certificate. On the other hand, you can create self-signed certificate using this CSR.

Creating a Self-Signed Certificate

To create the self-signed certificate, run the following command at a terminal prompt:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

The above command will prompt you to enter the passphrase. Once you enter the correct passphrase, your certificate will be created and it will be stored in the server.crt file.

Warning

If your secure server is to be used in a production environment, you probably need a CA-signed certificate. It is not recommended to use self-signed certificate.

Installing the Certificate

You can install the key file server.key and certificate file server.crt, or the certificate file issued by your CA, by running following commands at a terminal prompt:

sudo cp server.crt /etc/ssl/certs
sudo cp server.key /etc/ssl/private

Now simply configure any applications, with the ability to use public-key cryptography, to use the *certificate* and *key* files. For example, Apache can provide HTTPS, Dovecot can provide IMAPS and POP3S, etc.

Certification Authority

If the services on your network require more than a few self-signed certificates it may be worth the additional effort to setup your own internal Certification Authority (CA). Using certificates signed by your own CA, allows the various services using the certificates to easily trust other services using certificates issued from the same CA.

First, create the directories to hold the CA certificate and related files:

```
sudo mkdir /etc/ssl/CA
sudo mkdir /etc/ssl/newcerts
```

The CA needs a few additional files to operate, one to keep track of the last serial number used by the CA, each certificate must have a unique serial number, and another file to record which certificates have been issued:

```
sudo sh -c "echo '01' > /etc/ssl/CA/serial"
sudo touch /etc/ssl/CA/index.txt
```

The third file is a CA configuration file. Though not strictly necessary, it is very convenient when issuing multiple certificates. Edit /etc/ssl/openssl.cnf, and in the / CA_default | change:

dir	=	/etc/ssl	#	Where everything is kept
database	=	<pre>\$dir/CA/index.txt</pre>	#	database index file.
certificate	=	<pre>\$dir/certs/cacert.pem</pre>	#	The CA certificate
serial	=	\$dir/CA/serial	#	The current serial number
private_key	=	<pre>\$dir/private/cakey.per</pre>	n#	The private key

Next, create the self-signed root certificate:

openssl req -new -x509 -extensions v3_ca -keyout cakey.pem -out cacert.pem -days 3650

You will then be asked to enter the details about the certificate.

Now install the root certificate and key:

```
sudo mv cakey.pem /etc/ssl/private/
sudo mv cacert.pem /etc/ssl/certs/
```

You are now ready to start signing certificates. The first item needed is a Certificate Signing Request (CSR), see Generating a Certificate Signing Request (CSR) for details. Once you have a CSR, enter the following to generate a certificate signed by the CA:

sudo openssl ca -in server.csr -config /etc/ssl/openssl.cnf

After entering the password for the CA key, you will be prompted to sign the certificate, and again to commit the new certificate. You should then see a somewhat large amount of output related to the certificate creation.

There should now be a new file, /etc/ssl/newcerts/01.pem, containing the same output. Copy and paste everything beginning with the line: ----BEGIN CERTIFICATE----- and continuing through the line: ----END CERTIFICATE----- lines to a file named after the hostname of the server where the certificate will be installed. For example mail.example.com.crt, is a nice descriptive name.

Subsequent certificates will be named 02.pem, 03.pem, etc.

Note

Replace *mail.example.com.crt* with your own descriptive name.

Finally, copy the new certificate to the host that needs it, and configure the appropriate applications to use it. The default location to install certificates is /etc/ssl/certs. This enables multiple services to use the same certificate without overly complicated file permissions.

For applications that can be configured to use a CA certificate, you should also copy the /etc/ssl/certs/cacert.pem file to the /etc/ssl/certs/ directory on each server.

References

- The Wikipedia HTTPS page has more information regarding HTTPS.
- For more information on *OpenSSL* see the OpenSSL Home Page.
- Also, O'Reilly's Network Security with OpenSSL is a good in-depth reference.

Installing a root CA certificate in the trust store

Enterprise environments sometimes have a local Certificate Authority (CA) that issues certificates for use within the organization. For an Ubuntu server to be functional and trust the hosts in this environment this CA must be installed in Ubuntu's trust store.

How to recognize the form (PEM or DER)?

To install a certificate in the trust store it must be in PEM form. A PEM-formatted certificate is human-readable in base64 format, and starts with the lines ----BEGIN CERTIFICATE----. If you see these lines, you're ready to install. If not, it is most likely a DER certificate and needs to be converted.

Installing a certificate in PEM form

Assuming a PEM-formatted root CA certificate is in local-ca.crt, follow the steps below to install it.

Note: It is important to have the .crt extension on the file, otherwise it will not be processed.

\$ sudo apt-get install -y ca-certificates \$ sudo cp local-ca.crt /usr/local/share/ca-certificates \$ sudo update-ca-certificates

After this point you can use Ubuntu's tools like curl and wget to connect to local sites.

Converting from DER-form to PEM-form

Convert a DER-formatted certificate called local-ca.der to PEM form like this: \$ sudo openssl x509 -inform der -outform pem -in local-ca.der -out local-ca.crt

The CA trust store location

The CA trust store as generated by update-ca-certificates is available at the following locations:

- As a single file (PEM bundle) in /etc/ssl/certs/ca-certificates.crt
- As an OpenSSL compatible certificate directory in /etc/ssl/certs

Console Security

As with any other security barrier you put in place to protect your server, it is pretty tough to defend against untold damage caused by someone with physical access to your environment, for example, theft of hard drives, power or service disruption, and so on. Therefore, console security should be addressed merely as one component of your overall physical security strategy. A locked "screen door" may deter a casual criminal, or at the very least slow down a determined one, so it is still advisable to perform basic precautions with regard to console security.

The following instructions will help defend your server against issues that could otherwise yield very serious consequences.

Disable Ctrl+Alt+Delete

Anyone that has physical access to the keyboard can simply use the Ctrl+Alt+Delete key combination to reboot the server without having to log on. While someone could simply unplug the power source, you should still prevent the use of this key combination on a production server. This forces an attacker to take more drastic measures to reboot the server, and will prevent accidental reboots at the same time.

To disable the reboot action taken by pressing the Ctrl+Alt+Delete key combination, run the following two commands:

sudo systemctl mask ctrl-alt-del.target
sudo systemctl daemon-reload

Virtualization

Virtualization is being adopted in many different environments and situations. If you are a developer, virtualization can provide you with a contained environment where you can safely do almost any sort of development safe from messing up your main working environment. If you are a systems administrator, you can use virtualization to more easily separate your services and move them around based on demand.

The default virtualization technology supported in Ubuntu is KVM. For Intel and AMD hardware KVM requires virtualization extensions. But KVM is also available for IBM Z and LinuxONE, IBM POWER as well as for ARM64. Qemu is part of the KVM experience being the userspace backend for it, but it also can be used for hardware without virtualization extensions by using its TCG mode.

While virtualization is in many ways similar to containers those are different and implemented via other solutions like LXD, systemd-nspawn, containerd and others.

Qemu

Qemu is a machine emulator that can run operating systems and programs for one machine on a different machine. Mostly it is not used as emulator but as virtualizer in collaboration with KVM kernel components. In that case it utilizes the virtualization technology of the hardware to virtualize guests.

While qemu has a command line interface and a monitor to interact with running guests those is rarely used that way for other means than development purposes. Libvirt provides an abstraction from specific versions and hypervisors and encapsulates some workarounds and best practices.

Running Qemu/KVM

While there are much more user friendly and comfortable ways, using the command below is probably the quickest way to see some called Ubuntu moving on screen is directly running it from the netboot iso.

Warning: this is just for illustration - not generally recommended without verifying the checksums; Multipass and UVTool are much better ways to get actual guests easily.

Run:

sudo qemu-system-x86_64 -enable-kvm -cdrom http://archive.ubuntu.com/ubuntu/dists/bionicupdates/main/installer-amd64/current/images/netboot/mini.iso

You could download the ISO for faster access at runtime and e.g. add a disk to the same by:

• creating the disk

```
qemu-img create -f qcow2 disk.qcow 5G
```

• Using the disk by adding -drive file=disk.qcow,format=qcow2

Those tools can do much more, as you'll find in their respective (long) man pages. There also is a vast assortment of auxiliary tools to make them more consumable for specific use-cases and needs - for example virt-manager for UI driven use through libvirt. But in general - even the tools eventually use that - it comes down to:

```
qemu-system-x86_64 options image[s]
```

So take a look at the man page of qemu, qemu-img and the documentation of qemu and see which options are the right one for your needs.

Graphics

Graphics for qemu/kvm always comes in two pieces.

- A front end controlled via the -vga argument which is provided to the guest. Usually one of cirrus, std, qxl, virtio. The default these days is qxl which strikes a good balance between guest compatibility and performance. The guest needs a driver for what is selected, which is the most common reason to switch from the default to either cirrus (e.g. very old Windows versions)
- A back end controlled via the -display argument which is what the host uses to actually display the graphical content. That can be an application window via gtk or a vnc.
- In addition one can enable the -spice back-end (can be done in addition to vnc) which can be faster and provides more authentication methods than vnc.
- if you want no graphical output at all you can save some memory and cpu cycles by setting -nographic

If you run with spice or vnc you can use native vnc tools or virtualization focused tools like virt-viewer. More about these in the libvirt section.

All those options above are considered basic usage of graphics. There are advanced options for further needs. Those cases usually differ in their ease-of-use and capability are:

- Need some 3D acceleration: -vga virtio with a local display having a GL context -display gtk,gl=on; That will use virgil3d on the host and needs guest drivers for [virt3d] which are common in Linux since Kernels >=4.4 but hard to get by for other cases. While not as fast as the next two options, the big benefit is that it can be used without additional hardware and without a proper IOMMU setup for device passthrough.
- *Need native performance*: use PCI passthrough of additional GPUs in the system. You'll need an IOMMU setup and unbind the cards from the host before you can pass it through like -device vfio-pci,host=05:00.0,bus=1,addr=00.0,multifunction=on,x-vga=on -device vfio-pci,host=05:00.1,bus=1,addr=00.1
- Need native performance, but multiple guests per card: Like PCI Passthrough, but using mediated devices to shard a card on the Host into multiple devices and pass those like -display gtk,gl=on -device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000:00:02.0/4dd511f6-ec08-11e8-b839-2f163ddee3b3,display=on,rombar=0. More at kraxel on vgpu and Ubuntu GPU mdev evaluation. The sharding of the cards is driver specific and therefore will differ per manufacturer like Intel or Nvidia.

Especially the advanced cases can get pretty complex, therefore it is recommended to use qemu through libvirt for those cases. Libvirt will take care of all but the host kernel/bios tasks of such configurations. Below are the common basic actions needed for faster options being pass-through and mediated devices pass-through.

Preparations for PCI and mediated devices pass-through - IOMMU

The initial steps for both of these are the same, you want to ensure your system has its IOMMU enabled and the device to pass should be in a group of its own. Enablement of the VT-d and IOMMU is usually an bios action and thereby manufacturer dependent.

On the kernel side for the IOMMU feature there are various options you can enable/configure. In recent Ubuntu Kernels ($\geq =5.4 =>$ Focal or Bionic-HWE kernels) everything usually works by default, unless your hardware setup makes you need any of those tuning options.

** Note **: The card used in all examples below e.g. when filtering for PCI IDs or assigning it is an Nvidia V100 on PCI id 41.00.0
\$ lspci | grep 3D
41:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 PCIe 16GB] (rev a1)

You can check your bootup kernel messages for iommu/dmar messages or even filter it for a particular pci id.

```
# List all
$ dmesg | grep -i -e DMAR -e IOMMU
[ 3.509232] iommu: Default domain type: Translated
...
[ 4.516995] pci 0000:00:01.0: Adding to iommu group 0
```

[4.702729] perf/amd iommu: Detected AMD IOMMU #0 (2 banks, 4 counters/bank).

```
# Filtered for the installed 3D card
dmesg | grep -i -e DMAR -e IOMMU | grep $(lspci | awk '/ 3D / {print $1}' )
[ 4.598150] pci 0000:41:00.0: Adding to iommu group 66
```

If you have a particular device and want to check for its group you can do that via sysfs. If you have multiple cards or want the full list you can traverse the same sysfs paths for that.

```
# Find the group for our example card
$ find /sys/kernel/iommu_groups/ -name "*$(lspci | awk '/ 3D / {print $1}')*"
/sys/kernel/iommu_groups/66/devices/0000:41:00.0
# Check if there are other devices in this group
ll /sys/kernel/iommu_groups/66/devices/
lrwxrwxrwx 1 root root 0 Jan 3 06:57 0000:41:00.0 -> ../../../devices/pci0000:40/0000:40:03.1/0000:41:00.0/
```

Another useful tool for this stage - but not going into details here - can be virsh node*, especially virsh nodedev-list --tree and virsh nodedev-dumpxml pcidev>

** Note **

. . .

Some older or non server boards tend to group devices in one IOMMU group which isn't very useful as it means you'll need to pass "all or none of them" to the same guest.

Preparations for PCI and mediated devices pass-through - block host drivers

For both you'd want to ensure the normal driver isn't loaded. In some cases you can do that at runtime via virsh nodedev-detach <pcidevice>. Libvirt would later even do that automatically is on the passthrough configuration you have set <hostdev mode='subsystem' type='pci' managed='yes'>.

This usually works fine e.g. for network cards, but some other devices like GPUs do not like too much to be unassigned, so there the required step usually is to block loading the drivers you do not want to be loaded. In our GPU example the nouveau driver would load and that has to be blocked. To do so you can create a modprobe blacklist.

echo "blacklist nouveau" | sudo tee /etc/modprobe.d/blacklist-nouveau.conf
echo "options nouveau modeset=0" | sudo tee -a /etc/modprobe.d/blacklist-nouveau.conf
sudo update-initramfs -u
sudo reboot

You can check the kernel modules loaded and available via ${\tt lspci}$ -v

```
$ lspci -v | grep -A 10 " 3D "
41:00.0 3D controller: NVIDIA Corporation GV100GL [Tesla V100 PCIe 16GB] (rev a1)
...
Kernel modules: nvidiafb, nouveau
```

If the configuration did not work instead it would show:

Kernel driver in use: nouveau

Preparations for mediated devices pass-through - driver

For PCI passthrough the above would be all preparations needed, for mediated devices one also needs to install and set up the host driver, the example here continues with our Nvidia V100 which is supported and available by Nvidia. There also is a Nvidia document about the same steps available about installation and configuration of vGPU on Ubuntu.

Once you got the drivers from Nvidia like nvidia-vgpu-ubuntu-470_470.68_amd64.deb install them and check (as above) that that driver is loaded. The one you need to see is nvidia_vgpu_vfio:

<pre>\$ lsmod grep nvid</pre>	ia	
nvidia_vgpu_vfio	53248	38
nvidia	35282944	586 nvidia_vgpu_vfio
mdev	24576	2 vfio_mdev,nvidia_vgpu_vfio
drm	491520	<pre>6 drm_kms_helper,drm_vram_helper,nvidia</pre>

Note:

While it is working without, for full capabilities you'll also need to configure the vgpu manager (came with above package) and a license server so that each guest can grab a license for the vGPU provided to it. Please see Nvidias documentation for the license server. While (as of Q1 2022) not officially supported on

Linux, it might be worth to note that it runs quite fine in Ubuntu with sudo apt install unzip defaultjre tomcat9 liblog4j2-java libslf4j-java using /var/lib/tomcat9 as the server path in the license server installer.

Examples of those when running fine

```
# general status
$ systemctl status nvidia-vgpu-mgr
    Loaded: loaded (/lib/system/system/nvidia-vgpu-mgr.service; enabled; vendor preset: enabled)
    Active: active (running) since Tue 2021-09-14 07:30:19 UTC; 3min 58s ago
    Process: 1559 ExecStart=/usr/bin/nvidia-vgpu-mgr (code=exited, status=0/SUCCESS)
  Main PID: 1564 (nvidia-vgpu-mgr)
     Tasks: 1 (limit: 309020)
    Memory: 1.1M
    CGroup: /system.slice/nvidia-vgpu-mgr.service
            └─1564 /usr/bin/nvidia-vgpu-mgr
Sep 14 07:30:19 node-watt systemd[1]: Starting NVIDIA vGPU Manager Daemon...
Sep 14 07:30:19 node-watt systemd[1]: Started NVIDIA vGPU Manager Daemon.
Sep 14 07:30:20 node-watt nvidia-vgpu-mgr[1564]: notice: vmiop_env_log: nvidia-vgpu-mgr daemon started
# Entries when a guest gets a vGPU passed
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): gpu-pci-id : 0x4100
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): vgpu_type : Quadro
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): Framebuffer: 0x1dc000000
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): Virtual Device Id: 0x1db4:0x1252
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): FRL Value: 60 FPS
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: Driver Version: 470.68
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop log: (0x0): vGPU supported range: (0x70001, 0xb0001)
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): Init frame copy engine: syncing...
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: (0x0): vGPU migration enabled
Sep 14 08:29:50 node-watt nvidia-vgpu-mgr[2866]: notice: vmiop_log: display_init inst: 0 successful
```

Entries when a guest grabs a license Sep 15 06:55:50 node-watt nvidia-vgpu-mgr[4260]: notice: vmiop_log: (0x0): vGPU license state: Unlicensed (Unrestricted) Sep 15 06:55:52 node-watt nvidia-vgpu-mgr[4260]: notice: vmiop_log: (0x0): vGPU license state: Licensed

In the guest the card is then fully recognized and enabled

\$	nvidia-smi	-a		grep	- A	2	"Licensed	Product
----	------------	----	--	------	-----	---	-----------	---------

vGPU Software Licensed Product

Product Name	: NVIDIA RTX Virtual Workstation
License Status	: Licensed

A mediated device is essentially partitioning of a hardware device utilizing firmware and host driver features. That brings quite some flexibility and options, in our example we can split our 16G GPU into 2x8G, 4x4G, 8x2G or 16x1G just as we need it. The following gives an example how to split it into two 8G cards for a compute profile and pass those to guests.

Please refer to the nvidia documentation for advanced tunungs and different card profiles.

The tool to list and configure those mediated devices is mdevctl. It will list the available types.

```
$ sudo mdevctl types
...
nvidia-300
Available instances: 0
Device API: vfio-pci
Name: GRID V100-8C
Description: num_heads=1, frl_config=60, framebuffer=8192M, max_resolution=4096x2160, max_instance=2
```

Knowing the pci id 0000:41:00.0 and the mediated device type we want nvidia-300 we can now create those mediated devices:

```
$ sudo mdevctl define --parent 0000:41:00.0 --type nvidia-300
bc127e23-aaaa-4d06-a7aa-88db2dd538e0
$ sudo mdevctl define --parent 0000:41:00.0 --type nvidia-300
1360ce4b-2ed2-4f63-abb6-8cdb92100085
```

\$ sudo mdevctl start --parent 0000:41:00.0 --uuid bc127e23-aaaa-4d06-a7aa-88db2dd538e0 \$ sudo mdevctl start --parent 0000:41:00.0 --uuid 1360ce4b-2ed2-4f63-abb6-8cdb92100085

After the above you can check the UUID of your ready mediated devices

```
$ sudo mdevctl list -d
bcl27e23-aaaa-4d06-a7aa-88db2dd538e0 0000:41:00.0 nvidia-108 manual (active)
1360ce4b-2ed2-4f63-abb6-8cdb92100085 0000:41:00.0 nvidia-108 manual (active)
```

Those UUIDs can then be used to pass the mediated devices to the guest - which from here is rather similar to the pass through of a full PCI device.

Passing through PCI or mediated devices

After the above setup is ready one can pass through those devices, in libvirt for a PCI pass-through that looks like:

```
<hostdev mode='subsystem' type='pci' managed='yes'>
<source>
<address domain='0x0000' bus='0x41' slot='0x00' function='0x0'/>
</source>
</hostdev>
```

And for mediated devices it is quite similar, but using the UUID.

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
<source>
<address uuid='634fc146-50a3-4960-ac30-f09e5cedc674'/>
</source>
</hostdev>
```

Those sections can either be part of the guest definition itself to be added on guest startup and freed on guest shutdown. Or they can be in a file and used by for hot-add remove if the hardware device and its drivers support it virsh attach-device.

Note

This works great on Focal, but type='none' as well as display='off' weren't available on Bionic. If this level of control is required one would need to consider using the Ubuntu Cloud Archive or Server-Backports for a newer stack of the virtualization components.

And finally it might be worth to not that while mediated devices become more common and known for vGPU handling, they are a general infrastructure also used for example for s390x vfio-ccw.

Upgrading the machine type

If you are unsure what this is, you might consider this as buying (virtual) Hardware of the same spec but a newer release date. You are encouraged in general and might want to update your machine type of an existing defined guests in particular to:

- to pick up latest security fixes and features
- continue using a guest created on a now unsupported release

In general it is recommended to update machine types when upgrading qemu/kvm to a new major version. But this can likely never be an automated task as this change is guest visible. The guest devices might change in appearance, new features will be announced to the guest and so on. Linux is usually very good at tolerating such changes, but it depends so much on the setup and workload of the guest that this has to be evaluated by the owner/admin of the system. Other operating systems where known to often have severe impacts by changing the hardware. Consider a machine type change similar to replacing all devices and firmware of a physical machine to the latest revision - all considerations that apply there apply to evaluating a machine type upgrade as well.

As usual with major configuration changes it is wise to back up your guest definition and disk state to be able to do a rollback just in case. There is no integrated single command to update the machine type via virsh or similar tools. It is a normal part of your machine definition. And therefore updated the same way as most others.

First shutdown your machine and wait until it has reached that state.

virsh shutdown <yourmachine>
wait
virsh list --inactive
should now list your machine as "shut off"

Then edit the machine definition and find the type in the type tag at the machine attribute.

virsh edit <yourmachine>
<type arch='x86_64' machine='pc-i440fx-bionic'>hvm</type>

Change this to the value you want. If you need to check what types are available via "-M ?" Note that while providing upstream types as convenience only Ubuntu types are supported. There you can also see what the current default would be. In general it is strongly recommended that you change to newer types if possible to exploit newer features, but also to benefit of bugfixes that only apply to the newer device virtualization.

```
kvm -M ?
# lists machine types, e.g.
pc-i440fx-xenial Ubuntu 16.04 PC (i440FX + PIIX, 1996) (default)
...
pc-i440fx-bionic Ubuntu 18.04 PC (i440FX + PIIX, 1996) (default)
...
```

After this you can start your guest again. You can check the current machine type from guest and host depending on your needs.

If you keep non-live definitions around - like xml files - remember to update those as well.

Note

This also is documented along some more constraints and considerations at the Ubuntu Wiki

QEMU usage for microvms

QEMU became another use case being used in a container-like style providing an enhanced isolation compared to containers but being focused on initialization speed.

To achieve that several components have been added:

- the microvm machine type
- alternative simple FW that can boot linux called qboot
- qemu build with reduced features matching these use cases called qemu-system-x86-microvm

For example if you happen to already have a stripped down workload that has all it would execute in an initrd you would run it maybe like the following:

\$ sudo qemu-system-x86_64 -M ubuntu-q35 -cpu host -m 1024 -enable-kvm -serial mon:stdio -nographic -display curses -append 'console=ttyS0,115200,8n1' -kernel vmlinuz-5.4.0-21 -initrd /boot/initrd.img-5.4.0-21-workload

To run the same with microvm, qboot and the minimized qemu you would do the following

- 1. run it with type microvm, so change -M to -M microvm
- 2. use the qboot bios, add
 -bios /usr/share/qemu/bios-microvm.bin
- 3. install the feature-minimized qemu-system package, do
 \$ sudo apt install qemu-system-x86-microvm

An invocation will now look like:

 $sudo qemu-system-x86_64$ -M microvm -bios /usr/share/qemu/bios-microvm.bin -cpu host -m 1024 -enable-kvm -serial mon:stdio -nographic -display curses -append 'console=ttyS0,115200,8n1' -kernel vmlinuz-5.4.0-21 -initrd /boot/initrd.img-5.4.0-21-workload

That will have cut down the qemu, bios and virtual-hw initialization time down a lot.

You will now - more than you already have before - spend the majority inside the guest which implies that further tuning probably has to go into that kernel and userspace initialization time.

** Note **

For now microvm, the qboot bios and other components of this are rather new upstream and not as verified as many other parts of the virtualization stack. Therefore none of the above is the default. Further being the default would also mean many upgraders would regress finding a qemu that doesn't have most features they are used to use. Due to that the qemu-system-x86-microvm package is intentionally a strong opt-in conflicting with the normal qemu-system-x86 package.

libvirt

The libvirt library is used to interface with different virtualization technologies. Before getting started with libvirt it is best to make sure your hardware supports the necessary virtualization extensions for KVM. Enter the following from a terminal prompt:

kvm-ok

A message will be printed informing you if your CPU does or does not support hardware virtualization.

Note

On many computers with processors supporting hardware assisted virtualization, it is necessary to activate an option in the BIOS to enable it.

Virtual Networking

There are a few different ways to allow a virtual machine access to the external network. The default virtual network configuration includes *bridging* and *iptables* rules implementing *usermode* networking, which uses the SLIRP protocol. Traffic is NATed through the host interface to the outside network.

To enable external hosts to directly access services on virtual machines a different type of *bridge* than the default needs to be configured. This allows the virtual interfaces to connect to the outside network through the physical interface, making them appear as normal hosts to the rest of the network.

There is a great example how to configure an own bridge and combining it with libvirt so that guests will use it at the netplan.io.

Installation

To install the necessary packages, from a terminal prompt enter:

```
sudo apt update
sudo apt install qemu-kvm libvirt-daemon-system
```

After installing libvirt-daemon-system, the user used to manage virtual machines will need to be added to the *libvirt* group. This is done automatically for members of the sudo group, but needs to be done in additon for anyone else that should access system wide libvirt resources. Doing so will grant the user access to the advanced networking options.

In a terminal enter:

sudo adduser \$USER libvirt

Note

If the user chosen is the current user, you will need to log out and back in for the new group membership to take effect.

You are now ready to install a *Guest* operating system. Installing a virtual machine follows the same process as installing the operating system directly on the hardware.

You either need:

- a way to automate the installation
- a keyboard and monitor will need to be attached to the physical machine.
- use cloud images which are meant to self-initialize (see Multipass and UVTool)

In the case of virtual machines a Graphical User Interface (GUI) is analogous to using a physical keyboard and mouse on a real computer. Instead of installing a GUI the virt-viewer or virt-manager application can be used to connect to a virtual machine's console using VNC. See Virtual Machine Manager / Viewer for more information.

Virtual Machine Management

The following section covers the command-line tools around *virsh* that are part of libvirt itself. But there are various options at different levels of complexities and feature-sets, like:

- multipass
- uvt
- virt-* tools
- openstack

virsh

There are several utilities available to manage virtual machines and libvirt. The virsh utility can be used from the command line. Some examples:

• To list running virtual machines:

virsh list

• To start a virtual machine:

virsh start <guestname>

• Similarly, to start a virtual machine at boot:

virsh autostart <guestname>

• Reboot a virtual machine with:

virsh reboot <guestname>

• The *state* of virtual machines can be saved to a file in order to be restored later. The following will save the virtual machine state into a file named according to the date:

```
virsh save <guestname> save-my.state
```

Once saved the virtual machine will no longer be running.

• A saved virtual machine can be restored using:

virsh restore save-my.state

• To shutdown a virtual machine do:

virsh shutdown <guestname>

• A CDROM device can be mounted in a virtual machine by entering:

virsh attach-disk <guestname> /dev/cdrom /media/cdrom

• To change the definition of a guest virsh exposes the domain via

virsh edit <guestname>

That will allow one to edit the XML representation that defines the guest and when saving it will apply format and integrity checks on these definitions.

Editing the XML directly certainly is the most powerful way, but also the most complex one. Tools like Virtual Machine Manager / Viewer can help unexperienced users to do most of the common tasks.

If virsh (or other vir* tools) shall connect to something else than the default qemu-kvm/system hypervisor one can find alternatives for the *connect* option in *man virsh* or libvirt doc

system and session scope

Virsh - as well as most other tools to manage virtualization - can be passed connection strings.

\$ virsh --connect qemu:///system

There are two options for the connection.

- qemu:///system connect locally as root to the daemon supervising QEMU and KVM domains
- $\mathsf{qemu:///session}$ connect locally as a normal user to his own set of QEMU and KVM domains

The *default* always was (and still is) qemu:///system as that is the behavior users are used to. But there are a few benefits (and drawbacks) to qemu:///session to consider it. qemu:///session is per user and can on a multi-user system be used to separate the people.

But most importantly things run under the permissions of the user which means no permission struggle on the just donwloaded image in your *\$HOME* or the just attached USB-stick.

On the other hand it can't access system resources that well, which includes network setup that is known to be hard with qemu:///session. It falls back to slirp networking which is functional but slow and makes it impossible to be reached from other systems.

qemu:///system is different in that it is run by the global system wide libvirt that can arbitrate resources as needed. But you might need to my and/or chown files to the right places permssions to have them usable.

Applications usually will decide on their primary use-case. Desktop-centric applications often choose qemu:///session while most solutions that involve an administrator anyway continue to default to qemu:///system.

Read more about that in the libvirt FAQ and this blog about the topic.

Migration

There are different types of migration available depending on the versions of libvirt and the hypervisor being used. In general those types are:

- offline migration
- live migration
- postcopy migration

There are various options to those methods, but the entry point for all of them is *virsh migrate*. Read the integrated help for more detail.

virsh migrate --help

Some useful documentation on constraints and considerations about live migration can be found at the Ubuntu Wiki

Device Passthrough / Hotplug

If instead of the here described hotplugging you want to always pass through a device add the xml content of the device to your static guest xml representation via e.g. virsh edit <guestname>. In that case you don't need to use attach/detach. There are different kinds of passthrough. Types available to you depend on your Hardware and software setup.

- USB hotplug/passthrough
- VF hotplug/Passthrough

But both kinds are handled in a very similar way and while there are various way to do it (e.g. also via qemu monitor) driving such a change via libvirt is recommended. That way libvirt can try to manage all sorts of special cases for you and also somewhat masks version differences.

In general when driving hotplug via libvirt you create a xml snippet that describes the device just as you would do in a static guest description. A usb device is usually identified by Vendor/Product id's:

```
<hostdev mode='subsystem' type='usb' managed='yes'>
<source>
<vendor id='0x0b6d'/>
<product id='0x3880'/>
</source>
</hostdev>
```

Virtual functions are usually assigned via their PCI-ID (domain, bus, slot, function).

```
<hostdev mode='subsystem' type='pci' managed='yes'>
<source>
<address domain='0x0000' bus='0x04' slot='0x10' function='0x0'/>
</source>
```

</hostdev>

Note

To get the Virtual function in the first place is very device dependent and can therefore not be fully covered here. But in general it involves setting up an iommu, registering via VFIO and sometimes requesting a number of VFs. Here an example on ppc64el to get 4 VFs on a device:

```
$ sudo modprobe vfio-pci
# identify device
$ lspci -n -s 0005:01:01.3
0005:01:01.3 0200: 10df:e228 (rev 10)
# register and request VFs
$ echo 10df e228 | sudo tee /sys/bus/pci/drivers/vfio-pci/new_id
$ echo 4 | sudo tee /sys/bus/pci/devices/0005\:01\:00.0/sriov_numvfs
```

You then attach or detach the device via libvirt by relating the guest with the xml snippet.

virsh attach-device <guestname> <device-xml>
Use the Device int the Guest
virsh detach-device <guestname> <device-xml>

Access Qemu Monitor via libvirt

The Qemu Monitor is the way to interact with qemu/KVM while a guest is running. This interface has many and very powerful features for experienced users. When running under libvirt that monitor interface is bound by libvirt itself for management purposes, but a user can run qemu monitor commands via libvirt still. The general syntax is virsh qemu-monitor-command [options] [guest] 'command'

Libvirt covers most use cases needed, but if you every want/need to work around libvirt or want to tweak very special options you can e.g. add a device that way:

virsh qemu-monitor-command --hmp focal-test-log 'drive_add 0 if=none,file=/var/lib/libvirt/images/test.img,format=raw,i

But since the monitor is so powerful, you can do a lot especially for debugging purposes like showing the guest registers:

```
virsh qemu-monitor-command --hmp y-ipns 'info registers'
RAX=00ffffc000000000 RBX=ffff8f0f5d5c7e48 RCX=000000000000000 RDX=ffffea00007571c0
RSI=000000000000000 RDI=ffff8f0fdd5c7e48 RBP=ffff8f0f5d5c7e18 RSP=ffff8f0f5d5c7df8
[...]
```

Huge Pages

Using huge pages can help to reduce TLB pressure, page table overhead and speed up some further memory relate actions. Furthermore by default Transparent huge pages are useful, but can be quite some overhead - so if it is clear that using huge pages is preferred making them explicit usually has some gains.

While huge page are admittedly harder to manage, especially later in the lifetime of a system if memory is fragmented they provide a useful boost especially for rather large guests.

Bonus: When using device pass through on very large guests there is an extra benefit of using huge pages as it is faster to do the initial memory clear on vfio dma pin.

Huge page allocation

Huge pages come in different sizes. A *normal* page usually is 4k and huge pages are eithe 2M or 1G, but depending on the architecture other options are possible.

The most simple, yet least reliable way to allocate some huge pages is to just echo a value to sysfs Be sure to re-check if it worked.

```
$ echo 256 | sudo tee /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
$ cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
256
```

There one of these sizes is "default huge page size" which will be used in the auto-mounted /dev/huge pages. Changing the default size requires a reboot and is set via default_huge pagesz

You can check the current default size:

\$ grep Hugepagesize /proc/meminfo
Hugepagesize: 2048 kB

But there can be more than one at the same time one better check:

```
$ tail /sys/kernel/mm/hugepages/hugepages-*/nr_hugepages`
==> /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages <==</pre>
```

```
0
```

==> /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages <==
2</pre>

And even that could on bigger systems be further split per Numa node.

One can allocate huge pages at boot or runtime, but due to fragmentation there are no guarantees it works later. The kernel documentation lists details on both ways.

Huge pages need to be allocated by the kernel as mentioned above but to be consumable they also have to be mounted. By default *systemd* will make /dev/hugepages available for the default huge page size. Feel free to add more mount points if you need different sized. An overview can be queried with

\$ hugeadm --list-all-mounts
Mount Point Options
/dev/hugepages rw,relatime,pagesize=2M

A one-stop info for the overall huge page status of the system can be reported with

```
$ hugeadm --explain
```

Huge page usage in libvirt

With the above in place libvirt can map guest memory to huge pages. In a guest definition add the most simple form of

```
<memoryBacking>
<hugepages/>
</memoryBacking>
```

That will allocate the huge pages using the default huge page size from a autodetected mountpoint. For more control e.g. how memory is spread over Numa nodes or which page size to use check out the details at the libvirt doc.

Apparmor isolation

By default libvirt will spawn qemu guests using apparmor isolation for enhanced security. The apparmor rules for a guest will consist of multiple elements:

- a static part that all guests share => /etc/apparmor.d/abstractions/libvirt-qemu
- a dynamic part created at guest start time and modified on hotplug/unplug => /etc/apparmor.d/libvirt/libvirtf9533e35-6b63-45f5-96be-7cccc9696d5e.files

Of the above the former is provided and updated by the <code>libvirt-daemon</code> package and the latter is generated on guest start. None of the two should be manually edited. They will by default cover the vast majority of use cases and work fine. But there are certain cases where users either want to:

- further lock down the guest (e.g. by explicitly denying access that usually would be allowed)
- open up the guest isolation (most of the time this is needed if the setup on the local machine does not follow the commonly used paths.

To do so there are two files to do so. Both are local overrides which allow you to modify them without getting them clobbered or command file prompts on package upgrades.

- /etc/apparmor.d/local/abstractions/libvirt-qemu this will be applied to every guest. Therefore it is rather powerful, but also a rather blunt tool. It is a quite useful place thou to add additional deny rules.
- /etc/apparmor.d/local/usr.lib.libvirt.virt-aa-helper the above mentioned *dynamic part* that is individual per guest is generated by a tool called libvirt.virt-aa-helper. That is under apparmor isolation as well. This is most commonly used if you want to use uncommon paths as it allows to ahve those uncommon paths in the guest XML (see virsh edit) and have those paths rendered to the per-guest dynamic rules.

Sharing files between Host<->Guest

To be able to exchange data the memory of the guest has to be allocated as "shared" to do so you need to add the following to the guest config:

```
<memoryBacking>
<access mode='shared'/>
</memoryBacking>
```

For performance reasons (it helps virtiofs, but also is generally wise to consider) it is recommended to use huge pages which then would look like:

```
<memoryBacking>
<hugepages>
<page size='2048' unit='KiB'/>
</hugepages>
<access mode='shared'/>
</memoryBacking>
```

In the guest definition one then can add filesytem sections to specify host paths to share with the guest. The *target dir* is a bit special as it isn't really a directory, instead it is a *tag* that in the guest can be used to access this particular virtiofs instance.

```
<filesystem type='mount' accessmode='passthrough'>
<driver type='virtiofs'/>
<source dir='/var/guests/h-virtiofs'/>
<target dir='myfs'/>
</filesystem>
```

And in the guest this can now be used based on the tag myfs like:

```
$ sudo mount -t virtiofs myfs /mnt/
```

Compared to other Host/Guest file sharing options - commonly samba, nfs or 9p - virtiofs is usually much faster and also more compatible with usual file system semantics. For some extra compatibility in regard to filesystem semantics one can add:

```
<br/><binary xattr='on'>
<lock posix='on' flock='on'/>
</binary>
```

See the libvirt domain/filesytem documentation for further details on these.

Note:

While virtiofs works with >=20.10 (Groovy), with >=21.04 (Hirsute) it got much more comfortable, especially in small environments (no hard requirement to specify guest numa topology, no hard requirement to use huge pages). If needed to set up on 20.10 or just interested in those details - the libvirt knowledge-base about virtiofs hold more details about these.

Resources

- See the KVM home page for more details.
- For more information on libvirt see the libvirt home page
 - xml configuration of domains and storage being the most often used libvirt reference
- Another good resource is the Ubuntu Wiki KVM page.
- For basics how to assign VT-d devices to qemu/KVM, please see the linux-kvm page.

OpenStack

OpenStack is the most popular open source cloud computing platform that enables the management of distributed compute, network and storage resources in the data centre.

OpenStack Introduction

While the reference virtualisation stack, consisting of qemu/kvm and libvirt enables hardware virtualisation and the management of virtual machines (VMs) on a single host, in most of the cases compute, network and storage resources are distributed across multiple hosts in the data centre. This creates a challenge with centralised management of those resources, scheduling VMs, etc. OpenStack solves this problem by aggregating distributed pools of resources, allocating them to VMs on-demand and enabling automated VM provisioning through a self-service portal.

OpenStack consists of the following primary components:

- **Keystone** serves as an identity service, providing authentication and authorisation functions for the users and enabling multi-tenancy.
- Glance is an image service, responsible for uploading, managing and retrieving cloud images for VMs running on OpenStack.
- Nova is the primary compute engine of OpenStack, responsible for VM scheduling, creation and termination
- Neutron provides network connectivity between VMs, enabling multi-VM deployments.
- **Cinder** is a storage component that is responsible for provisioning, management and termination of persistent block devices.
- Swift is another storage component that provides a highly available and scalable object storage service

There are also many other OpenStack components and supporting services available in the OpenStack ecosystem, enabling more advanced functions, such as load balancing, secrets management, etc.

OpenStack Installation

The most straightforward way to get started with OpenStack on Ubuntu is to use MicroStack since the entire installation process requires only 2 commands and takes around 20 minutes.

Apart from MicroStack, multiple different installation methods for OpenStack on Ubuntu are available. Those include:

- OpenStack Charms
- OpenStack Ansible
- Manual Installation
- DevStack

Multipass

Multipass is the recommended method to create Ubuntu VMs on Ubuntu. It's designed for developers who want a fresh Ubuntu environment with a single command and works on Linux, Windows and macOS.

On Linux it's available as a snap:

sudo snap install multipass --beta --classic

Usage

. . .

Find available images

c · . .

\$ multipass find			
Image	Aliases	Version	Description
core	core16	20190424	Ubuntu Core 16
core18		20190213	Ubuntu Core 18
16.04	xenial	20190628	Ubuntu 16.04 LTS
18.04	bionic,lts	20190627.1	Ubuntu 18.04 LTS
18.10	cosmic	20190628	Ubuntu 18.10
19.04	disco	20190628	Ubuntu 19.04
daily:19.10	devel,eoan	20190623	Ubuntu 19.10

Launch a fresh instance of the current Ubuntu LTS

\$ multipass launch ubuntu Launching dancing-chipmunk... Downloading Ubuntu 18.04 LTS..... Launched: dancing chipmunk

Check out the running instances

\$ multipass list			
Name	State	IPv4	Release
dancing-chipmunk	RUNNING	10.125.174.247	Ubuntu 18.04 LTS
live-naiad	RUNNING	10.125.174.243	Ubuntu 18.04 LTS
snapcraft-asciinema	STOPPED		Ubuntu Snapcraft builder for Core 18

Learn more about the VM instance you just launched

<pre>\$ multipass</pre>	info dancing-chipmunk
Name:	dancing-chipmunk

State:	RUNNING
IPv4:	10.125.174.247
Release:	Ubuntu 18.04.1 LTS
Image hash:	19e9853d8267 (Ubuntu 18.04 LTS)
Load:	0.97 0.30 0.10
Disk usage:	1.1G out of 4.7G
Memory usage:	85.1M out of 985.4M

Connect to a running instance

\$ multipass shell dancing-chipmunk
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-42-generic x86_64)

. . .

Don't forget to logout (or Ctrl-D) or you may find yourself heading all the



way down the Inception levels...

Run commands inside an instance from outside

\$ multipass exec dancing-chipmunk -- lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 18.04.1 LTS
Release: 18.04
Codename: bionic

Stop an instance to save resources

\$ multipass stop dancing-chipmunk

Delete the instance

\$ multipass delete dancing-chipmunk

It will now show up as deleted:

Name	State	IPv4	Release
snapcraft-asciinema	STOPPED		Ubuntu Snapcraft builder for Core 18
dancing-chipmunk	DELETED		Not Available

And when you want to completely get rid of it:

\$ multipass purge

Integrate into the rest of your virtualization

You might have other virtualization already based on libvirt either through using the similar older uvtool already or through the common virt-manager.

You might for example want those guests to be on the same bridge to communicate to each other or you need access to the graphical output for some reason.

Fortunately it is possible to integrate this by using the libvirt backend of multipass

\$ sudo multipass set local.driver=libvirt

After that when you start a guest you can also access it via tools like virt-manager or virsh

\$ multipass launch ubuntu
Launched: engaged-amberjack

```
$ virsh list
Id Name
```

State

15	engaged-amberjack	running

Get help

multipass help
multipass help <command>

See the multipass documentation for more details.

Cloud images and uvtool

Introduction

With Ubuntu being one of the most used operating systems on many cloud platforms, the availability of stable and secure cloud images has become very important. As of 12.04 the utilization of cloud images outside of a cloud infrastructure has been improved. It is now possible to use those images to create a virtual machine without the need of a complete installation.

Creating virtual machines using uvtool

Starting with 14.04 LTS, a tool called uvtool greatly facilitates the task of generating virtual machines (VM) using the cloud images. uvtool provides a simple mechanism to synchronize cloud-images locally and use them to create new VMs in minutes.

Uvtool packages

The following packages and their dependencies will be required in order to use uvtool:

- uvtool
- uvtool-libvirt

To install uvtool, run:

\$ sudo apt -y install uvtool

This will install uvtool's main commands:

- uvt-simplestreams-libvirt
- uvt-kvm

Get the Ubuntu Cloud Image with uvt-simplestreams-libvirt

This is one of the major simplifications that uvtool brings. It is aware of where to find the cloud images so only one command is required to get a new cloud image. For instance, if you want to synchronize all cloud images for the amd64 architecture, the uvtool command would be:

\$ uvt-simplestreams-libvirt --verbose sync arch=amd64

After an amount of time required to download all the images from the Internet, you will have a complete set of cloud images stored locally. To see what has been downloaded use the following command:

```
$ uvt-simplestreams-libvirt query
release=bionic arch=amd64 label=daily (20191107)
release=focal arch=amd64 label=daily (20191029)
...
```

In the case where you want to synchronize only one specific cloud-image, you need to use the release= and arch= filters to identify which image needs to be synchronized.

\$ uvt-simplestreams-libvirt sync release=DISTRO-SHORT-CODENAME arch=amd64

Furthermore you can provide an alternative URL to fetch images from. A common case are the daily images which helps to get the very latest images or if you need access to the not yet released development release of Ubuntu.

\$ uvt-simplestreams-libvirt sync --source http://cloud-images.ubuntu.com/daily [... further options]

Create the VM using uvt-kvm

In order to connect to the virtual machine once it has been created, you must have a valid SSH key available for the Ubuntu user. If your environment does not have an SSH key, you can easily create one using the following command:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa.
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub.
The key fingerprint is:
4d:ba:5d:57:c9:49:ef:b5:ab:71:14:56:6e:2b:ad:9b ubuntu@DISTRO-SHORT-CODENAMES
The key's randomart image is:
+--[ RSA 2048]----+
I
                ..|
T
               0.=
               **|
T
         +
               0+=|
         S . ...=.|
0..+.|
.. 00|
*
                 Е
         ----+
+
```

To create of a new virtual machine using uvtool, run the following in a terminal:

\$ uvt-kvm create firsttest

This will create a VM named **firsttest** using the current LTS cloud image available locally. If you want to specify a release to be used to create the VM, you need to use the **release**= filter:

\$ uvt-kvm create secondtest release=DISTRO-SHORT-CODENAME

uvt-kvm wait can be used to wait until the creation of the VM has completed:

\$ uvt-kvm wait secondttest

Connect to the running VM

Once the virtual machine creation is completed, you can connect to it using SSH:

\$ uvt-kvm ssh secondtest

You can also connect to your VM using a regular SSH session using the IP address of the VM. The address can be queried using the following command:

```
$ uvt-kvm ip secondtest
192.168.122.199
$ ssh -i ~/.ssh/id_rsa ubuntu@192.168.122.199
[...]
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

ubuntu@secondtest:~\$

Get the list of running VMs

You can get the list of VMs running on your system with this command:

\$ uvt-kvm list
secondtest

Destroy your VM

Once you are done with your VM, you can destroy it with:

\$ uvt-kvm destroy secondtest

Note: other than libvirts destroy action this will by default also remove the associated virtual storage files.

More uvt-kvm options

The following options can be used to change some of the characteristics of the VM that you are creating:

- -memory : Amount of RAM in megabytes. Default: 512.
- -disk : Size of the OS disk in gigabytes. Default: 8.
- -cpu : Number of CPU cores. Default: 1.

Some other parameters will have an impact on the cloud-init configuration:

- -password password : Allow login to the VM using the Ubuntu account and this provided password.
- -run-script-once script_file : Run script_file as root on the VM the first time it is booted, but never again.
- _packages package_list : Install the comma-separated packages specified in package_list on first boot.

A complete description of all available modifiers is available in the manpage of uvt-kvm.

Resources

If you are interested in learning more, have questions or suggestions, please contact the Ubuntu Server Team at:

- IRC: #ubuntu-server on freenode
- Mailing list: ubuntu-server at lists.ubuntu.com

Virtualization tools

The virt-manager source contains not only virt-manager itself but also a collection of further helpful tools like virt-install, virt-clone and virt-viewer.

Virtual Machine Manager

The virt-manager package contains a graphical utility to manage local and remote virtual machines. To install virtmanager enter:

sudo apt install virt-manager

Since virt-manager requires a Graphical User Interface (GUI) environment it is recommended to be installed on a workstation or test machine instead of a production server. To connect to the local libvirt service enter:

virt-manager

You can connect to the libvirt service running on another host by entering the following in a terminal prompt:

virt-manager -c qemu+ssh://virtnode1.mydomain.com/system

Note

The above example assumes that SSH connectivity between the management system and the target system has already been configured, and uses SSH keys for authentication. SSH *keys* are needed because libvirt sends the password prompt to another process.

virt-manager guest lifecycle

When using virt-manager it is always important to know the context you look at.

The main window initially lists only the currently defined guests, you'll see their *name*, *state* and a small chart on *cpu* usage.



On that context there isn't much one can do except start/stop a guest.

But by double-clicking on a guest or by clicking the *open* button at the top one can see the guest itself. For a running guest that includes the guests main-console/virtual-screen output.



virt-manager-gui-showoutput 1594 $\times 894$ 364 KB

If you are deeper in the guest config a click in the top left onto "*show the graphical console*" will get you back to this output.

virt-manager guest modification

virt-manager provides a gui assisted way to edit guest definitions which can be handy. To do so the per-guest context view will at the top have "show virtual hardware details". Here a user can edit the virtual hardware of the guest which will under the cover alter the guest representation.

Virtual Machine Manager 🗸 🗠 🛛	focal-nvdimm on QEMU/KVM	~ ^ 0	
File Edit View Help	File Virtual Machine View Send Key		
😂 💻 Open 🕨 🔢 🕛 👻		×	
Name - CPU usage - QEMU/KVM Sho	vvirtual hardware details Details XML		
debian10.1 Shotoff	OS information Basic Details Performance Name: focal-ovdimm		
Shutoff	CPUS Memory UUID: 3bf3fe9c-9bf6-40a6-bbca-Sebb16837165		
Shotoff	Boot Options Status: Shutoff VittiO Disk 1 Title:		
	Virtuo Disk 2 Description:		
1	Mouse Keyboard		
	Display VNC Hypervisor Details Display Spice Hypervisor: KVM		
1. A A A A A A A A A A A A A A A A A A A	Serial 1 Architecture: x86_64 Emulator: /usr/bin/gemu-system-x86_64		
3	Video QXL Chipset: Q35 Firmware: BIOS		
	Controller USB 0		
	Controller PCIe 0		
State State	Add Hardware	Cancel Apply	

virt-manager-gui-edit 1406 $\times706$ 282 KB

The UI edit is limited to the features known and supported to that GUI feature. Not only does libvirt grow features faster than virt-manager can keep up - adding every feature would also overload the UI to the extend to be unusable. To strike a balance between the two there also is the XML view which can be reached via the "*edit libvirt XML*" button.

Virtual Machine Manager 🗸 🔿 🛛		focal-rwdimm on QEMU/KVM V 🗸 🔿 🛛	
File Edit View Help	File Virtual Machine View Send Key		
😫 🜉 Open 🕨 🔢 🕐 👻	🗏 🔲 🕨 II 🕚	- 6 X	
Name - CPU usage - QEMUKVM	Overview	Details XML	
deblast0.1 Shutoff	Performance	XM EGR UBWR XML Preference'. Only enable it if you know what you are doing.	
Shateff whentu20.04 whentu20.0	Boot Options	<pre><om_reador>restart</om_reador> <om_reador>reador> <om_reador>reador> <om_reador>reador> <om_reador>reador> <om_reador>reador> <om_reador>reador> <om_reador>reador> <om_reador> <om_reador< om_reador=""> <om_reador> <om_reador> <om_reador< om_reador=""> <om_reador< om_reador=""> <om_reador< om_reador=""> <om_reador< om_reado<="" om_reador<="" th=""></om_reador<></om_reador<></om_reador<></om_reador<></om_reador></om_reador></om_reador<></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></om_reador></pre>	
	VirtIO Disk 1		
	➢ NIC:64:6e:da INC:64:6e:da INC:64:6e:da		
	Keyboard OisplayVNC	 <disk device="disk" type="file"></disk>	
	Serial 1	<pre><driver nome"qemu"="" type"qcow27=""></driver> <source file='/var/lib/uttool/libvirt/images/focal-nvdimm-ds.qc <target dev="vdb" bus="virtio"/></pre>	
	Video QXL Controller USB 0	<pre><address bus="0x05" domain="0x0000" func<br="" slot="0x00" type="pci"> <controller index="0" model="nemu-xbci" type="usb"></controller></address></pre>	
	Controller SATA 0	<pre><address bus="0x02" domain="@x0000" func<br="" slot="0x00" type="pci"></address></pre>	
and the second second	Controller VirtiO Serial 0	<pre><controller index="0" type="sata"></controller></pre>	
	Add Hardware	Cancel Apply	

virt-manager-gui-XML1406 $\times706$ 340 KB

By default this will be read-only and you can see what the UI driven actions have changed, but one can allow RW access in this view in the *preferences*.

This is the same content that the virsh edit of the libvirt-client exposes.

Virtual Machine Viewer

The virt-viewer application allows you to connect to a virtual machine's console like virt-manager reduced to the GUI functionality. virt-viewer does require a Graphical User Interface (GUI) to interface with the virtual machine.

To install virt-viewer from a terminal enter:

sudo apt install virt-viewer

Once a virtual machine is installed and running you can connect to the virtual machine's console by using:

virt-viewer <guestname>

The UI will be a window representing the virtual screen of the guest, just like virt-manager above but without the extra buttons and features around it.


virt-viewer-gui-showoutput 1105×958 164 KB

Similar to virt-manager, virt-viewer can connect to a remote host using SSH with key authentication, as well:

virt-viewer -c qemu+ssh://virtnodel.mydomain.com/system <guestname>

Be sure to replace *web_devel* with the appropriate virtual machine name.

If configured to use a *bridged* network interface you can also setup SSH access to the virtual machine.

virt-install

virt-install is part of the virtinst package.

It can help installing classic ISO based systems and provides a CLI options for the most common options needed to do so. To install it, from a terminal prompt enter:

sudo apt install virtinst

There are several options available when using virt-install. For example:

virt-install -n web_devel -r 8192 \

```
--disk path=/home/doug/vm/web_devel.img,bus=virtio,size=50 \
```

-c focal-desktop-amd64.iso \

```
--network network=default,model=virtio \
```

```
--graphics vnc,listen=0.0.0.0 --noautoconsole -v --vcpus=4
```

There are many more arguments that can be found in the man page. Explaining those of the example above one by one:

• -n web_devel

the name of the new virtual machine will be web_devel in this example.

- -r 8192
- specifies the amount of memory the virtual machine will use in megabytes.
- -disk path=/home/doug/vm/web_devel.img,bus=virtio,size=50
- indicates the path to the virtual disk which can be a file, partition, or logical volume. In this example a file named web_devel.img in the current users directory, with a size of 50 gigabytes, and using virtio for the disk bus. Depending on the disk path, virt-install my need to be run with elevated privileges.
- c focal-desktop-amd64.iso
- file to be used as a virtual CDROM. The file can be either an ISO file or the path to the host's CDROM device. • –network

provides details related to the VM's network interface. Here the default network is used, and the interface model is configured for virtio.

- -graphics vnc,listen=0.0.0.0
 exports the guest's virtual console using VNC and on all host interfaces. Typically servers have no GUI, so another GUI based computer on the Local Area Network (LAN) can connect via VNC to complete the installation.
- –noautoconsole

will not automatically connect to the virtual machine's console.

- -v: creates a fully virtualized guest.
- -vcpus=4
- allocate 4 virtual CPUs.

After launching virt-install you can connect to the virtual machine's console either locally using a GUI (if your server has a GUI), or via a remote VNC client from a GUI-based computer.

virt-clone

The virt-clone application can be used to copy one virtual machine to another. For example:

virt-clone --auto-clone --original focal

Options used:

- -auto-clone: to have virt-clone come up with guest names and disk paths on its own
- -original: name of the virtual machine to copy

Also, use -d or --debug option to help troubleshoot problems with virt-clone.

Replace *focal* and with appropriate virtual machine names of your case.

Warning: please be aware that this is a full clone, therefore any sorts of secrets, keys and for example /etc/machine-id will be shared causing e.g. issues to security and anything that needs to identify the machine like DHCP. You most likely want to edit those afterwards and de-duplicate them as needed.

Resources

- See the KVM home page for more details.
- For more information on libvirt see the libvirt home page
- The Virtual Machine Manager site has more information on virt-manager development.

$\mathbf{L}\mathbf{X}\mathbf{D}$

LXD (pronounced lex-dee) is the lightervisor, or lightweight container hypervisor. LXC (lex-see) is a program which creates and administers "containers" on a local system. It also provides an API to allow higher level managers, such as LXD, to administer containers. In a sense, one could compare LXC to QEMU, while comparing LXD to libvirt.

The LXC API deals with a 'container'. The LXD API deals with 'remotes', which serve images and containers. This extends the LXC functionality over the network, and allows concise management of tasks like container migration and container image publishing.

LXD uses LXC under the covers for some container management tasks. However, it keeps its own container configuration information and has its own conventions, so that it is best not to use classic LXC commands by hand with LXD containers. This document will focus on how to configure and administer LXD on Ubuntu systems.

Online Resources

There is excellent documentation for getting started with LXD and an online server allowing you to try out LXD remotely. Stephane Graber also has an excellent blog series on LXD 2.0. Finally, there is great documentation on how to drive lxd using juju.

This document will offer an Ubuntu Server-specific view of LXD, focusing on administration.

Installation

LXD is pre-installed on Ubuntu Server cloud images. On other systems, the lxd package can be installed using:

sudo snap install lxd

This will install the self-contained LXD snap package.

Kernel preparation

In general, Ubuntu should have all the desired features enabled by default. One exception to this is that in order to enable swap accounting the boot argument swapaccount=1 must be set. This can be done by appending it to the $GRUB_CMDLINE_LINUX_DEFAULT=variable in /etc/default/grub, then running 'update-grub' as root and rebooting.$

Configuration

In order to use LXD, some basic settings need to be configured first. This is done by running lxd init, which will allow you to choose:

- Directory or ZFS container backend. If you choose ZFS, you can choose which block devices to use, or the size of a file to use as backing store.
- Availability over the network.
- A 'trust password' used by remote clients to vouch for their client certificate.

You must run 'lxd init' as root. 'lxc' commands can be run as any user who is a member of group lxd. If user joe is not a member of group 'lxd', you may run:

adduser joe lxd

as root to change it. The new membership will take effect on the next login, or after running newgrp lxd from an existing login.

For more information on server, container, profile, and device configuration, please refer to the definitive configuration provided with the source code, which can be found online.

Creating your first container

This section will describe the simplest container tasks.

Creating a container

Every new container is created based on either an image, an existing container, or a container snapshot. At install time, LXD is configured with the following image servers:

- ubuntu: this serves official Ubuntu server cloud image releases.
- ubuntu-daily: this serves official Ubuntu server cloud images of the daily development releases.
- images: this is a default-installed alias for images.linuxcontainers.org. This is serves classical lxc images built using the same images which the LXC 'download' template uses. This includes various distributions and minimal custom-made Ubuntu images. This is not the recommended server for Ubuntu images.

The command to create and start a container is

lxc launch remote:image containername

Images are identified by their hash, but are also aliased. The ubuntu remote knows many aliases such as 18.04 and bionic. A list of all images available from the Ubuntu Server can be seen using:

lxc image list ubuntu:

To see more information about a particular image, including all the aliases it is known by, you can use:

lxc image info ubuntu:bionic

You can generally refer to an Ubuntu image using the release name (bionic) or the release number (18.04). In addition, lts is an alias for the latest supported LTS release. To choose a different architecture, you can specify the desired architecture:

lxc image info ubuntu:lts/arm64

Now, let's start our first container:

lxc launch ubuntu:bionic b1

This will download the official current Bionic cloud image for your current architecture, then create a container named **b1** using that image, and finally start it. Once the command returns, you can see it using:

lxc list lxc info b1

and open a shell in it using:

lxc exec b1 -- bash

A convenient alias for the command above is:

lxc shell b1

The try-it page mentioned above gives a full synopsis of the commands you can use to administer containers.

Now that the **bionic** image has been downloaded, it will be kept in sync until no new containers have been created based on it for (by default) 10 days. After that, it will be deleted.

LXD Server Configuration

By default, LXD is socket activated and configured to listen only on a local UNIX socket. While LXD may not be running when you first look at the process listing, any LXC command will start it up. For instance:

lxc list

This will create your client certificate and contact the LXD server for a list of containers. To make the server accessible over the network you can set the http port using:

lxc config set core.https_address :8443

This will tell LXD to listen to port 8443 on all addresses.

Authentication

By default, LXD will allow all members of group 1xd to talk to it over the UNIX socket. Communication over the network is authorized using server and client certificates.

Before client c1 wishes to use remote r1, r1 must be registered using:

lxc remote add r1 r1.example.com:8443

The fingerprint of r1's certificate will be shown, to allow the user at c1 to reject a false certificate. The server in turn will verify that c1 may be trusted in one of two ways. The first is to register it in advance from any already-registered client, using:

lxc config trust add r1 certfile.crt

Now when the client adds r1 as a known remote, it will not need to provide a password as it is already trusted by the server.

The other step is to configure a 'trust password' with r1, either at initial configuration using lxd init, or after the fact using:

lxc config set core.trust_password PASSWORD

The password can then be provided when the client registers r1 as a known remote.

Backing store

LXD supports several backing stores. The recommended and the default backing store is zfs. If you already have a ZFS pool configured, you can tell LXD to use it during the lxd init procedure, otherwise a file-backed zpool will be created automatically. With ZFS, launching a new container is fast because the filesystem starts as a copy on write clone of the images' filesystem. Note that unless the container is privileged (see below) LXD will need to change

ownership of all files before the container can start, however this is fast and change very little of the actual filesystem data.

The other supported backing stores are described in detail in the Storage configuration section of the LXD documentation.

Container configuration

Containers are configured according to a set of profiles, described in the next section, and a set of container-specific configuration. Profiles are applied first, so that container specific configuration can override profile configuration.

Container configuration includes properties like the architecture, limits on resources such as CPU and RAM, security details including apparmor restriction overrides, and devices to apply to the container.

Devices can be of several types, including UNIX character, UNIX block, network interface, or disk. In order to insert a host mount into a container, a 'disk' device type would be used. For instance, to mount /opt in container cl at /opt, you could use:

lxc config device add c1 opt disk source=/opt path=opt

See:

lxc help config

for more information about editing container configurations. You may also use:

```
lxc config edit c1
```

to edit the whole of cl's configuration. Comments at the top of the configuration will show examples of correct syntax to help administrators hit the ground running. If the edited configuration is not valid when the editor is exited, then the editor will be restarted.

Profiles

Profiles are named collections of configurations which may be applied to more than one container. For instance, all containers created with lxc launch, by default, include the default profile, which provides a network interface eth0.

To mask a device which would be inherited from a profile but which should not be in the final container, define a device by the same name but of type 'none':

lxc config device add c1 eth1 none

Nesting

Containers all share the same host kernel. This means that there is always an inherent trade-off between features exposed to the container and host security from malicious containers. Containers by default are therefore restricted from features needed to nest child containers. In order to run lxc or lxd containers under a lxd container, the security.nesting feature must be set to true:

lxc config set container1 security.nesting true

Once this is done, container1 will be able to start sub-containers.

In order to run unprivileged (the default in LXD) containers nested under an unprivileged container, you will need to ensure a wide enough UID mapping. Please see the 'UID mapping' section below.

Limits

LXD supports flexible constraints on the resources which containers can consume. The limits come in the following categories:

- CPU: limit cpu available to the container in several ways.
- Disk: configure the priority of I/O requests under load
- RAM: configure memory and swap availability
- Network: configure the network priority under load
- Processes: limit the number of concurrent processes in the container.

For a full list of limits known to LXD, see the configuration documentation.

UID mappings and Privileged containers

By default, LXD creates unprivileged containers. This means that root in the container is a non-root UID on the host. It is privileged against the resources owned by the container, but unprivileged with respect to the host, making root in a container roughly equivalent to an unprivileged user on the host. (The main exception is the increased attack surface exposed through the system call interface)

Briefly, in an unprivileged container, 65536 UIDs are 'shifted' into the container. For instance, UID 0 in the container may be 100000 on the host, UID 1 in the container is 100001, etc, up to 165535. The starting value for UIDs and GIDs, respectively, is determined by the 'root' entry the /etc/subuid and /etc/subgid files. (See the subuid(5) man page.)

It is possible to request a container to run without a UID mapping by setting the security.privileged flag to true:

lxc config set c1 security.privileged true

Note however that in this case the root user in the container is the root user on the host.

Apparmor

LXD confines containers by default with an apparmor profile which protects containers from each other and the host from containers. For instance this will prevent root in one container from signaling root in another container, even though they have the same uid mapping. It also prevents writing to dangerous, un-namespaced files such as many sysctls and <code>/proc/sysrq-trigger</code>.

If the apparmor policy for a container needs to be modified for a container c1, specific apparmor policy lines can be added in the raw.apparmor configuration key.

Seccomp

All containers are confined by a default seccomp policy. This policy prevents some dangerous actions such as forced unounts, kernel module loading and unloading, kexec, and the open_by_handle_at system call. The seccomp configuration cannot be modified, however a completely different seccomp policy – or none – can be requested using raw.lxc (see below).

Raw LXC configuration

LXD configures containers for the best balance of host safety and container usability. Whenever possible it is highly recommended to use the defaults, and use the LXD configuration keys to request LXD to modify as needed. Sometimes, however, it may be necessary to talk to the underlying lxc driver itself. This can be done by specifying LXC configuration items in the 'raw.lxc' LXD configuration key. These must be valid items as documented in the lxc.container.conf(5) manual page.

Snapshots

Containers can be renamed and live-migrated using the lxc move command:

lxc move c1 final-beta

They can also be snapshotted:

lxc snapshot c1 YYYY-MM-DD

Later changes to c1 can then be reverted by restoring the snapshot:

lxc restore u1 YYYY-MM-DD

New containers can also be created by copying a container or snapshot:

lxc copy u1/YYY-MM-DD testcontainer

Publishing images

When a container or container snapshot is ready for consumption by others, it can be published as a new image using;

lxc publish u1/YYYY-MM-DD --alias foo-2.0

The published image will be private by default, meaning that LXD will not allow clients without a trusted certificate to see them. If the image is safe for public viewing (i.e. contains no private information), then the 'public' flag can be set, either at publish time using

lxc publish u1/YYYY-MM-DD --alias foo-2.0 public=true

or after the fact using

lxc image edit foo-2.0

and changing the value of the public field.

Image export and import

Image can be exported as, and imported from, tarballs:

lxc image export foo-2.0 foo-2.0.tar.gz
lxc image import foo-2.0.tar.gz --alias foo-2.0 --public

Troubleshooting

To view debug information about LXD itself, on a systemd based host use

journalctl -u lxd

Container logfiles for container c1 may be seen using:

lxc info c1 --show-log

The configuration file which was used may be found under /var/log/lxd/cl/lxc.conf while apparmor profiles can be found in /var/lib/lxd/security/apparmor/profiles/cl and seccomp profiles in /var/lib/lxd/security/seccomp/cl.

\mathbf{LXC}

Containers are a lightweight virtualization technology. They are more akin to an enhanced chroot than to full virtualization like Qemu or VMware, both because they do not emulate hardware and because containers share the same operating system as the host. Containers are similar to Solaris zones or BSD jails. Linux-vserver and OpenVZ are two pre-existing, independently developed implementations of containers-like functionality for Linux. In fact, containers came about as a result of the work to upstream the vserver and OpenVZ functionality.

There are two user-space implementations of containers, each exploiting the same kernel features. Libvirt allows the use of containers through the LXC driver by connecting to lxc:///. This can be very convenient as it supports the same usage as its other drivers. The other implementation, called simply 'LXC', is not compatible with libvirt, but is more flexible with more userspace tools. It is possible to switch between the two, though there are peculiarities which can cause confusion.

In this document we will mainly describe the *lxc* package. Use of libvirt-lxc is not generally recommended due to a lack of Apparmor protection for libvirt-lxc containers.

In this document, a container name will be shown as CN, C1, or C2.

Installation

The lxc package can be installed using

sudo apt install lxc

This will pull in the required and recommended dependencies, as well as set up a network bridge for containers to use. If you wish to use unprivileged containers, you will need to ensure that users have sufficient allocated subuids and subgids, and will likely want to allow users to connect containers to a bridge (see *Basic unprivileged usage* below).

Basic usage

LXC can be used in two distinct ways - privileged, by running the lxc commands as the root user; or unprivileged, by running the lxc commands as a non-root user. (The starting of unprivileged containers by the root user is possible, but not described here.) Unprivileged containers are more limited, for instance being unable to create device nodes or mount block-backed filesystems. However they are less dangerous to the host, as the root UID in the container is mapped to a non-root UID on the host.

Basic privileged usage

To create a privileged container, you can simply do:

```
sudo lxc-create --template download --name u1
```

or, abbreviated

sudo lxc-create -t download -n u1

This will interactively ask for a container root filesystem type to download – in particular the distribution, release, and architecture. To create the container non-interactively, you can specify these values on the command line:

sudo lxc-create -t download -n ul -- --dist ubuntu --release DISTRO-SHORT-CODENAME --arch amd64
or

sudo lxc-create -t download -n ul -- -d ubuntu -r DISTRO-SHORT-CODENAME -a amd64

You can now use lxc-ls to list containers, lxc-info to obtain detailed container information, lxc-start to start and lxc-stop to stop the container. lxc-attach and lxc-console allow you to enter a container, if ssh is not an option. lxc-destroy removes the container, including its rootfs. See the manual pages for more information on each command. An example session might look like:

```
sudo lxc-ls --fancy
sudo lxc-start --name ul --daemon
sudo lxc-info --name ul
sudo lxc-stop --name ul
sudo lxc-destroy --name ul
```

User namespaces

Unprivileged containers allow users to create and administer containers without having any root privilege. The feature underpinning this is called user namespaces. User namespaces are hierarchical, with privileged tasks in a parent namespace being able to map its ids into child namespaces. By default every task on the host runs in the initial user namespace, where the full range of ids is mapped onto the full range. This can be seen by looking at /proc/self/uid_map and /proc/self/gid_map, which both will show 0 0 4294967295 when read from the initial user namespace. As of Ubuntu 14.04, when new users are created they are by default offered a range of UIDs. The list of assigned ids can be seen in the files /etc/subuid and /etc/subgid See their respective manpages for more information. Subuids and subgids are by convention started at id 100000 to avoid conflicting with system users.

If a user was created on an earlier release, it can be granted a range of ids using usermod, as follows:

sudo usermod -v 100000-200000 -w 100000-200000 user1

The programs newuidmap and newgidmap are setuid-root programs in the uidmap package, which are used internally by lxc to map subuids and subgids from the host into the unprivileged container. They ensure that the user only maps ids which are authorized by the host configuration.

Basic unprivileged usage

To create unprivileged containers, a few first steps are needed. You will need to create a default container configuration file, specifying your desired id mappings and network setup, as well as configure the host to allow the unprivileged user to hook into the host network. The example below assumes that your mapped user and group id ranges are 100000–165536. Check your actual user and group id ranges and modify the example accordingly:

```
grep $USER /etc/subuid
grep $USER /etc/subgid
```

mkdir -p ~/.config/lxc echo "lxc.id_map = u 0 100000 65536" > ~/.config/lxc/default.conf echo "lxc.id_map = g 0 100000 65536" >> ~/.config/lxc/default.conf echo "lxc.network.type = veth" >> ~/.config/lxc/default.conf echo "lxc.network.link = lxcbr0" >> ~/.config/lxc/default.conf echo "\$USER veth lxcbr0 2" | sudo tee -a /etc/lxc/lxc-usernet

After this, you can create unprivileged containers the same way as privileged ones, simply without using sudo.

```
lxc-create -t download -n ul -- -d ubuntu -r DISTRO-SHORT-CODENAME -a amd64
lxc-start -n ul -d
lxc-attach -n ul
lxc-stop -n ul
lxc-destroy -n ul
```

Nesting

In order to run containers inside containers - referred to as nested containers - two lines must be present in the parent container configuration file:

lxc.mount.auto = cgroup
lxc.aa_profile = lxc-container-default-with-nesting

The first will cause the cgroup manager socket to be bound into the container, so that lxc inside the container is able to administer cgroups for its nested containers. The second causes the container to run in a looser Apparmor policy which allows the container to do the mounting required for starting containers. Note that this policy, when used with a privileged container, is much less safe than the regular policy or an unprivileged container. See the *Apparmor* section for more information.

Global configuration

The following configuration files are consulted by LXC. For privileged use, they are found under /etc/lxc, while for unprivileged use they are under ~/.config/lxc.

- lxc.conf may optionally specify alternate values for several lxc settings, including the lxcpath, the default configuration, cgroups to use, a cgroup creation pattern, and storage backend settings for lvm and zfs.
- default.conf specifies configuration which every newly created container should contain. This usually contains at least a network section, and, for unprivileged users, an id mapping section
- lxc-usernet.conf specifies how unprivileged users may connect their containers to the host-owned network.

lxc.conf and default.conf are both under /etc/lxc and \$HOME/.config/lxc, while lxc-usernet.conf is only host-wide.

By default, containers are located under /var/lib/lxc for the root user.

Networking

By default LXC creates a private network namespace for each container, which includes a layer 2 networking stack. Containers usually connect to the outside world by either having a physical NIC or a veth tunnel endpoint passed into the container. LXC creates a NATed bridge, lxcbr0, at host startup. Containers created using the default configuration will have one veth NIC with the remote end plugged into the lxcbr0 bridge. A NIC can only exist in one namespace at a time, so a physical NIC passed into the container is not usable on the host.

It is possible to create a container without a private network namespace. In this case, the container will have access to the host networking like any other application. Note that this is particularly dangerous if the container is running a distribution with upstart, like Ubuntu, since programs which talk to init, like shutdown, will talk over the abstract Unix domain socket to the host's upstart, and shut down the host.

To give containers on lxcbr0 a persistent ip address based on domain name, you can write entries to /etc/lxc/dnsmasq.conf like:

dhcp-host=lxcmail,10.0.3.100
dhcp-host=ttrss,10.0.3.101

If it is desirable for the container to be publicly accessible, there are a few ways to go about it. One is to use **iptables** to forward host ports to the container, for instance

Then, specify the host's bridge in the container configuration file in place of lxcbr0, for instance

lxc.network.type = veth
lxc.network.link = br0

Finally, you can ask LXC to use macvlan for the container's NIC. Note that this has limitations and depending on configuration may not allow the container to talk to the host itself. Therefore the other two options are preferred and more commonly used.

There are several ways to determine the ip address for a container. First, you can use lxc-ls --fancy which will print the ip addresses for all running containers, or lxc-info -i -H -n C1 which will print C1's ip address. If dnsmasq is installed on the host, you can also add an entry to /etc/dnsmasq.conf as follows

server=/lxc/10.0.3.1

after which dnsmasq will resolve C1.lxc locally, so that you can do:

ping Cl ssh Cl

For more information, see the lxc.conf(5) manpage as well as the example network configurations under /usr/share/doc/lxc/examples/.

LXC startup

LXC does not have a long-running daemon. However it does have three upstart jobs.

- /etc/init/lxc-net.conf: is an optional job which only runs if /etc/default/lxc-net specifies USE_LXC_BRIDGE (true by default). It sets up a NATed bridge for containers to use.
- /etc/init/lxc.conf loads the lxc apparmor profiles and optionally starts any autostart containers. The autostart containers will be ignored if LXC_AUTO (true by default) is set to true in /etc/default/lxc. See the lxc-autostart manual page for more information on autostarted containers.
- /etc/init/lxc-instance.conf is used by /etc/init/lxc.conf to autostart a container.

Backing Stores

LXC supports several backing stores for container root filesystems. The default is a simple directory backing store, because it requires no prior host customization, so long as the underlying filesystem is large enough. It also requires no root privilege to create the backing store, so that it is seamless for unprivileged use. The rootfs for a privileged directory backed container is located (by default) under /var/lib/lxc/Cl/rootfs, while the rootfs for an unprivileged container is under ~/.local/share/lxc/Cl/rootfs. If a custom lxcpath is specified in lxc.system.com, then the container rootfs will be under \$lxcpath/Cl/rootfs.

A snapshot clone C2 of a directory backed container C1 becomes an overlayfs backed container, with a rootfs called overlayfs:/var/lib/lxc/C1/rootfs:/var/lib/lxc/C2/delta0. Other backing store types include loop, btrfs, LVM and zfs.

A btrfs backed container mostly looks like a directory backed container, with its root filesystem in the same location. However, the root filesystem comprises a subvolume, so that a snapshot clone is created using a subvolume snapshot.

The root filesystem for an LVM backed container can be any separate LV. The default VG name can be specified in lxc.conf. The filesystem type and size are configurable per-container using lxc-create.

The rootfs for a zfs backed container is a separate zfs filesystem, mounted under the traditional /var/lib/lxc/Cl/rootfs location. The zfsroot can be specified at lxc-create, and a default can be specified in lxc.system.conf.

More information on creating containers with the various backing stores can be found in the lxc-create manual page.

Templates

Creating a container generally involves creating a root filesystem for the container. lxc-create delegates this work to *templates*, which are generally per-distribution. The lxc templates shipped with lxc can be found under /usr/share/lxc/templates, and include templates to create Ubuntu, Debian, Fedora, Oracle, centos, and gentoo containers among others.

Creating distribution images in most cases requires the ability to create device nodes, often requires tools which are not available in other distributions, and usually is quite time-consuming. Therefore lxc comes with a special *download* template, which downloads pre-built container images from a central lxc server. The most important use case is to allow simple creation of unprivileged containers by non-root users, who could not for instance easily run the debootstrap command.

When running lxc-create, all options which come after – are passed to the template. In the following command, –*name*, –*template* and –*bdev* are passed to lxc-create, while –*release* is passed to the template:

lxc-create --template ubuntu --name c1 --bdev loop -- --release DISTRO-SHORT-CODENAME

You can obtain help for the options supported by any particular container by passing -help and the template name to lxc-create. For instance, for help with the download template,

lxc-create --template download --help

Autostart

LXC supports marking containers to be started at system boot. Prior to Ubuntu 14.04, this was done using symbolic links under the directory /etc/lxc/auto. Starting with Ubuntu 14.04, it is done through the container configuration files. An entry

```
lxc.start.auto = 1
lxc.start.delay = 5
```

would mean that the container should be started at boot, and the system should wait 5 seconds before starting the next container. LXC also supports ordering and grouping of containers, as well as reboot and shutdown by autostart groups. See the manual pages for lxc-autostart and lxc.container.conf for more information.

Apparmor

LXC ships with a default Apparmor profile intended to protect the host from accidental misuses of privilege inside the container. For instance, the container will not be able to write to /proc/sysrq-trigger or to most /sys files.

The usr.bin.lxc-start profile is entered by running lxc-start. This profile mainly prevents lxc-start from mounting new filesystems outside of the container's root filesystem. Before executing the container's init, LXC requests a switch to the container's profile. By default, this profile is the lxc-container-default policy which is defined in /etc/apparmor.d/lxc/lxc-default. This profile prevents the container from accessing many dangerous paths, and from mounting most filesystems.

Programs in a container cannot be further confined - for instance, MySQL runs under the container profile (protecting the host) but will not be able to enter the MySQL profile (to protect the container).

lxc-execute does not enter an Apparmor profile, but the container it spawns will be confined.

Customizing container policies

If you find that lxc-start is failing due to a legitimate access which is being denied by its Apparmor policy, you can disable the lxc-start profile by doing:

sudo apparmor_parser -R /etc/apparmor.d/usr.bin.lxc-start
sudo ln -s /etc/apparmor.d/usr.bin.lxc-start /etc/apparmor.d/disabled/

This will make *lxc-start* run unconfined, but continue to confine the container itself. If you also wish to disable confinement of the container, then in addition to disabling the *usr.bin.lxc-start* profile, you must add:

lxc.aa_profile = unconfined

to the container's configuration file.

LXC ships with a few alternate policies for containers. If you wish to run containers inside containers (nesting), then you can use the lxc-container-default-with-nesting profile by adding the following line to the container configuration file

lxc.aa_profile = lxc-container-default-with-nesting

If you wish to use libvirt inside containers, then you will need to edit that policy (which is defined in /etc/apparmor.d/lxc/lxc-default-with-nesting) by uncommenting the following line:

mount fstype=cgroup -> /sys/fs/cgroup/**,

and re-load the policy.

Note that the nesting policy with privileged containers is far less safe than the default policy, as it allows containers to re-mount /sys and /proc in nonstandard locations, bypassing apparmor protections. Unprivileged containers do not have this drawback since the container root cannot write to root-owned proc and sys files.

Another profile shipped with lxc allows containers to mount block filesystem types like ext4. This can be useful in some cases like maas provisioning, but is deemed generally unsafe since the superblock handlers in the kernel have not been audited for safe handling of untrusted input.

If you need to run a container in a custom profile, you can create a new profile under /etc/apparmor.d/lxc/. Its name must start with lxc- in order for lxc-start to be allowed to transition to that profile. The lxc-default profile includes the re-usable abstractions file /etc/apparmor.d/abstractions/lxc/container-base. An easy way to start a new profile therefore is to do the same, then add extra permissions at the bottom of your policy.

After creating the policy, load it using:

sudo apparmor_parser -r /etc/apparmor.d/lxc-containers

The profile will automatically be loaded after a reboot, because it is sourced by the file /etc/apparmor.d/lxc-containers. Finally, to make container CN use this new lxc-CN-profile, add the following line to its configuration file:

```
lxc.aa_profile = lxc-CN-profile
```

Control Groups

Control groups (cgroups) are a kernel feature providing hierarchical task grouping and per-cgroup resource accounting and limits. They are used in containers to limit block and character device access and to freeze (suspend) containers. They can be further used to limit memory use and block i/o, guarantee minimum cpu shares, and to lock containers to specific cpus. By default, a privileged container CN will be assigned to a cgroup called /lxc/CN. In the case of name conflicts (which can occur when using custom lxcpaths) a suffix "-n", where n is an integer starting at 0, will be appended to the cgroup name.

By default, a privileged container CN will be assigned to a cgroup called CN under the cgroup of the task which started the container, for instance /usr/1000.user/1.session/CN. The container root will be given group ownership of the directory (but not all files) so that it is allowed to create new child cgroups.

As of Ubuntu 14.04, LXC uses the cgroup manager (cgmanager) to administer cgroups. The cgroup manager receives D-Bus requests over the Unix socket /sys/fs/cgroup/cgmanager/sock. To facilitate safe nested containers, the line

lxc.mount.auto = cgroup

can be added to the container configuration causing the /sys/fs/cgroup/cgmanager directory to be bindmounted into the container. The container in turn should start the cgroup management proxy (done by default if the cgmanager package is installed in the container) which will move the /sys/fs/cgroup/cgmanager directory to /sys/fs/cgroup/cgmanager.lower, then start listening for requests to proxy on its own socket /sys/fs/cgroup/cgmanager/sock. The host cgmanager will ensure that nested containers cannot escape their assigned cgroups or make requests for which they are not authorized.

Cloning

For rapid provisioning, you may wish to customize a canonical container according to your needs and then make multiple copies of it. This can be done with the lxc-clone program.

Clones are either snapshots or copies of another container. A copy is a new container copied from the original, and takes as much space on the host as the original. A snapshot exploits the underlying backing store's snapshotting ability to make a copy-on-write container referencing the first. Snapshots can be created from btrfs, LVM, zfs, and directory backed containers. Each backing store has its own peculiarities - for instance, LVM containers which are not thinpool-provisioned cannot support snapshots of snapshots; zfs containers with snapshots cannot be removed until all snapshots are released; LVM containers must be more carefully planned as the underlying filesystem may not support growing; btrfs does not suffer any of these shortcomings, but suffers from reduced fsync performance causing dpkg and apt to be slower.

Snapshots of directory-packed containers are created using the overlay filesystem. For instance, a privileged directorybacked container C1 will have its root filesystem under /var/lib/lxc/C1/rootfs. A snapshot clone of C1 called C2 will be started with C1's rootfs mounted readonly under /var/lib/lxc/C2/delta0. Importantly, in this case C1 should not be allowed to run or be removed while C2 is running. It is advised instead to consider C1 a *canonical* base container, and to only use its snapshots.

Given an existing container called C1, a copy can be created using:

sudo lxc-clone -o C1 -n C2

A snapshot can be created using:

sudo lxc-clone -s -o C1 -n C2

See the lxc-clone manpage for more information.

Snapshots

To more easily support the use of snapshot clones for iterative container development, LXC supports *snapshots*. When working on a container C1, before making a potentially dangerous or hard-to-revert change, you can create a snapshot

sudo lxc-snapshot -n C1

which is a snapshot-clone called 'snap0' under /var/lib/lxcsnaps or \$HOME/.local/share/lxcsnaps. The next snapshot will be called 'snap1', etc. Existing snapshots can be listed using lxc-snapshot -L -n Cl, and a snapshot can be restored - erasing the current Cl container - using lxc-snapshot -r snap1 -n Cl. After the restore command, the snap1 snapshot continues to exist, and the previous Cl is erased and replaced with the snap1 snapshot.

Snapshots are supported for btrfs, lvm, zfs, and overlayfs containers. If lxc-snapshot is called on a directory-backed container, an error will be logged and the snapshot will be created as a copy-clone. The reason for this is that if the user creates an overlayfs snapshot of a directory-backed container and then makes changes to the directory-backed container, then the original container changes will be partially reflected in the snapshot. If snapshots of a directory backed container C1 are desired, then an overlayfs clone of C1 should be created, C1 should not be touched again, and the overlayfs clone can be edited and snapshotted at will, as such

lxc-clone -s -o C1 -n C2
lxc-start -n C2 -d # make some changes

lxc-stop -n C2
lxc-snapshot -n C2
lxc-start -n C2 # etc

Ephemeral Containers

While snapshots are useful for longer-term incremental development of images, ephemeral containers utilize snapshots for quick, single-use throwaway containers. Given a base container C1, you can start an ephemeral container using

lxc-start-ephemeral -o C1

The container begins as a snapshot of C1. Instructions for logging into the container will be printed to the console. After shutdown, the ephemeral container will be destroyed. See the lxc-start-ephemeral manual page for more options.

Lifecycle management hooks

Beginning with Ubuntu 12.10, it is possible to define hooks to be executed at specific points in a container's lifetime:

- Pre-start hooks are run in the host's namespace before the container ttys, consoles, or mounts are up. If any mounts are done in this hook, they should be cleaned up in the post-stop hook.
- Pre-mount hooks are run in the container's namespaces, but before the root filesystem has been mounted. Mounts done in this hook will be automatically cleaned up when the container shuts down.
- Mount hooks are run after the container filesystems have been mounted, but before the container has called pivot_root to change its root filesystem.
- Start hooks are run immediately before executing the container's init. Since these are executed after pivoting into the container's filesystem, the command to be executed must be copied into the container's filesystem.
- Post-stop hooks are executed after the container has been shut down.

If any hook returns an error, the container's run will be aborted. Any *post-stop* hook will still be executed. Any output generated by the script will be logged at the debug priority.

Please see the lxc.container.conf(5) manual page for the configuration file format with which to specify hooks. Some sample hooks are shipped with the lxc package to serve as an example of how to write and use such hooks.

Consoles

Containers have a configurable number of consoles. One always exists on the container's /dev/console. This is shown on the terminal from which you ran lxc-start, unless the -d option is specified. The output on /dev/console can be redirected to a file using the -c console-file option to lxc-start. The number of extra consoles is specified by the lxc.tty variable, and is usually set to 4. Those consoles are shown on /dev/ttyN (for $1 \le N \le 4$). To log into console 3 from the host, use:

sudo lxc-console -n container -t 3

or if the -t N option is not specified, an unused console will be automatically chosen. To exit the console, use the escape sequence Ctrl-a q. Note that the escape sequence does not work in the console resulting from lxc-start without the -d option.

Each container console is actually a Unix98 pty in the host's (not the guest's) pty mount, bind-mounted over the guest's /dev/ttyN and /dev/console. Therefore, if the guest unmounts those or otherwise tries to access the actual character device 4:N, it will not be serving getty to the LXC consoles. (With the default settings, the container will not be able to access that character device and getty will therefore fail.) This can easily happen when a boot script blindly mounts a new /dev.

Troubleshooting

Logging

If something goes wrong when starting a container, the first step should be to get full logging from LXC:

sudo lxc-start -n C1 -l trace -o debug.out

This will cause lxc to log at the most verbose level, trace, and to output log information to a file called 'debug.out'. If the file debug.out already exists, the new log information will be appended.

Monitoring container status

Two commands are available to monitor container state changes. lxc-monitor monitors one or more containers for any state changes. It takes a container name as usual with the -n option, but in this case the container name can be a

posix regular expression to allow monitoring desirable sets of containers. lxc-monitor continues running as it prints container changes. lxc-wait waits for a specific state change and then exits. For instance,

sudo lxc-monitor -n cont[0-5]*

would print all state changes to any containers matching the listed regular expression, whereas

```
sudo lxc-wait -n cont1 -s 'STOPPED|FROZEN'
```

will wait until container cont1 enters state STOPPED or state FROZEN and then exit.

Attach

As of Ubuntu 14.04, it is possible to attach to a container's namespaces. The simplest case is to simply do

```
sudo lxc-attach -n C1
```

which will start a shell attached to C1's namespaces, or, effectively inside the container. The attach functionality is very flexible, allowing attaching to a subset of the container's namespaces and security context. See the manual page for more information.

Container init verbosity

If LXC completes the container startup, but the container init fails to complete (for instance, no login prompt is shown), it can be useful to request additional verbosity from the init process. For an upstart container, this might be:

sudo lxc-start -n C1 /sbin/init loglevel=debug

You can also start an entirely different program in place of init, for instance

sudo lxc-start -n C1 /bin/bash
sudo lxc-start -n C1 /bin/sleep 100
sudo lxc-start -n C1 /bin/cat /proc/1/status

LXC API

Most of the LXC functionality can now be accessed through an API exported by liblxc for which bindings are available in several languages, including Python, lua, ruby, and go.

Below is an example using the python bindings (which are available in the python3-lxc package) which creates and starts a container, then waits until it has been shut down:

```
# sudo python3
Python 3.2.3 (default, Aug 28 2012, 08:26:03)
[GCC 4.7.1 20120814 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import lxc
___main__:1: Warning: The python-lxc API isn't yet stable and may change at any point in the future.
>>> c=lxc.Container("C1")
>>> c.create("ubuntu")
True
>>> c.start()
True
>>> c.wait("STOPPED")
True
```

Security

A namespace maps ids to resources. By not providing a container any id with which to reference a resource, the resource can be protected. This is the basis of some of the security afforded to container users. For instance, IPC namespaces are completely isolated. Other namespaces, however, have various *leaks* which allow privilege to be inappropriately exerted from a container into another container or to the host.

By default, LXC containers are started under a Apparmor policy to restrict some actions. The details of AppArmor integration with lxc are in section *Apparmor*. Unprivileged containers go further by mapping root in the container to an unprivileged host UID. This prevents access to /proc and /sys files representing host resources, as well as any other files owned by root on the host.

Exploitable system calls

It is a core container feature that containers share a kernel with the host. Therefore if the kernel contains any exploitable system calls the container can exploit these as well. Once the container controls the kernel it can fully control any resource known to the host.

In general to run a full distribution container a large number of system calls will be needed. However for application containers it may be possible to reduce the number of available system calls to only a few. Even for system containers running a full distribution security gains may be had, for instance by removing the 32-bit compatibility system calls in a 64-bit container. See the lxc.container.conf manual page for details of how to configure a container to use seccomp. By default, no seccomp policy is loaded.

Resources

- The DeveloperWorks article LXC: Linux container tools was an early introduction to the use of containers.
- The Secure Containers Cookbook demonstrated the use of security modules to make containers more secure.
- The upstream LXC project is hosted at linuxcontainers.org.

Introduction to High Availability

A definition of High Availability Clusters from Wikipedia:

High Availability Clusters

High-availability clusters (also known as **HA clusters**, **fail-over clusters** or **Metroclusters Active/Active**) are groups of computers that support server applications that can be reliably utilized with a minimum amount of down-time.

They operate by using high availability software to harness redundant computers in groups or clusters that provide continued service when system components fail.

Without clustering, if a server running a particular application crashes, the application will be unavailable until the crashed server is fixed. HA clustering remedies this situation by detecting hardware/software faults, and immediately restarting the application on another system without requiring administrative intervention, a process known as failover.

As part of this process, clustering software may configure the node before starting the application on it. For example, appropriate file systems may need to be imported and mounted, network hardware may have to be configured, and some supporting applications may need to be running as well.

HA clusters are often used for critical databases, file sharing on a network, business applications, and customer services such as electronic commerce websites.

High Availability Cluster Heartbeat

HA cluster implementations attempt to build redundancy into a cluster to eliminate single points of failure, including multiple network connections and data storage which is redundantly connected via storage area networks.

HA clusters usually use a heartbeat private network connection which is used to monitor the health and status of each node in the cluster. One subtle but serious condition all clustering software must be able to handle is split-brain, which occurs when all of the private links go down simultaneously, but the cluster nodes are still running.

If that happens, each node in the cluster may mistakenly decide that every other node has gone down and attempt to start services that other nodes are still running. Having duplicate instances of services may cause data corruption on the shared storage.

High Availability Cluster Quorum

HA clusters often also use quorum witness storage (local or cloud) to avoid this scenario. A witness device cannot be shared between two halves of a split cluster, so in the event that all cluster members cannot communicate with each other (e.g., failed heartbeat), if a member cannot access the witness, it cannot become active.

Example



Fencing

Fencing protects your data from being corrupted, and your application from becoming unavailable, due to unintended concurrent access by rogue nodes.

Just because a node is unresponsive doesn't mean it has stopped accessing your data. The only way to be 100% sure that your data is safe, is to use fencing to ensure that the node is truly offline before allowing the data to be accessed from another node.

Fencing also has a role to play in the event that a clustered service cannot be stopped. In this case, the cluster uses fencing to force the whole node offline, thereby making it safe to start the service elsewhere. The most popular example of fencing is cutting a host's power.

Key Benefits:

- Active countermeasure taken by a functioning host to isolate a misbehaving (usually dead) host from shared data.
- **MOST CRITICAL** part of a cluster utilizing SAN or other shared storage technology (*Ubuntu HA Clusters* can only be supported if the fencing mechanism is configured).
- Required by OCFS2, GFS2, cLVMd (before Ubuntu 20.04), lvmlockd (from 20.04 and beyond).

Linux High Availability Projects

There are many upstream high availability related projects that are included in Ubuntu Linux. This section will describe the most important ones.

The following packages are present in latest Ubuntu LTS release:

Ubuntu HA Core Packages

Packages in this list are supported just like any other package available in [main] repository would be.

Package	URL	
libqb	Ubuntu	Upstream
kronosnet	Ubuntu	Upstream
corosync	Ubuntu	Upstream
pacemaker	Ubuntu	Upstream
resource-agents	Ubuntu	Upstream
fence-agents	Ubuntu	Upstream
crmsh	Ubuntu	Upstream
cluster-glue	Ubuntu	Upstream
drbd-utils	Ubuntu	Upstream
dlm	Ubuntu	Upstream
gfs2-utils	Ubuntu	Upstream
keepalived	Ubuntu	Upstream

- **libqb** Library which provides a set of high performance client-server reusable features. It offers high performance logging, tracing, IPC and poll. Its initial features were spun off the *Corosync* cluster communication suite to make them accessible for other projects.
- **Kronosnet** Kronosnet, often referred to as knet, is a network abstraction layer designed for High Availability. Corosync uses Kronosnet to provide multiple networks for its interconnect (replacing the old Totem Redundant Ring Protocol) and add support for some more features like interconnect network hot-plug.
- **Corosync** or *Cluster Membership Layer*, provides reliable messaging, membership and quorum information about the cluster. Currently, Pacemaker supports Corosync as this layer.
- **Pacemaker** or *Cluster Resource Manager*, provides the brain that processes and reacts to events that occur in the cluster. Events might be: nodes joining or leaving the cluster, resource events caused by failures, maintenance, or scheduled activities. To achieve the desired availability, Pacemaker may start and stop resources and fence nodes.
- **Resource Agents** Scripts or operating system components that start, stop or monitor resources, given a set of resource parameters. These provide a uniform interface between pacemaker and the managed services.
- Fence Agents Scripts that execute node fencing actions, given a target and fence device parameters.
- crmsh Advanced command-line interface for High-Availability cluster management in GNU/Linux.
- **cluster-glue** Reusable cluster components for Linux HA. This package contains node fencing plugins, an error reporting utility, and other reusable cluster components from the Linux HA project.
- **DRBD** Distributed Replicated Block Device, **DRBD** is a distributed replicated storage system for the Linuxplatform. It is implemented as a kernel driver, several userspace management applications, and some shell scripts. DRBD is traditionally used in high availability (HA) clusters.
- **DLM** A distributed lock manager (DLM) runs in every machine in a cluster, with an identical copy of a clusterwide lock database. In this way DLM provides software applications which are distributed across a cluster on multiple machines with a means to synchronize their accesses to shared resources.
- **gfs2-utils** Global File System 2 filesystem tools. The Global File System allows a cluster of machines to concurrently access shared storage hardware like SANs or iSCSI and network block devices.
- Keepalived Keepalived provides simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures. Loadbalancing framework relies on well-known and widely used Linux Virtual Server (IPVS) kernel module providing Layer4 loadbalancing. Keepalived implements a set of checkers to dynamically and adaptively maintain and manage loadbalanced server pool according their health. On the other hand high-availability is achieved by VRRP protocol.

Ubuntu HA Community Packages

Packages in this list are supported just like any other package available in **[universe] repository** would be.

Package	URL
pcs*	Ubuntu Upstream
csync2	Ubuntu Upstream
corosync-qdevice	Ubuntu Upstream
fence-virt	Ubuntu Upstream
sbd	Ubuntu Upstream

Package	URL
booth	Ubuntu Upstream

- **Corosync-Qdevice** Its primary use is for even-node clusters, operates at corosync (quorum) layer. Corosync-Qdevice is an independent arbiter for solving split-brain situations. (qdevice-net supports multiple algorithms).
- **SBD** A Fencing Block Device can be particularly useful in environments where traditional fencing mechanisms are not possible. SBD integrates with Pacemaker, a watchdog device and shared storage to arrange for nodes to reliably self-terminate when fencing is required.

Note: pcs will likely replace **crmsh** in [main] repository in future Ubuntu versions.

Ubuntu HA Deprecated Packages

Packages in this list are **only supported by the upstream community**. All bugs opened against these agents will be forwarded to upstream IF makes sense (affected version is close to upstream).

Package	URL	
ocfs2-tools	Ubuntu	Upstream

Ubuntu HA Related Packages

Packages in this list aren't necessarily **HA** related packages, but they have a very important role in High Availability Clusters and are supported like any other package provide by the [main] repository.

Package	URL
multipath-tools	Ubuntu Upstream
open-iscsi	Ubuntu Upstream
sg3-utils	Ubuntu Upstream
tgt OR targetcli-fb*	Ubuntu Upstream
lvm2	Ubuntu Upstream

• LVM2 in a Shared-Storage Cluster Scenario:

\mathbf{CLVM} - supported before $\mathbf{Ubuntu}\ \mathbf{20.04}$

A distributed lock manager (DLM) is used to broker concurrent LVM metadata accesses. Whenever a cluster node needs to modify the LVM metadata, it must secure permission from its local clvmd , which is in constant contact with other clvmd daemons in the cluster and can communicate a desire to get a lock on a particular set of objects.

lvmlockd - supported after Ubuntu 20.04

As of 2017, a stable LVM component that is designed to replace clvmd by making the locking of LVM objects transparent to the rest of LVM, without relying on a distributed lock manager.

The lvmlockd benefits over clvm are:

- lvmlockd supports two cluster locking plugins: DLM and SANLOCK. SANLOCK plugin can supports up to ${\sim}2000$ nodes that benefits LVM usage in big virtualization / storage cluster, while DLM plugin fits HA cluster.
- lvmlockd has better design than clvmd. clvmd is command-line level based locking system, which means the whole LVM software will get hang if any LVM command gets dead-locking issue.
 lvmlockd can work with lvmetad.
- Note: targetcli-fb (Linux LIO) will likely replace tgt in future Ubuntu versions.

Upstream Documentation

The server guide does not have the intent to document every existing option for all the HA related softwares described in this page, but to document recommended scenarios for Ubuntu HA Clusters. You will find more complete documentation upstream at:

• ClusterLabs

- Clusters From Scratch
- Managing Pacemaker Clusters
- Pacemaker Configuration Explained
- Pacemaker Remote Scaling HA Clusters
- Other
 - Ubuntu Bionic HA in Shared Disk Environments (Azure)

A very special thanks, and all the credits, to ClusterLabs Project for all that detailed documentation.

Ubuntu HA - Pacemaker Resource Agents

From the ClusterLabs definition:

Resource agents are the abstraction that allows Pacemaker to manage services it knows nothing about. They contain the logic for what to do when the cluster wishes to start, stop or check the health of a service. This particular set of agents conform to the Open Cluster Framework (OCF) specification.

At the moment, the resource-agents binary package has been split into two: resource-agents-base and resource-agents-extra. The resource-agents-base binary package contains a set of curated agents which the Ubuntu Server team continuously run tests to make sure everything is working as expected. All the other agents previously in the resource-agents binary package are now moved to the resource-agents-extra.

The resource-agents-base binary package contains the following agents in the latest Ubuntu release:

- IPaddr2
- iscsi
- iSCSILogicalUnit
- iSCSITarget
- LVM-activate
- systemd

All those agents are in main and are fully supported. The remaining agents are in resource-agents-extra and most of them are supported by upstream but they are not curated by the Ubuntu Server team. The set of resource agents that are not maintained by upstream is listed in /usr/share/doc/resource-agents-extra/DEPRECATED_AGENTS, the use of those agents is discouraged.

For the resource agents provided by resource-agents-base, we will be briefly describing how to use them.

IPaddr2

From its manpage:

This Linux-specific resource manages IP alias IP addresses. It can add an IP alias, or remove one. In addition, it can implement Cluster Alias IP functionality if invoked as a clone resource.

One could configure a IPaddr2 resource with the following command:

This is one way to set up IPaddr2, for more information please check its manpage.

iscsi

From its manpage:

Manages a local iSCSI initiator and its connections to iSCSI targets.

Once the iSCSI target is ready to accept connections from the initiator(s), with all the appropriate permissions, one could configure the iscsi resource with the following command:

```
$ crm configure primitive $RESOURCE_NAME iscsi \
    target=$TARGET \
    portal=$PORTAL
```

Where **\$TARGET** is the IQN (iSCSI Qualified Name) of the iSCSI target and **\$PORTAL** its address, which can be, for instance, formed by the IP address and port number used by the target daemon.

This is one way to set up iscsi, for more information please check its manpage.

iSCSILogicalUnit

From its manpage:

Manages iSCSI Logical Unit. An iSCSI Logical unit is a subdivision of an SCSI Target, exported via a daemon that speaks the iSCSI protocol.

This agent is usually used alongside with iSCSITarget to manage the target itself and its Logical Units. The supported implementation of iSCSI targets is using targetcli-fb, due to that, make sure to use lio-t as the implementation type. Considering one has an iSCSI target in place, the iSCSILogicalUnit resource could be configured with the following command:

```
$ crm configure primitive $RESOURCE_NAME iSCSILogicalUnit \
    implementation=lio-t \
    target_iqn=$IQN_TARGET \
    path=$DEVICE \
    lun=$LUN
```

Where implementation is set to lio-t as mentioned before, \$IQN_TARGET is the IQN (iSCSI Qualified Name) that this Logical Unit belongs to, \$DEVICE is the path to the exposed block device, and \$LUN is the number representing the Logical Unit which will be exposed to initiators.

This is one way to set up iSCSILogicalUnit, for more information please check its manpage.

iSCSITarget

From its manpage:

Manages iSCSI targets. An iSCSI target is a collection of SCSI Logical Units (LUs) exported via a daemon that speaks the iSCSI protocol.

This agent is usually used alongside with iSCSILogicalUnit to manage the target itself and its Logical Units. The supported implementation of iSCSI targets is using targetcli-fb, due to that, make sure to use lio-t as the implementation type. Considering one has targetcli-fb installed on the system, the iSCSITarget resource could be configured with the following command:

```
$ crm configure primitive $RESOURCE_NAME iSCSITarget \
    implementation=lio-t \
    iqn=$IQN_TARGET
```

Where implementation is set to lio-t as mentioned before and IQN_TARGET is the IQN (iSCSI Qualified Name) of the target.

This is one way to set up iSCSITarget, for more information please check its manpage.

LVM-activate

From its manpage:

This agent manages LVM activation/deactivation work for a given volume group.

Considering the LVM setup is ready to be activated and deactivated by this resource agent (make sure the system_id_resource is set to uname in /etc/lvm/lvm.conf), one could configure a LVM-activate resource with the following command:

```
$ crm configure primitive $RESOURCE_NAME ocf:heartbeat:LVM-activate \
    vgname=$VOLUME_GROUP \
    vg_access_mode=system_id
```

This is one way to set up LVM-activate, for more information please check its manpage.

Systemd

There is also a way to manage systemd unit files via a resource agent. One need to have the systemd unit file in place (already loaded by systemd) and configure a resource using the following command:

\$ crm configure primitive \$RESOURCE_NAME systemd:\$SERVICE_NAME

The $\$SERVICE_NAME$ can be any service managed by a systemd unit file, and it needs to be available for the cluster nodes.

References

- ClusterLabs website
- The OCF resource-agent developer's guide
- Users mailing list

Ubuntu HA - Pacemaker Fence Agents

From the ClusterLabs definition:

A fence agent (or fencing agent) is a **stonith**-class resource agent.

The fence agent standard provides commands (such as off and reboot) that the cluster can use to fence nodes. As with other resource agent classes, this allows a layer of abstraction so that Pacemaker doesn't need any knowledge about specific fencing technologies — that knowledge is isolated in the agent.

At the moment, the fence-agents binary package has been split into two: fence-agents-base and fence-agents-extra. The fence-agents-base binary package contains a set of curated agents which the Ubuntu Server team continuously run tests to make sure everything is working as expected. All the other agents previously in the fence-agents binary package are now moved to the fence-agents-extra.

The fence-agents-base binary package contains the following agents in the latest Ubuntu release:

- fence_ipmilan
 - fence_idrac
 - fence_ilo3
 - fence_ilo4
 - fence_ilo5
 - fence_imm
 - fence_ipmilanplus
- fence_mpath
- fence_sbd
- fence_scsi
- fence_virsh

All those agents are in main and are fully supported. The remaining agents in fence-agents-extra are supported by upstream but they are not curated by the Ubuntu Server team.

For the fence agents provided by fence-agents-base, we will be briefly describing how to use them.

fence_ipmilan

The content of this section is also applicable to the following fence agents: fence_idrac, fence_ilo3, fence_ilo4, fence_ilo5, fence_imm, and fence_ipmilanplus. All of them are symlinks to fence_ipmilan.

From its manpage:

fence_ipmilan is an I/O Fencing agentwhich can be used with machines controlled by IPMI. This agent calls support software ipmitool. WARNING! This fence agent might report success before the node is powered off. You should use -m/method onoff if your fence device works correctly with that option.

In a system which supports IPMI and with ipmitool installed, one could configure a fence_ipmilan resource with the following command:

 $\$ crm configure primitive $RESOURCE_NAME \ stonith:fence_ipmilan \$

```
ip=$IP \
ipport=$PORT \
username=$USER \
password=$PASSWD \
lanplus=1 \
action=$ACTION
```

Where \$IP is the IP address or hostname of fencing device, \$PORT is the TCP/UDP port to use for connection, \$USER is the login name and \$PASSWD its password, and \$ACTION is the fencing actions which by default is reboot.

This is one way to set up fence_ipmilan, for more information please check its manpage.

$fence_mpath$

From its manpage:

fence_mpath is an I/O fencing agent that uses SCSI-3 persistent reservations to control access multipath devices. Underlying devices must support SCSI-3 persistent reservations (SPC-3 or greater) as well as the "preempt-and-abort" subcommand. The fence_mpath agent works by having a unique key for each node that has to be set in /etc/multipath.conf. Once registered, a single node will become the reservation holder by creating a "write exclusive, registrants only" reservation on the device(s). The result is that only registered nodes may write to the device(s). When a node failure occurs, the fence_mpath agent will remove the key belonging to the failed node from the device(s). The failed node will no longer be able to write to the device(s). A manual reboot is required.

One could configure a fence_mpath resource with the following command:

```
$ crm configure primitive $RESOURCE_NAME stonith:fence_mpath \
    pcmk_host_map="$NODE1:$NODE1_RES_KEY;$NODE2:$NODE2_RES_KEY;$NODE3:$NODE3_RES_KEY" \
    pcmk_host_argument=plug \
    pcmk_monitor_action=metadata \
    pcmk_reboot_action=off \
    devices=$MPATH_DEVICE \
    meta provides=unfencing
```

The \$NODE1_RES_KEY is the reservation key used by this node 1 (same for the others node with access to the multipath device), please make sure you have reservation_key <key> in the default section inside /etc/multipath.conf and the multipathd service was reloaded after it.

This is one way to set up fence_mpath, for more information please check its manpage.

fence_sbd

From its manpage:

fence_sbd is I/O Fencing agent which can be used in environments where sbd can be used (shared storage).

With SBD (STONITH Block Device) configured on a system, one could configure the $fence_sbd$ resource with the following command:

\$ crm configure primitive \$RESOURCE_NAME stonith:external/sbd

This is one way to set up fence_sbd, for more information please check its manpage.

fence_scsi

From its manpage:

fence_scsi is an I/O fencing agent that uses SCSI-3 persistent reservations to control access to shared storage devices. These devices must support SCSI-3 persistent reservations (SPC-3 or greater) as well as the "preempt-and-abort" subcommand. The fence_scsi agent works by having each node in the cluster register a unique key with the SCSI device(s). Reservation key is generated from "node id" (default) or from "node name hash" (RECOMMENDED) by adjusting "key_value" option. Using hash is recommended to prevent issues when removing nodes from cluster without full cluster restart. Once registered, a single node will become the reservation holder by creating a "write exclusive, registrants only" reservation on the device(s). The result is that only registered nodes may write to the device(s). When a node failure occurs, the fence_scsi agent will remove the key belonging to the failed node from the device(s). The failed node will no longer be able to write to the device(s). A manual reboot is required.

One could configure a fence_scsi resource with the following command:

```
$ crm configure primitive $RESOURCE_NAME stonith:fence_scsi \
    pcmk_host_list="$NODE1 $NODE2 $NODE3" \
    devices=$SCSI_DEVICE \
    meta provides=unfencing
```

The pcmk_host_list parameter contains a list of cluster nodes that can access the managed SCSI device.

This is one way to set up fence_scsi, for more information please check its manpage.

$fence_virsh$

From its manpage:

fence_virsh is an I/O Fencing agent which can be used with the virtual machines managed by libvirt. It logs via ssh to a dom0 and there run virsh command, which does all work. By default, virsh needs root account to do properly work. So you must allow ssh login in your sshd_config.

One could configure a fence_virsh resource with the following command:

```
$ crm configure primitive $RESOURCE_NAME stonith:fence_virsh \
```

```
ip=$HOST_IP_ADDRESS \
login=$HOST_USER \
identity_file=$SSH_KEY \
plug=$NODE \
ssh=true \
use_sudo=true
```

This is one way to set up fence_virsh, for more information please check its manpage.

In order to avoid running the resource in the same node that should be fenced, we need to add a location restriction:

\$ crm configure location fence-\$NODE-location \$RESOURCE_NAME -inf: \$NODE

References

- ClusterLabs website
- Fence agents API documentation
- Users mailing list

Ubuntu HA - DRBD

Distributed Replicated Block Device (DRBD) mirrors block devices between multiple hosts. The replication is transparent to other applications on the host systems. Any block device hard disks, partitions, RAID devices, logical volumes, etc can be mirrored.

To get started using drbd, first install the necessary packages. From a terminal enter:

sudo apt install drbd-utils

Note

If you are using the *virtual kernel* as part of a virtual machine you will need to manually compile the drbd module. It may be easier to install the linux-server package inside the virtual machine.

This section covers setting up a drbd to replicate a separate /srv partition, with an ext3 filesystem between two hosts. The partition size is not particularly relevant, but both partitions need to be the same size.

Configuration

The two hosts in this example will be called *drbd01* and *drbd02*. They will need to have name resolution configured either through DNS or the /etc/hosts file. See ??? for details.

• To configure drbd, on the first host edit /etc/drbd.conf:

```
global { usage-count no; }
common { syncer { rate 100M; } }
resource r0 {
    protocol C;
    startup {
        wfc-timeout 15;
        degr-wfc-timeout 60;
    }
    net {
            cram-hmac-alg shal;
            shared-secret "secret";
    }
    on drbd01 {
            device /dev/drbd0;
            disk /dev/sdb1;
    }
}
```

```
address 192.168.0.1:7788;
meta-disk internal;
}
on drbd02 {
    device /dev/drbd0;
    disk /dev/sdb1;
    address 192.168.0.2:7788;
    meta-disk internal;
}
```

```
}
```

Note

There are many other options in /etc/drbd.conf, but for this example their default values are fine.

• Now copy /etc/drbd.conf to the second host:

scp /etc/drbd.conf drbd02:~

• And, on *drbd02* move the file to /etc:

sudo mv drbd.conf /etc/

• Now using the drbdadm utility initialize the meta data storage. On each server execute:

```
sudo drbdadm create-md r0
```

• Next, on both hosts, start the drbd daemon:

```
sudo systemctl start drbd.service
```

• On the *drbd01*, or whichever host you wish to be the primary, enter the following:

```
sudo drbdadm -- --overwrite-data-of-peer primary all
```

• After executing the above command, the data will start syncing with the secondary host. To watch the progress, on $drbd\theta 2$ enter the following:

watch -n1 cat /proc/drbd

To stop watching the output press Ctrl+c.

- Finally, add a file system to /dev/drbd0 and mount it:

sudo mkfs.ext3 /dev/drbd0
sudo mount /dev/drbd0 /srv

Testing

To test that the data is actually syncing between the hosts copy some files on the drbd01, the primary, to /srv:

sudo cp -r /etc/default /srv

```
Next, unmount /srv:
```

sudo umount /srv

Demote the primary server to the secondary role:

sudo drbdadm secondary r0

Now on the *secondary* server *promote* it to the *primary* role:

sudo drbdadm primary r0

Lastly, mount the partition:

sudo mount /dev/drbd0 /srv

Using *ls* you should see /srv/default copied from the former *primary* host *drbd01*.

References

- For more information on DRBD see the DRBD web site.
- The drbd.conf man page contains details on the options not covered in this guide.
- Also, see the drbdadm man page.

• The DRBD Ubuntu Wiki page also has more information.

Databases

Ubuntu provides two popular database servers. They are:

- MySQL
- PostgreSQL

Both are popular choices among developers, with similar feature sets and performance capabilities. Historically, Postgres tended to be a preferred choice for its attention to standards conformance, features, and extensibility, whereas Mysql may be more preferred for higher performance requirements, however over time each has made good strides catching up with the other. Specialized needs may make one a better option for a certain application, but in general both are good, strong options.

They are available in the main repository and equally supported by Ubuntu. This section explains how to install and configure these database servers.

MySQL

MySQL is a fast, multi-threaded, multi-user, and robust SQL database server. It is intended for mission-critical, heavy-load production systems and mass-deployed software.

Installation

To install MySQL, run the following command from a terminal prompt:

```
sudo apt install mysql-server
```

Once the installation is complete, the MySQL server should be started automatically. You can quickly check its current status via systemd:

sudo service mysql status

Oct 08 14:37:36 db.example.org systemd[1]: Starting MySQL Community Server...
Oct 08 14:37:38 db.example.org systemd[1]: Started MySQL Community Server.

The network status of the MySQL service can also be checked by running the ss command at the terminal prompt:

sudo ss -tap | grep mysql

When you run this command, you should see something similar to the following:

LISTEN	0	151	127.0.0.1:mysql	0.0.0:*	users:(("mysqld",pid=149190,fd=29))
LISTEN	0	70	*:33060	* *	users:(("mysqld",pid=149190,fd=32))

If the server is not running correctly, you can type the following command to start it:

sudo service mysql restart

A good starting point for troubleshooting problems is the systemd journal, which can be accessed at the terminal prompt with this command:

sudo journalctl -u mysql

Configuration

You can edit the files in /etc/mysql/ to configure the basic settings – log file, port number, etc. For example, to configure MySQL to listen for connections from network hosts, in the file /etc/mysql/mysql.conf.d/mysqld.cnf, change the *bind-address* directive to the server's IP address:

bind-address = 192.168.0.5

Note

Replace 192.168.0.5 with the appropriate address, which can be determined via ip address show.

After making a configuration change, the MySQL daemon will need to be restarted:

sudo systemctl restart mysql.service

Database Engines

Whilst the default configuration of MySQL provided by the Ubuntu packages is perfectly functional and performs well there are things you may wish to consider before you proceed.

MySQL is designed to allow data to be stored in different ways. These methods are referred to as either database or storage engines. There are two main engines that you'll be interested in: InnoDB and MyISAM. Storage engines are transparent to the end user. MySQL will handle things differently under the surface, but regardless of which storage engine is in use, you will interact with the database in the same way.

Each engine has its own advantages and disadvantages.

While it is possible, and may be advantageous to mix and match database engines on a table level, doing so reduces the effectiveness of the performance tuning you can do as you'll be splitting the resources between two engines instead of dedicating them to one.

- MyISAM is the older of the two. It can be faster than InnoDB under certain circumstances and favours a read only workload. Some web applications have been tuned around MyISAM (though that's not to imply that they will slow under InnoDB). MyISAM also supports the FULLTEXT data type, which allows very fast searches of large quantities of text data. However MyISAM is only capable of locking an entire table for writing. This means only one process can update a table at a time. As any application that uses the table scales this may prove to be a hindrance. It also lacks journaling, which makes it harder for data to be recovered after a crash. The following link provides some points for consideration about using MyISAM on a production database.
- InnoDB is a more modern database engine, designed to be ACID compliant which guarantees database transactions are processed reliably. Write locking can occur on a row level basis within a table. That means multiple updates can occur on a single table simultaneously. Data caching is also handled in memory within the database engine, allowing caching on a more efficient row level basis rather than file block. To meet ACID compliance all transactions are journaled independently of the main tables. This allows for much more reliable data recovery as data consistency can be checked.

As of MySQL 5.5 InnoDB is the default engine, and is highly recommended over MyISAM unless you have specific need for features unique to the engine.

Advanced configuration

Creating a tuned configuration

There are a number of parameters that can be adjusted within MySQL's configuration files that will allow you to improve the performance of the server over time.

Many of the parameters can be adjusted with the existing database, however some may affect the data layout and thus need more care to apply.

First, if you have existing data, you will need to carry out a mysqldump and reload:

mysqldump --all-databases --routines -u root -p > ~/fulldump.sql

This will then prompt you for the root password before creating a copy of the data. It is advisable to make sure there are no other users or processes using the database whilst this takes place. Depending on how much data you've got in your database, this may take a while. You won't see anything on the screen during this process.

Once the dump has been completed, shut down MySQL:

sudo service mysql stop

It's also a good idea to backup the original configuration:

sudo rsync -avz /etc/mysql /root/mysql-backup

Next, make any desired configuration changes.

Then delete and re-initialise the database space and make sure ownership is correct before restarting MySQL:

sudo rm -rf /var/lib/mysql/*
sudo mysqld --initialize
sudo chown -R mysql: /var/lib/mysql
sudo service mysql start

The final step is re-importation of your data by piping your SQL commands to the database.

cat ~/fulldump.sql | mysql

For large data imports, the 'Pipe Viewer' utility can be useful to track import progress. Ignore any ETA times produced by pv, they're based on the average time taken to handle each row of the file, but the speed of inserting can vary wildly from row to row with mysqldumps:

```
sudo apt install pv
pv ~/fulldump.sql | mysql
```

Once that is complete all is good to go!

Note

This is not necessary for all my.cnf changes. Most of the variables you may wish to change to improve performance are adjustable even whilst the server is running. As with anything, make sure to have a good backup copy of config files and data before making changes.

MySQL Tuner

MySQL Tuner connects to a running MySQL instance and offer configuration suggestions to optimize the database for your workload. The longer the server has been running, the better the advice mysqltuner can provide. In a production environment, consider waiting for at least 24 hours before running the tool. You can install mysqltuner from the Ubuntu repositories:

sudo apt install mysqltuner

Then once its been installed, run:

mysqltuner

and wait for its final report. The top section provides general information about the database server, and the bottom section provides tuning suggestions to alter in your my.cnf. Most of these can be altered live on the server without restarting; look through the official MySQL documentation (link in Resources section) for the relevant variables to change in production. The following example is part of a report from a production database showing potential benefits from increasing the query cache:

```
----- Recommendations ------
General recommendations:
    Run OPTIMIZE TABLE to defragment tables for better performance
    Increase table_cache gradually to avoid file descriptor limits
Variables to adjust:
    key_buffer_size (> 1.4G)
    query_cache_size (> 32M)
    table_cache (> 64)
    innodb_buffer_pool_size (>= 22G)
```

It goes without saying that performance optimization strategies vary from application to application. So for example, what works best for Wordpress might not be the best for Drupal or Joomla. Performance can be dependent on the types of queries, use of indexes, how efficient the database design is and so on. You may find it useful to spend some time searching for database tuning tips based on what applications you're using. Once you've reached the point of diminishing returns from database configuration adjustments, look to the application itself for improvements, or invest in more powerful hardware and/or scaling up the database environment.

Resources

- See the MySQL Home Page for more information.
- Full documentation is available in both online and offline formats from the MySQL Developers portal
- For general SQL information see the O'Reilly books Getting Started with SQL: A Hands-On Approach for Beginners by Thomas Nield as an entry point and SQL in a Nutshell as a quick reference.
- The Apache MySQL PHP Ubuntu Wiki page also has useful information.

PostgreSQL

PostgreSQL is an object-relational database system that has the features of traditional commercial database systems with enhancements to be found in next-generation DBMS systems.

Installation

To install PostgreSQL, run the following command in the command prompt:

sudo apt install postgresql

The database service is automatically configured with viable defaults, but can be customized based on your specialized needs.

Configuration

PostgreSQL supports multiple client authentication methods. In Ubuntu, peer is the default authentication method used for local connections, while scram-sha-256 is the default for host connections (this used to be md5 until Ubuntu 21.10). Please refer to the PostgreSQL Administrator's Guide if you would like to configure alternatives like Kerberos.

The following discussion assumes that you wish to enable TCP/IP connections and use the MD5 method for client authentication. PostgreSQL configuration files are stored in the /etc/postgresql/<version>/main directory. For example, if you install PostgreSQL 12, the configuration files are stored in the /etc/postgresql/l2/main directory.

Tip

To configure *IDENT* authentication, add entries to the /etc/postgresql/l2/main/pg_ident.conf file. There are detailed comments in the file to guide you.

To enable other computers to connect to your PostgreSQL server, edit the file /etc/postgresql/12/main/postgresql.conf

Locate the line $\#listen_addresses = `localhost'$ and change it to:

listen_addresses = '*'

Note

To allow both IPv4 and IPv6 connections replace 'localhost' with '::'

For details on other parameters, refer to the configuration file or to the PostgreSQL documentation for information on how they can be edited.

Now that we can connect to our PostgreSQL server, the next step is to set a password for the *postgres* user. Run the following command at a terminal prompt to connect to the default PostgreSQL template database:

sudo -u postgres psql template1

The above command connects to PostgreSQL database *template1* as user *postgres*. Once you connect to the PostgreSQL server, you will be at a SQL prompt. You can run the following SQL command at the psql prompt to configure the password for the user *postgres*.

ALTER USER postgres with encrypted password 'your_password';

After configuring the password, edit the file $/etc/postgresql/l2/main/pg_hba.conf$ to use MD5 authentication with the *postgres* user:

local all postgres md5

Finally, you should restart the PostgreSQL service to initialize the new configuration. From a terminal prompt enter the following to restart PostgreSQL:

sudo systemctl restart postgresql.service

Warning

The above configuration is not complete by any means. Please refer to the PostgreSQL Administrator's Guide to configure more parameters.

You can test server connections from other machines by using the PostgreSQL client.

```
sudo apt install postgresql-client
psql -h postgres.example.com -U postgres -W
```

Note

Replace the domain name with your actual server domain name.

Streaming Replication

PostgreSQL has a nice feature called Streaming Replication which provides the capability to continuously ship and apply the WAL XLOG records to some number of standby servers in order to keep them current. Here is presented a very basic and simple way to replicate a PostgreSQL server (master) in a standby server.

First, create a replication user in the master to be used by the standby server:

\$ sudo -u postgres createuser --replication -P -e replicator

Let's configure the master server to turn on the streaming replication. Open the file /etc/postgresql/l2/main/postgresql.conf and make sure you have the following lines:

```
listen_addresses = '*'
wal_level = replica
max_wal_senders = 10
```

Also edit the file /etc/postgresql/12/main/pg_hba.conf to add an extra line to allow the standby server connection using the replicator user:

host replication replicator <IP address of the standby> md5

Restart the service to apply changes:

\$ sudo systemctl restart postgresql

Now, in the standby server, let's stop the PostgreSQL service:

```
$ sudo systemctl stop postgresql
```

Edit the /etc/postgresql/12/main/postgresql.conf to set up hot standby:

```
hot_standby = on
```

Back up the current state of the master server:

```
$ sudo su - postgres
$ cp -R /var/lib/postgresql/12/main /var/lib/postgresql/12/main_bak
$ rm -rf /var/lib/postgresql/12/main/* # remove all the files in the data directory
$ pg_basebackup -h <IP adrress of the master> -D /var/lib/postgresql/12/main -U replicator -P -v -R
```

In the pg_basebackup command the flags represent the following:

- -h: the hostname or IP address of the master server
- -D: the data directory
- $\mathsf{-U}:$ the user to be used in the operation
- -P: tuns on progess reporting
- -v: enables verbose mode
- -R: Creates a standby.signal file and append connection settings to postgresql.auto.conf

Finally, let's start the PostgreSQL service on standby server:

\$ sudo systemctl start postgresql

To make sure it is working, go to the master server and run the following command:

\$ sudo -u postgres psql -c "select * from pg_stat_replication;"

You need to see an entry for the standby server. As you can see in the presented row the data synchronization is assynchronous. If you want to make it synchronous, go to the /etc/postgresql/l2/main/postgresql.conf file in the master server and make sure you have the following lines:

synchronous_commit = on
synchronous_standby_names = '*' # you can also specify a IP address

And restart the service:

\$ sudo systemctl restart postgresql

Now, you can check the pg_stat_replication table again and the sync_state of the standby server need to become sync instead of async:

\$ sudo -u postgres psql -c "select * from pg_stat_replication;"

To test the replication you can now create a test database in the master and check if it is replicated in the standby server:

\$ sudo -u postgres createdb test # on the master server \$ sudo -u postgres psgl -c "\l" # on the standby server

You need to be able to see the test database created on the master in the standby server.

Backups

PostgreSQL databases should be backed up regularly. Refer to the PostgreSQL Administrator's Guide for different approaches.

Resources

• As mentioned above the PostgreSQL Administrator's Guide is an excellent resource. The guide is also available in the postgresql-doc-12 package. Execute the following in a terminal to install the package:

sudo apt install postgresql-doc-12

To view the guide enter file:///usr/share/doc/postgresql-doc-12/html/index.html into the address bar of your browser.

- For general SQL information see the O'Reilly books Getting Started with SQL: A Hands-On Approach for Beginners by Thomas Nield as an entry point and SQL in a Nutshell as a quick reference.
- Also, see the PostgreSQL Ubuntu Wiki page for more information.

Logging, Monitoring, and Alerting - Introduction

Logging, Monitoring, and Alerting (LMA) is a collection of tools used to guarantee the availability of your running infrastructure. Your LMA stack will help point out issues in load, networking, and other resources before it becomes a failure point.

Architectural Overview

Canonical's LMA stack involves several discrete software services acting in concert, including:

- Prometheus
- Prometheus Alertmanager
- Grafana
- Telegraf

Telegraf collects metrics from the operating system, running software, and other inputs. Its plugin system permits export of data in any arbitrary format; for this system we collect the data in a central data manager called Prometheus.

Prometheus works as a hub, polling data from different Telegraf nodes and sending it to various outputs, including persistent storage. For this LMA stack, visualization is handled via Grafana and email/pager alerts are generated via the Prometheus Alertmanager plugin.

Getting Started

Let's set up a basic demonstration with two nodes, the first acting as a placeholder load with Telegraf installed - the "Workload", and the second acting as our data visualization system - the "Monitor". This will help familiarize ourselves with the various components and how they interoperate.

The Workload node will be running Telegraf to collect metrics from whatever load we're monitoring. For demo purposes we'll just read the cpu/mem data from the node. In a real environment, we'd have multiple hosts each with their own Telegraf instance collecting hardware, network, and software status particular to that node.

Our Monitor node will double as both data store and web UI, receiving data from the Workload, storing it to disk, and displaying it for analysis.

For clarity, we'll refer to these two hosts as named 'workload' and 'monitor'. If you select other hostnames, simply substitute the correct ones as we go.

As reference, here are the ports we'll be binding for each service:

Prometheus	monitor:9090
Alertmanager	monitor:9093
Grafana	monitor:3000
Telegraf	workload:9273

Workload Node

First, let's set up the Workload. We'll be using LXD as our container technology, but there's very little that depends on this particularly; any VM, container, or bare metal host should work for purposes of this example, so long as it's running Ubuntu 20.10. With LXD installed in our host we can use its lxc command line tool to create our containers:

\$ lxc launch ubuntu:20.10 workload Creating workload Starting workload

\$ lxc exec workload -- bash
workload:~#

On the Workload, install Telegraf:

```
workload:~# apt update
workload:~# apt install telegraf
```

Telegraf processes input data to transform, filter, and decorate it, and then performs selected aggregation functions on it such as tallies, averages, etc. The results are published for collection by external services; in our case Prometheus will be collecting the cpu:mem data to the Monitor node.

Open /etc/telegraf/telegraf.conf and scroll down to the "INPUT PLUGINS" section. What we'll need is the following configuration settings, which you should find already enabled by default:

```
[[inputs.cpu]]
percpu = true
totalcpu = true
collect_cpu_time = false
report_active = false
```

Looking at the config file you'll notice it's almost entirely commented out. There are three different types of sections in the file: [[inputs]], such as we set above; [[outputs]], that we'll set up next; and the [[agent]] setting with several performance tuning parameters such as the collection interval, which we're setting to 10 seconds. The agent defaults are fine for our demo and for basic usage.

Finally, we need to define where Telegraf will provide output. Open /etc/telegraf/telegraf.conf and scroll down to the "OUTPUT PLUGINS" section and add the following output configuration:

```
[[outputs.prometheus_client]]
listen = "workload:9273"
metric_version = 2
```

#[[outputs.influxdb]]

We won't be using Influxdb, so you can comment that section out if it's enabled.

Now restart the Telegraf service:

```
....I! [agent] Config: Interval:10s, Quiet:false, Hostname:"workload", Flush Interval:10s
```

```
...I! [outputs.prometheus_client] Listening on http://127.0.0.1:9273/metrics
```

Verify that it is collecting metrics by connecting to Telegraf's web interface:

```
workload:~# wget -0- http://workload:9273/metrics
```

```
# HELP cpu_usage_guest Telegraf collected metric
# TYPE cpu_usage_guest gauge
cpu_usage_guest{cpu="cpu-total",host="workload"} 0
cpu_usage_guest{cpu="cpu0",host="workload"} 0
cpu_usage_guest{cpu="cpu1",host="workload"} 0
cpu_usage_guest{cpu="cpu10",host="workload"} 0
...
cpu_usage_idle{cpu="cpu10",host="workload"} 92.74914376428686
cpu_usage_idle{cpu="cpu0",host="workload"} 86.72897196325539
cpu_usage_idle{cpu="cpu1",host="workload"} 90.11857707405758
cpu_usage_idle{cpu="cpu10",host="workload"} 95.95141700494543
```

Monitor Node

Now let's create the Monitor. As before, we'll be using LXD as the container technology but if you use some other approach it should only require some adaptation:

```
$ lxc launch ubuntu:20.10 monitor
Creating monitor
Starting monitor
$ lxc exec monitor -- bash
monitor:~#
```

Make a note of the newly created container's IP address, which we'll need later on;

```
monitor:~# ip addr | grep 'inet .* global'
inet 10.69.244.104/24 brd 10.69.244.255 scope global dynamic eth0
```

Verify the Workload's Telegraf instance can be reached from the Monitor:

```
monitor:~# wget -0- http://workload:9273/metrics
```

We'll be setting up a few components to run on this node using their Snap packages. LXD images should normally have snap pre-installed but if not install it manually:

monitor:~# apt install snapd

A. Prometheus

Prometheus will be our data manager. It collects data from external sources - Telegraf in our case - and distributes it to various destinations such as email/pager alerts, web UI's, API clients, remote storage services, etc. We'll get into those shortly.

Let's install Prometheus itself and the Prometheus Alertmanager plugin for alerts, along with a bunch of required dependencies:

monitor:~# snap install prometheus
monitor:~# snap install prometheus-alertmanager

The snap will automatically configure and start the service. To verify this, run:

```
monitor:~# snap services
Service Startup Current Notes
lxd.activate enabled inactive -
lxd.daemon enabled inactive socket-activated
prometheus.prometheus enabled active -
prometheus-alertmanager enabled active -
```

Verify that Prometheus is listening on the port as we expect:

visualizer:~# ss -tulpn | grep prometheus
tcp LISTEN 0 128 *:9090 *:* users:(("prometheus",pid=618,fd=8))

journalctl can be also used to review the state of snap services, particularly if more detail is desired. For example, to see where Prometheus is loading its config from:

monitor:~# journalctl | grep "prometheus.*config"

...msg="Completed loading of configuration file" filename=/var/snap/prometheus/32/prometheus.yml

Although the filename points to a specific snap revision (32, in this case), we can use the generic file /var/snap/prometheus/current/prometheus.yml here in order to make things more generic.

Edit this config file to register the targets we'll be reading data from. This will go under the scrape_configs section of the file:

```
scrape_configs:
    # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
    - job_name: 'prometheus'
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.
    static_configs:
    - targets: ['localhost:9090']
    - job_name: 'telegraf'
    static_configs:
    - targets: ['workload:9273']
Then restart Prometheus:
```

monitor:~# snap restart prometheus

While we'll be using Grafana for visualization, Prometheus also has a web interface for viewing and interacting with the collected data. At this stage we can load it just to verify our setup is working properly. In a web browser, navigate to the Monitor's IP address, and port 9090. You should see Prometheus' interface, as in the following image:

normetheus Time Series Collection and Processing Server - Mozilla Firefox				^ .	- 0	×
● Prometheus Time Seri × +						
(e) → C (a) (c) 2/2 10.69.244.104:9090/graph ···· ☺ (c)	Ψ	n D	٠	۲	8	≡
Prometheus Alerts Graph Status - Help						
C Enable query history		Try exp	erime	ntal R	eact l	я
Expression (press Shift+Enter for newlines)						
Execute - insert metric at cursor - •						
Graph Console						
4 Moment						
Element Value						
no data						
			Re	move	Gra	ph
Add Graph						
		_				_

prometheus_0960 $\times 632$ 37.7 KB

In the entry box, enter cpu_usage_system, select the Graph tab and click Execute. This should show a graph of our collected cpu data so far. Prometheus also has a secondary web UI using React.js.



prometheus_1960 \times 888 137 KB



prometheus_2960 $\times 888$ 227 KB

B. Alerts

Let's tackle the Alert Manager next.

Edit /var/snap/prometheus/current/prometheus.yml again, adding the following to the alerting and rules_files sections:

/var/snap/prometheus/current/prometheus.yml

```
#...
# Alertmanager configuration
alerting:
    alertmanagers:
        static_configs:
            targets:
               127.0.0.1:9093
rule_files:
               'alerts.yml'
```

Now create /var/snap/prometheus/current/alerts.yml with the following contents:

/var/snap/prometheus/current/alerts.yml

```
groups:
- name: demo-alerts
rules:
- alert: HighLoad
expr: node_load1 > 2.0
for: 60m
labels:
```

```
severity: normal
annotations:
    description: '{{ $labels.instance }} of job {{ $labels.job }} is under high load.'
    summary: Instance {{ $labels.instance }} under high load.
    value: '{{ $value }}'
- alert: InstanceDown
    expr: up == 0
    for: 5m
    labels:
        severity: major
annotations:
        summary: "Instance {{ $labels.instance }} down"
    description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes."
```

This adds two alerts: one for high processor load, and one to report if the node has been unreachable for over 5 minutes. We're considering high cpu to be a load of 2 or higher for an hour; obviously this would need set to something more sensible for the style of workloads your production system experiences.

With the alerts themselves now defined, we next need to instruct Prometheus's alert manager how to handle them. There is a sample configuration installed to /var/snap/prometheus-alertmanager/current/alertmanager.yml, however it's full of example data. Instead, replace it entirely with this content:

/var/snap/prometheus-alertmanager/current/alertmanager.yml
global:

```
resolve_timeout: 5m
route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
inhibit_rules:
        source_match:
        severity: 'critical'
        target_match:
        severity: 'warning'
        equal: ['alertname', 'dev', 'instance']
```

Restart the alertmanager after making the configuration change:

workload:~# snap restart prometheus-alertmanager

C. Grafana

Grafana provides our main dashboard, from which we can generate graphs and other visuals to study the collected metrics. Grafana can read its data directly from log files, but we'll focus on using Prometheus as its principle data source. Grafana is available as a snap. You can install it like this:

```
monitor:~# snap install grafana
grafana 6.7.4 from Alvaro Uría (aluria) installed
```

It uses port 3000:

# ss	-tulpn	grep	grafana	
tcp	LISTEN	0	128	

users:(("grafana-server",pid=1449,fd=10))

We next need to know where it expects its configuration:

```
monitor:~# journalctl | grep "grafana.*conf"
```

```
... msg="Config loaded from" logger=settings file=/snap/grafana/36/conf/defaults.ini
```

*:3000

```
... msg="Config overridden from Environment variable" logger=settings var="GF_PATHS_PROVISIONING=/var/snap/grafana/comment_variable" logger=settings var="GF_PATHS_PROVISIONING=/var/snap/grafanave" logger=settings var="GF_PATHS_PA
```

* *

... error="open /var/snap/grafana/common/conf/provisioning/datasources: no such file or directory"

So we see it's getting its defaults from /snap/grafana/36/conf/, but /snap/ is a read-only directoy and therefore we cannot edit the file. Instead, we should put our customizations inside /var/snap/grafana/36/conf/grafana.ini. You can also use the generic path /var/snap/grafana/current/conf/grafana.ini.

For a production installation, the defaults.ini has numerous parameters we'd want to customize for our site, however for the demo we'll accept all the defaults. We do need to configure our data sources, but can do this via the web interface:

\$ firefox http://10.69.244.104:3000

Login with 'admin' and 'admin' as the username and password. This should bring you to the main Grafana page, where you can find links to tutorials and documentation. Go ahead and delete any example data sources and/or dashboards.



grafana_0960 \times 888 98.5 KB

Select the button to add a new data source and select "Prometheus". On the "Data Sources / Prometheus" edit page, set the name to Prometheus, URL to http://localhost:9090, and Access to "Server (default)" to make Grafana pull data from the Prometheus service we set up. The remaining settings can be left to defaults. Click "Save & Test".

*		and data summer and data summer devices a Marilla th	and the second		
	And a subserver of	Inter sona tenerori pare sona tenerori o comuna i resona re			
			_		
	0.6	0 6 19.49 244 104 1000/talasturres/rew/pettripitaried		* * * *	
6	9	Add data source			
					Canad
0					
	Tere series	databasers			
•					
۰		Ponetos			
08					
Ť	57	Enghine Core exercitive activabilities			
	24	Open TSDB Conte manufacture anter-distance			
	\otimes	Made Igen sources was desired			
	Logging &	locarrent databases			
	2	Cable Constitues for for tops 200 hogging without from the trade Labo			
8 0	3	Darsenarch Com marchagea brancha Addan			
grafana_1960 \times 888 79.3 KB



grafana_2960 $\times 888$ 64.9 KB

Returning to the Grafana home page, next set up a "New Dashboard". A dashboard can hold one or more panels, each of which can be connected to one or more data queries. Let's add a panel for CPU data. For the query, enter "cpu_usage_system" in the Metrics field.

٠	trace trains	- Madia Forka	
8	CONTRACTOR OF THE	m 0.0	
0 =			
	-	comerco Craferea	
•	0	± ±	
* Banka		LAURANCE U	Victoria Processor Normania Processor Processor Processor
5			

grafana_3960 $\times 888$ 92.8 KB



grafana_4960 \times 888 44.6 KB



grafana_5960 $\times 888$ 107 KB

On the left you can see four buttons to configure four elements of the panel: It's data source, its visualization, general settings, and alerts. The general settings page allows us to set a title for the panel, for instance. Make any other customizations you desire, and then save the dashboard using the save icon at the top of the page.



grafana_6960 $\times 888$ 112 KB



grafana_7960×888 90.1 KB

Using the same procedure, add additional panels for processor load and memory usage. Panels can be used to present other types of data as well, such as numerical indicators, logs, newsfeeds, or markdown-formatted documentation. For example, you can add a panel to display the system uptime, such as in the following image:



grafana_9960 $\times 888$ 58.1 KB

Try also adding a panel with the "Text" visualization option for entering descriptive text about our demo. Save, and then view the final dashboard:



grafana_X960 \times 888 141 KB

Monitoring

Overview

The monitoring of essential servers and services is an important part of system administration. Most network services are monitored for performance, availability, or both. This section will cover installation and configuration of Nagios for availability monitoring, and Munin for performance monitoring.

The examples in this section will use two servers with hostnames server01 and server02. Server01 will be configured with Nagios to monitor services on itself and server02. Server01 will also be setup with the munin package to gather information from the network. Using the munin-node package, server02 will be configured to send information to server01.

Hopefully these simple examples will allow you to monitor additional servers and services on your network.

Nagios

Installation

First, on server01 install the nagios package. In a terminal enter:

sudo apt install nagios3 nagios-nrpe-plugin

You will be asked to enter a password for the *nagiosadmin* user. The user's credentials are stored in /etc/nagios3/htpasswd.users. To change the *nagiosadmin* password, or add additional users to the Nagios CGI scripts, use the htpasswd that is part of the apache2-utils package.

For example, to change the password for the nagios admin user enter:

sudo htpasswd /etc/nagios3/htpasswd.users nagiosadmin

To add a user:

sudo htpasswd /etc/nagios3/htpasswd.users steve

Next, on server02 install the nagios-nrpe-server package. From a terminal on server02 enter:

sudo apt install nagios-nrpe-server

Note

NRPE allows you to execute local checks on remote hosts. There are other ways of accomplishing this through other Nagios plugins as well as other checks.

Configuration Overview

There are a couple of directories containing Nagios configuration and check files.

- /etc/nagios3: contains configuration files for the operation of the nagios daemon, CGI files, hosts, etc.
- /etc/nagios-plugins: houses configuration files for the service checks.
- /etc/nagios: on the remote host contains the nagios-nrpe-server configuration files.
- /usr/lib/nagios/plugins/: where the check binaries are stored. To see the options of a check use the -h option.

For example: /usr/lib/nagios/plugins/check_dhcp -h

There are a plethora of checks Nagios can be configured to execute for any given host. For this example Nagios will be configured to check disk space, DNS, and a MySQL hostgroup. The DNS check will be on *server02*, and the MySQL hostgroup will include both *server01* and *server02*.

Note

See details on setting up MySQL.

Additionally, there are some terms that once explained will hopefully make understanding Nagios configuration easier:

- Host: a server, workstation, network device, etc that is being monitored.
- Host Group: a group of similar hosts. For example, you could group all web servers, file server, etc.
- Service: the service being monitored on the host. Such as HTTP, DNS, NFS, etc.
- *Service Group*: allows you to group multiple services together. This is useful for grouping multiple HTTP for example.
- *Contact*: person to be notified when an event takes place. Nagios can be configured to send emails, SMS messages, etc.

By default Nagios is configured to check HTTP, disk space, SSH, current users, processes, and load on the *localhost*. Nagios will also ping check the *gateway*.

Large Nagios installations can be quite complex to configure. It is usually best to start small, one or two hosts, get things configured the way you like then expand.

Configuration

• First, create a *host* configuration file for *server02*. Unless otherwise specified, run all these commands on *server01*. In a terminal enter:

```
sudo cp /etc/nagios3/conf.d/localhost_nagios2.cfg \
/etc/nagios3/conf.d/server02.cfg
```

Note

check DNS service.

In the above and following command examples, replace "server01", "server02" 172.18.100.100, and 172.18.100.101 with the host names and IP addresses of your servers.

 $Next, \, edit \, \mbox{/etc/nagios3/conf.d/server02.cfg:}$

```
define host{
    use generic-host ; Name of host template to use
    host_name server02
    alias Server 02
    address 172.18.100.101
}
```

```
256
```

```
define service {
    use generic-service
    host_name server02
    service_description DNS
    check_command check_dns!172.18.100.101
```

}

Restart the nagios daemon to enable the new configuration:

```
sudo systemctl restart nagio3.service
```

• Now add a service definition for the MySQL check by adding the following to /etc/nagios3/conf.d/services_nagios2.cfg:

```
# check MySQL servers.
define service {
    hostgroup_name mysql-servers
    service_description MySQL
    check_command check_mysql_cmdlinecred!nagios!secret!$HOSTADDRESS
    use generic-service
    notification_interval 0 ; set > 0 if you want to be renotified
}
```

ß

A mysql-servers hostgroup now needs to be defined. Edit /etc/nagios3/conf.d/hostgroups_nagios2.cfg adding:

```
# MySQL hostgroup.
define hostgroup {
    hostgroup_name mysql-servers
        alias MySQL servers
        members localhost, server02
    }
```

The Nagios check needs to authenticate to MySQL. To add a nagios user to MySQL enter:

mysql -u root -p -e "create user nagios identified by 'secret';"

Note

The nagios user will need to be added all hosts in the mysql-servers hostgroup.

Restart nagios to start checking the MySQL servers.

sudo systemctl restart nagios3.service

• Lastly configure NRPE to check the disk space on server02.

On *server01* add the service check to /etc/nagios3/conf.d/server02.cfg:

```
# NRPE disk check.
define service {
    use generic-service
    host_name server02
    service_description nrpe-disk
    check_command check_nrpe_larg!check_all_disks!172.18.100.101
}
```

}

Now on *server02* edit /etc/nagios/nrpe.cfg changing:

allowed_hosts=172.18.100.100

And below in the command definition area add:

```
command[check_all_disks]=/usr/lib/nagios/plugins/check_disk -w 20% -c 10% -e
```

Finally, restart nagios-nrpe-server:

sudo systemctl restart nagios-nrpe-server.service

Also, on server01 restart nagios:

sudo systemctl restart nagios3.service

You should now be able to see the host and service checks in the Nagios CGI files. To access them point a browser to http://server01/nagios3. You will then be prompted for the *nagiosadmin* username and password.

References

This section has just scratched the surface of Nagios' features. The nagios-plugins-extra and nagios-snmp-plugins contain many more service checks.

- For more information see Nagios website.
- Specifically the Nagios Online Documentation site.
- There is also a list of books related to Nagios and network monitoring:
- The Nagios Ubuntu Wiki page also has more details.

Munin

Installation

Before installing Munin on *server01* apache2 will need to be installed. The default configuration is fine for running a munin server. For more information see setting up Apache.

First, on server01 install munin. In a terminal enter:

sudo apt install munin

Now on server02 install the munin-node package:

sudo apt install munin-node

Configuration

On *server01* edit the /etc/munin/munin.conf adding the IP address for *server02*:

First our "normal" host.

[server02]

address 172.18.100.101

Note

Replace server02 and 172.18.100.101 with the actual hostname and IP address for your server.

Next, configure munin-node on server02. Edit /etc/munin/munin-node.conf to allow access by server01:

allow ^172\.18\.100\.100\$

Note

Replace 172 . 18 . 100 . 100 with IP address for your munin server.

Now restart munin-node on server02 for the changes to take effect:

sudo systemctl restart munin-node.service

Finally, in a browser go to http://server01/munin, and you should see links to nice graphs displaying information from the standard *munin-plugins* for disk, network, processes, and system.

Note

Since this is a new install it may take some time for the graphs to display anything useful.

Additional Plugins

The munin-plugins-extra package contains performance checks additional services such as DNS, DHCP, Samba, etc. To install the package, from a terminal enter:

sudo apt install munin-plugins-extra

Be sure to install the package on both the server and node machines.

References

- See the Munin website for more details.
- Specifically the Munin Documentation page includes information on additional plugins, writing plugins, etc.

Tools - Logwatch

Logs are an invaluable source of information about problems that may arise in your server. Logwatch keeps an eye on your logs for you, flags items that may be of interest, and report them via email.

Install Logwatch using the normal method:

\$ sudo apt install logwatch

You will also need to manually create its temporary directory in order for it to work:

\$ sudo mkdir /var/cache/logwatch

Logwatch's stock configuration is kept in /usr/share/logwatch/default.conf/logwatch.conf, however rather than edit our configuration changes directly to that file, the convention is to copy it into /etc for modification:

\$ sudo cp /usr/share/logwatch/default.conf/logwatch.conf /etc/logwatch/conf/

With your favorite editor, open /etc/logwatch/conf/logwatch.conf. The uncommented lines indicate the default configuration values. First, lets customize some of the basics:

```
Output = mail
MailTo = me@mydomain.org
MailFrom = logwatch@host1.mydomain.org
Detail = Low
Service = All
```

This assumes you've set up mail services on host1 that enables it to deliver email to your me@mydomain.org address. These emails will be addressed as from logwatch@host1.mydomain.org.

The Detail level defines how much information is included in the reports. Other values are Medium and High.

Logwatch will then monitor logs for all services on the system, unless specified otherwise with the Service parameter. If there are undesired services being included in the reports, they can be disabled by removing them with additional Service fields. E.g.:

```
Service = "-http"
Service = "-eximstats"
```

Next, run Logwatch manually to verify your configuration changes are valid.

```
$ sudo logwatch --detail Low --range today
```

----- pam_unix Begin ------

sudo: Sessions Opened: bryce -> root: 1 Time(s)

----- pam_unix End ------

----- rsnapshot Begin ------

ERRORS:

/usr/bin/rsnapshot hourly: completed, but with some errors: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host2:/etc/: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host2:/home/: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host3:/proc/uptime: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host3:/etc/: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host3:/home/: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host3:/home/: 5 Time(s) /usr/bin/rsync returned 127 while processing root@host3:/home/: 5 Time(s)

----- rsnapshot End -----

----- SSHD Begin -----Users logging in through sshd: brvce: 192.168.1.123 (host4.mydomain.org): 1 time ----- SSHD End ---------- Sudo (secure-log) Begin -----bryce => root_____ /bin/bash - 1 Time(s). ----- Sudo (secure-log) End ----------- Disk Space Begin -----Filesystem Size Used Avail Use% Mounted on /dev/sdc1 220G 19G 190G 9% / /dev/loop1 157M 157M 0 100% /snap/gnome-3-28-1804/110 /dev/loop11 1.0M 1.0M 0 100% /snap/gnome-logs/81 /dev/md5 9.1T 7.3T 1.8T 81% /srv/Products /dev/md6 9.1T 5.6T 3.5T 62% /srv/Archives /dev/loop14 3.8M 3.8M 0 100% /snap/gnome-system-monitor/127 /dev/loop17 15M 15M 0 100% /snap/gnome-characters/399 /dev/loop18 161M 161M 0 100% /snap/gnome-3-28-1804/116 /dev/loop6 55M 55M 0 100% /snap/core18/1668 /dev/md1 1.8T 1.3T 548G 71% /srv/Staff /dev/md0 3.6T 3.5T 84G 98% /srv/Backup /dev/loop2 1.0M 1.0M 0 100% /snap/gnome-logs/93 /dev/loop5 15M 15M 0 100% /snap/gnome-characters/495 /dev/loop8 3.8M 3.8M 0 100% /snap/gnome-system-monitor/135 /dev/md7 3.6T 495G 3.0T 15% /srv/Customers /dev/loop9 55M 55M 0 100% /snap/core18/1705 /dev/loop10 94M 94M 0 100% /snap/core/8935 /dev/loop0 55M 55M 0 100% /snap/gtk-common-themes/1502 /dev/loop4 63M 63M 0 100% /snap/gtk-common-themes/1506 /dev/loop3 94M 94M 0 100% /snap/core/9066 /srv/Backup (/dev/md0) => 98% Used. Warning. Disk Filling up.

----- Disk Space End -----

Backups

There are many ways to backup an Ubuntu installation. The most important thing about backups is to develop a *backup plan* consisting of what to backup, where to back it up to, and how to restore it.

It is good practice to take backup media off-site in case of a disaster. For backup plans involving physical tape or removable hard drives, the tapes or drives can be manually taken off-site, but in other cases this may not be practical and the archives will need copied over a WAN link to a server in another location.

Archive Rotation

The shell script in Shell Scripts only allows for seven different archives. For a server whose data doesn't change often, this may be enough. If the server has a large amount of data, a more complex rotation scheme should be used.

Rotating NFS Archives

In this section, the shell script will be slightly modified to implement a grandfather-father-son rotation scheme (monthly-weekly-daily):

- The rotation will do a *daily* backup Sunday through Friday.
- On Saturday a *weekly* backup is done giving you four weekly backups a month.

• The *monthly* backup is done on the first of the month rotating two monthly backups based on if the month is odd or even.

Here is the new script:

```
#!/bin/bash
#
# Backup to NFS mount script with
# grandfather-father-son rotation.
# What to backup.
backup_files="/home /var/spool/mail /etc /root /boot /opt"
# Where to backup to.
dest="/mnt/backup"
# Setup variables for the archive filename.
day=$(date +%A)
hostname=$(hostname -s)
# Find which week of the month 1-4 it is.
day num=$(date +%-d)
if (( $day num <= 7 )); then
       week_file="$hostname-week1.tgz"
elif (( $day_num > 7 && $day_num <= 14 )); then
       week_file="$hostname-week2.tgz"
elif (( $day_num > 14 && $day_num <= 21 )); then
       week_file="$hostname-week3.tgz"
elif (( $day_num > 21 && $day_num < 32 )); then
       week_file="$hostname-week4.tgz"
fi
# Find if the Month is odd or even.
month_num=$(date +%m)
month=$(expr $month num % 2)
if [ $month -eq 0 ]; then
       month_file="$hostname-month2.tgz"
else
       month_file="$hostname-month1.tgz"
fi
# Create archive filename.
if [ $day num == 1 ]; then
   archive_file=$month_file
elif [ $day != "Saturday" ]; then
       archive_file="$hostname-$day.tgz"
else
   archive_file=$week_file
fi
# Print start status message.
echo "Backing up $backup_files to $dest/$archive_file"
date
echo
# Backup the files using tar.
tar czf $dest/$archive_file $backup_files
# Print end status message.
echo
echo "Backup finished"
```

date

Long listing of files in \$dest to check file sizes.
ls -lh \$dest/

The script can be executed using the same methods as in Executing the Script.

As discussed in the introduction, a copy of the backup archives and/or media can then be transferred off-site.

Tape Drives

A tape drive attached to the server can be used instead of an NFS share. Using a tape drive simplifies archive rotation, and makes taking the media off-site easier as well.

When using a tape drive, the filename portions of the script aren't needed because the data is sent directly to the tape device. Some commands to manipulate the tape are needed. This is accomplished using mt, a magnetic tape control utility part of the cpio package.

Here is the shell script modified to use a tape drive:

```
#!/bin/bash
#
# Backup to tape drive script.
#
# What to backup.
backup_files="/home /var/spool/mail /etc /root /boot /opt"
# Where to backup to.
dest="/dev/st0"
# Print start status message.
echo "Backing up $backup_files to $dest"
date
echo
# Make sure the tape is rewound.
mt -f $dest rewind
# Backup the files using tar.
tar czf $dest $backup_files
# Rewind and eject the tape.
mt -f $dest rewoffl
# Print end status message.
echo
echo "Backup finished"
date
```

Note

The default device name for a SCSI tape drive is /dev/st0. Use the appropriate device path for your system.

Restoring from a tape drive is basically the same as restoring from a file. Simply rewind the tape and use the device path instead of a file path. For example to restore the /etc/hosts file to /tmp/etc/hosts:

```
mt -f /dev/st0 rewind
tar -xzf /dev/st0 -C /tmp etc/hosts
```

Bacula

Bacula is a backup program enabling you to backup, restore, and verify data across your network. There are Bacula clients for Linux, Windows, and Mac OS X - making it a cross-platform network wide solution.

Overview

Bacula is made up of several components and services used to manage which files to backup and backup locations:

- Bacula Director: a service that controls all backup, restore, verify, and archive operations.
- Bacula Console: an application allowing communication with the Director. There are three versions of the Console:
 - Text based command line version.
 - Gnome based GTK+ Graphical User Interface (GUI) interface.
 - wxWidgets GUI interface.
- Bacula File: also known as the Bacula Client program. This application is installed on machines to be backed up, and is responsible for the data requested by the Director.
- Bacula Storage: the programs that perform the storage and recovery of data to the physical media.
- Bacula Catalog: is responsible for maintaining the file indexes and volume databases for all files backed up, enabling quick location and restoration of archived files. The Catalog supports three different databases MySQL, PostgreSQL, and SQLite.
- Bacula Monitor: allows the monitoring of the Director, File daemons, and Storage daemons. Currently the Monitor is only available as a GTK+ GUI application.

These services and applications can be run on multiple servers and clients, or they can be installed on one machine if backing up a single disk or volume.

Installation

Note

If using MySQL or PostgreSQL as your database, you should already have the services available. Bacula will not install them for you. For more information take a look at Databases -MySQL and Databases - PostgreSQL.

There are multiple packages containing the different Bacula components. To install Bacula, from a terminal prompt enter:

sudo apt install bacula

By default installing the bacula package will use a PostgreSQL database for the Catalog. If you want to use SQLite or MySQL, for the Catalog, install bacula-director-sqlite3 or bacula-director-mysql respectively.

During the install process you will be asked to supply a password for the database owner of the bacula database.

Configuration

Bacula configuration files are formatted based on *resources* comprising of *directives* surrounded by "{}" braces. Each Bacula component has an individual file in the /etc/bacula directory.

The various Bacula components must authorize themselves to each other. This is accomplished using the *password* directive. For example, the *Storage* resource password in the /etc/bacula/bacula-dir.conf file must match the *Director* resource password in /etc/bacula/bacula-sd.conf.

By default the backup job named *BackupClient1* is configured to archive the Bacula Catalog. If you plan on using the server to backup more than one client you should change the name of this job to something more descriptive. To change the name edit /etc/bacula/bacula-dir.conf:

```
#
#
# Define the main nightly save backup job
# By default, this job will back up to disk in
Job {
    Name = "BackupServer"
    JobDefs = "DefaultJob"
    Write Bootstrap = "/var/lib/bacula/Client1.bsr"
}
```

Note

The example above changes the job name to *BackupServer* matching the machine's host name. Replace "BackupServer" with your appropriate hostname, or other descriptive name.

The *Console* can be used to query the *Director* about jobs, but to use the Console with a *non-root* user, the user needs to be in the *bacula* group. To add a user to the bacula group enter the following from a terminal:

sudo adduser \$username bacula

Note

Replace *\$username* with the actual username. Also, if you are adding the current user to the group you should log out and back in for the new permissions to take effect.

Localhost Backup

This section describes how to backup specified directories on a single host to a local tape drive.

• First, the *Storage* device needs to be configured. Edit /etc/bacula/bacula-sd.conf add:

```
Device {
  Name = "Tape Drive"
  Device Type = tape
  Media Type = DDS-4
  Archive Device = /dev/st0
  Hardware end of medium = No;
  AutomaticMount = yes;  # when device opened, read it
  AlwaysOpen = Yes;
  RemovableMedia = yes;
  RandomAccess = no;
  Alert Command = "sh -c 'tapeinfo -f %c | grep TapeAlert'"
}
```

The example is for a DDS-4 tape drive. Adjust the "Media Type" and "Archive Device" to match your hardware.

You could also uncomment one of the other examples in the file.

• After editing /etc/bacula/bacula-sd.conf the Storage daemon will need to be restarted:

```
sudo systemctl restart bacula-sd.service
```

• Now add a *Storage* resource in /etc/bacula/bacula-dir.conf to use the new Device:

The *Address* directive needs to be the Fully Qualified Domain Name (FQDN) of the server. Change *backupserver* to the actual host name.

Also, make sure the *Password* directive matches the password string in /etc/bacula/bacula-sd.conf.

• Create a new *FileSet*, which will determine what directories to backup, by adding:

```
# LocalhostBacup FileSet.
FileSet {
   Name = "LocalhostFiles"
   Include {
        Options {
            signature = MD5
            compression=GZIP
        }
        File = /etc
        File = /home
   }
}
```

This *FileSet* will backup the /etc and /home directories. The *Options* resource directives configure the FileSet to create an MD5 signature for each file backed up, and to compress the files using GZIP.

• Next, create a new *Schedule* for the backup job:

```
# LocalhostBackup Schedule -- Daily.
Schedule {
  Name = "LocalhostDaily"
  Run = Full daily at 00:01
}
```

The job will run every day at 00:01 or 12:01 am. There are many other scheduling options available.

• Finally create the *Job*:

```
# Localhost backup.
Job {
    Name = "LocalhostBackup"
    JobDefs = "DefaultJob"
    Enabled = yes
    Level = Full
    FileSet = "LocalhostFiles"
    Schedule = "LocalhostDaily"
    Storage = TapeDrive
    Write Bootstrap = "/var/lib/bacula/LocalhostBackup.bsr"
}
```

The job will do a *Full* backup every day to the tape drive.

• Each tape used will need to have a *Label*. If the current tape does not have a label Bacula will send an email letting you know. To label a tape using the Console enter the following from a terminal:

bconsole

• At the Bacula Console prompt enter:

label

• You will then be prompted for the *Storage* resource:

```
Automatically selected Catalog: MyCatalog
Using Catalog "MyCatalog"
The defined Storage resources are:
1: File
2: TapeDrive
Select Storage resource (1-2):2
```

• Enter the new *Volume* name:

```
Enter new Volume name: Sunday
Defined Pools:
1: Default
2: Scratch
```

Replace *Sunday* with the desired label.

• Now, select the *Pool*:

```
Select the Pool (1-2): 1
Connecting to Storage daemon TapeDrive at backupserver:9103 ...
Sending label command for Volume "Sunday" Slot 0 ...
```

Congratulations, you have now configured Bacula to backup the localhost to an attached tape drive.

Resources

- For more *Bacula* configuration options, refer to Bacula's Documentation.
- The Bacula Home Page contains the latest Bacula news and developments.
- Also, see the Bacula Ubuntu Wiki page.

Shell Scripts

One of the simplest ways to backup a system is using a *shell script*. For example, a script can be used to configure which directories to backup, and pass those directories as arguments to the tar utility, which creates an archive file. The archive file can then be moved or copied to another location. The archive can also be created on a remote file system such as an *NFS* mount.

The tar utility creates one archive file out of many files or directories. tar can also filter the files through compression utilities, thus reducing the size of the archive file.

Simple Shell Script

The following shell script uses tar to create an archive file on a remotely mounted NFS file system. The archive filename is determined using additional command line utilities.

```
#!/bin/bash
#
# Backup to NFS mount script.
# What to backup.
backup_files="/home /var/spool/mail /etc /root /boot /opt"
# Where to backup to.
dest="/mnt/backup"
# Create archive filename.
day=$(date +%A)
hostname=$(hostname -s)
archive file="$hostname-$day.tgz"
# Print start status message.
echo "Backing up $backup files to $dest/$archive file"
date
echo
# Backup the files using tar.
tar czf $dest/$archive file $backup files
# Print end status message.
echo
echo "Backup finished"
date
# Long listing of files in $dest to check file sizes.
ls -lh $dest
```

- *\$backup_files:* a variable listing which directories you would like to backup. The list should be customized to fit your needs.
- *\$day:* a variable holding the day of the week (Monday, Tuesday, Wednesday, etc). This is used to create an archive file for each day of the week, giving a backup history of seven days. There are other ways to accomplish this including using the date utility.
- *Shostname:* variable containing the *short* hostname of the system. Using the hostname in the archive filename gives you the option of placing daily archive files from multiple systems in the same directory.
- *\$archive_file:* the full archive filename.
- *\$dest:* destination of the archive file. The directory needs to be created and in this case *mounted* before executing the backup script. See ??? for details of using *NFS*.
- status messages: optional messages printed to the console using the echo utility.
- tar czf \$dest/\$archive_file \$backup_files: the tar command used to create the archive file.

- c: creates an archive.
- -z: filter the archive through the gzip utility compressing the archive.
- -f: output to an archive file. Otherwise the tar output will be sent to STDOUT.
- *ls -lh \$dest:* optional statement prints a *-l* long listing in *-h* human readable format of the destination directory. This is useful for a quick file size check of the archive file. This check should not replace testing the archive file.

This is a simple example of a backup shell script; however there are many options that can be included in such a script. See References for links to resources providing more in-depth shell scripting information.

Executing the Script

Executing from a Terminal

The simplest way of executing the above backup script is to copy and paste the contents into a file. backup.sh for example. The file must be made executable:

chmod u+x backup.sh

Then from a terminal prompt:

sudo ./backup.sh

This is a great way to test the script to make sure everything works as expected.

Executing with cron

The cron utility can be used to automate the script execution. The cron daemon allows the execution of scripts, or commands, at a specified time and date.

cron is configured through entries in a crontab file. crontab files are separated into fields:

m h dom mon dow command

- *m*: minute the command executes on, between 0 and 59.
- *h:* hour the command executes on, between 0 and 23.
- *dom:* day of month the command executes on.
- *mon:* the month the command executes on, between 1 and 12.
- *dow:* the day of the week the command executes on, between 0 and 7. Sunday may be specified by using 0 or 7, both values are valid.
- *command:* the command to execute.

To add or change entries in a crontab file the crontab -e command should be used. Also, the contents of a crontab file can be viewed using the crontab -l command.

To execute the backup.sh script listed above using cron. Enter the following from a terminal prompt:

sudo crontab -e

Note

Using sudo with the crontab -e command edits the *root* user's crontab. This is necessary if you are backing up directories only the root user has access to.

Add the following entry to the crontab file:

```
# m h dom mon dow command
0 0 * * * bash /usr/local/bin/backup.sh
```

The backup.sh script will now be executed every day at 12:00 pm.

Note

The backup.sh script will need to be copied to the /usr/local/bin/ directory in order for this entry to execute properly. The script can reside anywhere on the file system, simply change the script path appropriately.

For more in-depth crontab options see References.

Restoring from the Archive

Once an archive has been created it is important to test the archive. The archive can be tested by listing the files it contains, but the best test is to *restore* a file from the archive.

• To see a listing of the archive contents. From a terminal prompt type:

tar -tzvf /mnt/backup/host-Monday.tgz

• To restore a file from the archive to a different directory enter:

tar -xzvf /mnt/backup/host-Monday.tgz -C /tmp etc/hosts

The -C option to tar redirects the extracted files to the specified directory. The above example will extract the /etc/hosts file to /tmp/etc/hosts. tar recreates the directory structure that it contains.

Also, notice the leading "/" is left off the path of the file to restore.

• To restore all files in the archive enter the following:

```
cd /
sudo tar -xzvf /mnt/backup/host-Monday.tgz
```

Note

This will overwrite the files currently on the file system.

References

- For more information on shell scripting see the Advanced Bash-Scripting Guide
- The CronHowto Wiki Page contains details on advanced cron options.
- See the GNU tar Manual for more tar options.
- The Wikipedia Backup Rotation Scheme article contains information on other backup rotation schemes.
- The shell script uses tar to create the archive, but there many other command line utilities that can be used. For example:
 - cpio: used to copy files to and from archives.
 - dd: part of the coreutils package. A low level utility that can copy data from one format to another.
 - rsnapshot: a file system snapshot utility used to create copies of an entire file system. Also check the Tools
 rsnapshot for some information.
 - rsync: a flexible utility used to create incremental copies of files.

Rsnapshot

Rsnapshot is an rsync-based filesystem snapshot utility. It can take incremental backups of local and remote filesystems for any number of machines. Rsnapshot makes extensive use of hard links, so disk space is only used when absolutely necessary. It leverages the power of rsync to create scheduled, incremental backups.

To install it open a terminal shell and run:

sudo apt-get install rsnapshot

If you want to backup a remote filesystem the rsnapshot server needs to be able to access the target machine over SSH without password. For more information on how to enable it please see OpenSSH documentation. If the backup target is a local filesystem there is no need to set up OpenSSH.

Configuration

The rsnapshot configuration resides in /etc/rsnapshot.conf. Below you can find some of the options available there.

The root directory where all snapshots will be stored:

snapshot_root /var/cache/rsnapshot/

How many old backups you would like to keep. Since rsnapshot uses incremental backups, we can afford to keep older backups for awhile before removing them. You set these up under the BACKUP LEVELS / INTERVALS section. You tell rsnapshot to retain a specific number of backups of each kind of interval.

retain daily 6 retain weekly 7 retain monthly 4

In this example we will keep 6 snapshots of our daily strategy, 7 snapshots of our weekly strategy, and 4 snapshots of our monthly strategy. These data will guide the rotation made by rsnapshot.

If you are accessing a remote machine over SSH and the port to bind is not the default (port 22), you need to set the following variable with the port number:

ssh_args -p 22222

And the most important part, you need to decide on what you would like to backup. If you are backing up locally to the same machine, this is as easy as specifying the directories that you want to save and following it with localhost/ which will be a sub-directory in the snapshot_root that you set up earlier.

backup	/home/	localhost/
backup	/etc/	localhost/
backup	/usr/local/	localhost/

If you are backing up a remote machine you just need to tell rsnapshot where the server is and which directories you would like to back up.

backup root@example.com:/home/ example.com/ +rsync_long_args=--bwlimit=16,exclude=core backup root@example.com:/etc/ example.com/ exclude=mtab,exclude=core

As you can see you can see you can pass extra rsync parameters (the + append the parameter to the default list, if you remove the + sign you override it) and also exclude directories.

You can check the comments in /etc/rsnapshot.conf and the rsnapshot man page for more options.

Test Configuration

After modifying the configuration file is a good practice to check if the syntax is ok:

sudo rsnapshot configtest

You can also test your backup levels with the following command:

sudo rsnapshot -t daily

If you are happy with the output and want to see it in action you can run:

sudo rsnapshot daily

Scheduling Backups

With rsnapshot working correctly with the current configuration, the only thing left to do is to schedule it to run at certain intervals. We will use cron to make this happen since rsnapshot includes a default cron file in /etc/cron.d/rsnapshot. If you open this file there are some entries commented out as reference.

0	4	*	*	*	root	/usr/bin/rsnapshot	daily
0	3	*	*	1	root	/usr/bin/rsnapshot	weekly
0	2	1	*	*	root	/usr/bin/rsnapshot	monthly

The settings above added to /etc/cron.d/rsnapshot run:

- the **daily snapshot** everyday at 4:00 am
- the weekly snapshot every Monday at 3:00 am
- the monthly snapshot on the first of every month at 2:00 am

For more information on how to schedule a backup using cron please take a look at Executing with cron section in Backups - Shell Scripts.

References

- Rsnapshot offical web page
- Rsnapshot man page
- Rsync man page

Email Services

The process of getting an email from one person to another over a network or the Internet involves many systems working together. Each of these systems must be correctly configured for the process to work. The sender uses a *Mail User Agent* (MUA), or email client, to send the message through one or more *Mail Transfer Agents* (MTA), the last of which will hand it off to a *Mail Delivery Agent* (MDA) for delivery to the recipient's mailbox, from which it will be retrieved by the recipient's email client, usually via a POP3 or IMAP server.

On Ubuntu, Postfix is the default, supported MTA.

Dovecot Server

Dovecot is a Mail Delivery Agent, written with security primarily in mind. It supports the major mailbox formats: mbox or Maildir. This section explains how to set it up as an IMAP or POP3 server.

Installation

To install a basic Dovecot server with common POP3 and IMAP functions, run the following command:

sudo apt install dovecot-imapd dovecot-pop3d

There are various other Dovecot modules including dovecot-sieve (mail filtering), dovecot-solr (full text search), dovecot-antispam (spam filter training), dovecot-ldap (user directory).

Configuration

To configure Dovecot, edit the file /etc/dovecot/dovecot.conf and its included config files in /etc/dovecot/conf.d/. By default all installed protocols will be enabled via an include directive in /etc/dovecot/dovecot.conf.

!include_try /usr/share/dovecot/protocols.d/*.protocol

IMAPS and POP3S are more secure because they use SSL encryption to connect. A basic self-signed ssl certificate is automatically set up by package ssl-cert and used by Dovecot in /etc/dovecot/conf.d/10-ssl.conf.

By default mbox format is configured, if required you can also use Maildir. More about that can be found in the comments in /etc/dovecot/conf.d//10-mail.conf. Also see the Dovecot web site to learn about further benefits and details.

Make sure to also configure your Mail Transport Agent (MTA) to transfer the incoming mail to the selected type of mailbox.

Once you have configured Dovecot, restart its daemon in order to test your setup:

```
sudo service dovecot restart
```

Try to log in with the commands telnet localhost pop3 (for POP3) or telnet localhost imap2 (for IMAP). You should see something like the following:

```
bhuvan@rainbow:~$ telnet localhost pop3
Trying 127.0.0.1...
Connected to localhost.localdomain.
Escape character is '^]'.
+OK Dovecot ready.
```

Dovecot SSL Configuration

Dovecot is configured to use SSL automatically by default, using the package ssl-cert which provides a self signed certificate.

You can instead generate your own custom certificate for Dovecot using <code>openssh</code>, for example:

See certificates-and-security for more details on creating custom certificates.

Then edit /etc/dovecot/conf.d/10-ssl.conf and amend following lines to specify Dovecat use these custom certificates :

ssl_cert = </etc/dovecot/private/dovecot.pem
ssl_key = </etc/dovecot/private/dovecot.key</pre>

You can get the SSL certificate from a Certificate Issuing Authority or you can create self-signed one (see certificatesand-security for details). Once you create the certificate, you will have a key file and a certificate file that you want to make known in the config shown above.

Firewall Configuration for an Email Server

To access your mail server from another computer, you must configure your firewall to allow connections to the server on the necessary ports.

- IMAP 143
- IMAPS 993
- POP3 110
- POP3S 995

References

- See the Dovecot website for more information.
- Also, the Dovecot Ubuntu Wiki page has more details.

Exim4

Exim4 is a Message Transfer Agent (MTA) developed at the University of Cambridge for use on Unix systems connected to the Internet. Exim can be installed in place of sendmail, although its configuration is quite different.

Installation

To install exim4, run the following command:

sudo apt install exim4

Configuration

To configure Exim4, run the following command:

```
sudo dpkg-reconfigure exim4-config
```

This displays a user interface "wizard" for configuring the software. For example, in Exim4 the configuration files are split among multiple files; if you wish to have them in one file you can configure accordingly via this user interface.

All the configurable parameters from the user interface are stored in /etc/exim4/update-exim4.conf.conf file, so to re-configure you can either re-run the wizard or manually edit this file using your favorite editor. Once you are finished, you can run the following command to generate the master configuration file:

sudo update-exim4.conf

The master configuration file is stored in /var/lib/exim4/config.autogenerated.

Warning

At any time, you should not manually edit the master configuration file, /var/lib/exim4/config.autogenerated, because it is updated automatically every time you run update-exim4.conf, so your changes will risk being accidentally lost during a future update.

The following command will start the Exim4 daemon:

sudo service exim4 start

SMTP Authentication

Exim4 can be configured to use SMTP-AUTH with TLS and SASL.

First, enter the following into a terminal prompt to create a certificate for use with TLS:

sudo /usr/share/doc/exim4-base/examples/exim-gencert

Configure Exim4 for TLS by editing /etc/exim4/conf.d/main/03_exim4-config_tlsoptions and adding the following: MAIN_TLS_ENABLE = yes Second, configure Exim4 to use the saslauthd for authentication by editing /etc/exim4/conf.d/auth/30_exim4-config_examples and uncomment the *plain_saslauthd_server* and *login_saslauthd_server* sections:

```
plain_saslauthd_server:
   driver = plaintext
   public_name = PLAIN
   server_condition = ${if saslauthd{{$auth2}{$auth3}}{1}{0}}
   server_set_id = $auth2
   server_prompts = :
   .ifndef AUTH SERVER ALLOW NOTLS PASSWORDS
   server_advertise_condition = ${if eq{$tls_cipher}{}}
   .endif
#
 login saslauthd server:
   driver = plaintext
   public_name = LOGIN
   server_prompts = "Username:: : Password::"
   # don't send system passwords over unencrypted connections
   server_condition = ${if saslauthd{{$auth1}{$auth2}}{1}{0}}
   server_set_id = $auth1
   .ifndef AUTH_SERVER_ALLOW_NOTLS_PASSWORDS
   server_advertise_condition = ${if eq{$tls_cipher}{}{*}}
   .endif
```

Additionally, to enable outside mail clients to connect to the new server, a new user needs to be added into exim by using the following commands.

sudo /usr/share/doc/exim4-base/examples/exim-adduser

Protect the new password files with the following commands:

```
sudo chown root:Debian-exim /etc/exim4/passwd
sudo chmod 640 /etc/exim4/passwd
```

Finally, update the Exim4 configuration and restart the service:

```
sudo update-exim4.conf
sudo systemctl restart exim4.service
```

Configuring SASL

To configure the saslauth tto provide authentication for Exim4, first install the sasl2-bin package by running this command at a terminal prompt:

sudo apt install sasl2-bin

To configure saslauthd, edit the /etc/default/saslauthd configuration file and set:

START=yes

Next, to make Exim4 use the saslauthd service, the *Debian-exim* user needs to be part of the sasl group:

sudo adduser Debian-exim sasl

Finally, start the saslauthd service:

sudo service saslauthd start

Exim4 is now configured with SMTP-AUTH using TLS and SASL authentication.

References

- See exim.org for more information.
- There is also an Exim4 Book available.
- Another resource is the Exim4 Ubuntu Wiki page.
- Further resources to set up mailman3 with exim4

Postfix

Postfix is the default Mail Transfer Agent (MTA) in Ubuntu. It attempts to be fast and secure, with flexibility in administration. It is compatible with the MTA sendmail. This section will explain installation, including how to configure SMTP for secure communications.

Note

This guide does not cover setting up Postfix *Virtual Domains*, for information on Virtual Domains and other advanced configurations see References.

Installation

To install Postfix run the following command:

```
sudo apt install postfix
```

For now, it is ok to simply accept defaults by pressing return for each question. Some of the configuration options will be investigated in greater detail in the next stage.

Deprecation warning: please note that the mail-stack-delivery metapackage has been deprecated in Focal. The package still exists for compatibility reasons, but won't setup a working email system.

Basic Configuration

There are four things you should decide before starting configuration:

- The <Domain> for which you'll accept email (we'll use mail.example.com in our example)
- The network and class range of your mail server (we'll use 192.168.0.0/24)
- The username (we're using *steve*)
- Type of mailbox format (mbox is default, we'll use the alternative, Maildir)

To configure postfix, run the following command:

```
sudo dpkg-reconfigure postfix
```

The user interface will be displayed. On each screen, select the following values:

- Internet Site
- mail.example.com
- steve
- mail.example.com, localhost.localdomain, localhost
- No
- 127.0.0.0/8 [::fff:127.0.0.0]/104 [::1]/128 192.168.0.0/24
- 0
- +
- all

To set the mailbox format, you can either edit the configuration file directly, or use the postconf command. In either case, the configuration parameters will be stored in /etc/postfix/main.cf file. Later if you wish to re-configure a particular parameter, you can either run the command or change it manually in the file.

To configure the mailbox format for Maildir:

sudo postconf -e 'home_mailbox = Maildir/'

Note

This will place new mail in /home/username/Maildir so you will need to configure your Mail Delivery Agent (MDA) to use the same path.

SMTP Authentication

SMTP-AUTH allows a client to identify itself through the SASL authentication mechanism, using Transport Layer Security (TLS) to encrypt the authentication process. Once authenticated the SMTP server will allow the client to relay mail.

To configure Postfix for SMTP-AUTH using SASL (Dovecot SASL), run these commands at a terminal prompt:

```
sudo postconf -e 'smtpd_sasl_type = dovecot'
sudo postconf -e 'smtpd_sasl_path = private/auth'
sudo postconf -e 'smtpd_sasl_local_domain ='
```

```
sudo postconf -e 'smtpd_sasl_security_options = noanonymous,noplaintext'
sudo postconf -e 'smtpd_sasl_tls_security_options = noanonymous'
sudo postconf -e 'broken_sasl_auth_clients = yes'
sudo postconf -e 'smtpd_sasl_auth_enable = yes'
sudo postconf -e 'smtpd_recipient_restrictions = \
permit_sasl_authenticated,permit_mynetworks,reject_unauth_destination'
```

Note

The *smtpd_sasl_path* config parameter is a path relative to the Postfix queue directory.

There are several SASL mechanism properties worth evaluating to improve the security of your deployment. The options "noanonymous,noplaintext" prevent use of mechanisms that permit anonymous authentication or that transmit credentials unencrypted.

Next, generate or obtain a digital certificate for TLS. See security - certificates in this guide for details about generating digital certificates and setting up your own Certificate Authority (CA).

Note

MUAs connecting to your mail server via TLS will need to recognize the certificate used for TLS. This can either be done using a certificate from Let's Encrypt, from a commercial CA or with a self-signed certificate that users manually install/accept. For MTA to MTA TLS certificates are never validated without advance agreement from the affected organizations. For MTA to MTA TLS, unless local policy requires it, there is no reason not to use a self-signed certificate. Refer to security - certificates in this guide for more details.

Once you have a certificate, configure Postfix to provide TLS encryption for both incoming and outgoing mail:

```
sudo postconf -e 'smtp_tls_security_level = may'
sudo postconf -e 'smtpd_tls_security_level = may'
sudo postconf -e 'smtp_tls_note_starttls_offer = yes'
sudo postconf -e 'smtpd_tls_key_file = /etc/ssl/private/server.key'
sudo postconf -e 'smtpd_tls_cert_file = /etc/ssl/certs/server.crt'
sudo postconf -e 'smtpd_tls_loglevel = 1'
sudo postconf -e 'smtpd_tls_received_header = yes'
sudo postconf -e 'myhostname = mail.example.com'
```

If you are using your own *Certificate Authority* to sign the certificate enter:

sudo postconf -e 'smtpd_tls_CAfile = /etc/ssl/certs/cacert.pem'

Again, for more details about certificates see security - certificates in this guide.

Note

After running all the commands, Postfix is configured for SMTP-AUTH and a self-signed certificate has been created for TLS encryption.

Now, the file /etc/postfix/main.cf should look like this:

```
# See /usr/share/postfix/main.cf.dist for a commented, more complete
# version
smtpd_banner = $myhostname ESMTP $mail_name (Ubuntu)
biff = no
# appending .domain is the MUA's job.
append dot mydomain = no
# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h
myhostname = server1.example.com
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
mydestination = server1.example.com, localhost.example.com, localhost
relayhost =
mynetworks = 127.0.0.0/8
mailbox command = procmail -a "$EXTENSION"
```

```
mailbox_size_limit = 0
recipient delimiter = +
inet interfaces = all
smtpd_sasl_local_domain =
smtpd_sasl_auth_enable = yes
smtpd sasl security options = noanonymous
broken_sasl_auth_clients = yes
smtpd_recipient_restrictions =
permit_sasl_authenticated,permit_mynetworks,reject _unauth_destination
smtpd_tls_auth_only = no
smtp_tls_security_level = may
smtpd tls security level = may
smtp tls note starttls offer = yes
smtpd tls key file = /etc/ssl/private/smtpd.key
smtpd_tls_cert_file = /etc/ssl/certs/smtpd.crt
smtpd_tls_CAfile = /etc/ssl/certs/cacert.pem
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
tls_random_source = dev:/dev/urandom
```

The postfix initial configuration is complete. Run the following command to restart the postfix daemon:

```
sudo systemctl restart postfix.service
```

Postfix supports SMTP-AUTH as defined in RFC2554. It is based on SASL. However it is still necessary to set up SASL authentication before you can use SMTP-AUTH.

When using ipv6, the mynetworks parameter may need to be modified to allow ipv6 addresses, for example:

mynetworks = 127.0.0.0/8, [::1]/128

Configuring SASL

Postfix supports two SASL implementations: *Cyrus SASL* and *Dovecot SASL*. To enable Dovecot SASL the dovecotcore package will need to be installed:

sudo apt install dovecot-core

Next, edit /etc/dovecot/conf.d/10-master.conf and change the following:

```
service auth {
 # auth socket path points to this userdb socket by default. It's typically
 # used by dovecot-lda, doveadm, possibly imap process, etc. Its default
  # permissions make it readable only by root, but you may need to relax these
  # permissions. Users that have access to this socket are able to get a list
  # of all usernames and get results of everyone's userdb lookups.
  unix_listener auth-userdb {
    #mode = 0600
    #user =
    #group =
  }
  # Postfix smtp-auth
  unix_listener /var/spool/postfix/private/auth {
   mode = 0660
    user = postfix
    group = postfix
 }
 }
```

To permit use of SMTP-AUTH by Outlook clients, change the following line in the *authentication mechanisms* section of /etc/dovecot/conf.d/10-auth.conf from:

```
auth_mechanisms = plain
to this:
auth_mechanisms = plain login
```

Once you have Dovecot configured, restart it with:

sudo systemctl restart dovecot.service

Testing

SMTP-AUTH configuration is complete. Now it is time to test the setup.

To see if SMTP-AUTH and TLS work properly, run the following command:

telnet mail.example.com 25

After you have established the connection to the Postfix mail server, type:

ehlo mail.example.com

If you see the following in the output, then everything is working perfectly. Type quit to exit.

250-STARTTLS 250-AUTH LOGIN PLAIN 250-AUTH=LOGIN PLAIN 250 8BITMIME

Troubleshooting

When problems arise, there are a few common ways to diagnose the cause.

Escaping chroot

The Ubuntu Postfix package will by default install into a *chroot* environment for security reasons. This can add greater complexity when troubleshooting problems.

To turn off the chroot usage, locate the following line in the /etc/postfix/master.cf configuration file:

smtp inet n - - - - smtpd

and modify it as follows:

smtp inet n - n - - smtpd

You will then need to restart Postfix to use the new configuration. From a terminal prompt enter:

sudo service postfix restart

\mathbf{SMTPS}

If you need secure SMTP, edit /etc/postfix/master.cf and uncomment the following line:

smtps inet n - - - - - smtpd

```
-o smtpd_tls_wrappermode=yes
```

-o smtpd_sasl_auth_enable=yes

-o smtpd_client_restrictions=permit_sasl_authenticated,reject

-o milter_macro_daemon_name=ORIGINATING

Log Viewing

Postfix sends all log messages to /var/log/mail.log. However, error and warning messages can sometimes get lost in the normal log output so they are also logged to /var/log/mail.err and /var/log/mail.warn respectively.

To see messages entered into the logs in real time you can use the tail -f command:

tail -f /var/log/mail.err

Increasing Logging Detail

The amount of detail that is recorded in the logs can be increased via the configuration options. For example, to increase TLS activity logging set the $smtpd_tls_loglevel$ option to a value from 1 to 4.

sudo postconf -e 'smtpd_tls_loglevel = 4'

Reload the service after any configuration change, to make the new config active:

```
sudo systemctl reload postfix.service
```

Logging mail delivery

If you are having trouble sending or receiving mail from a specific domain you can add the domain to the *de-bug_peer_list* parameter.

```
sudo postconf -e 'debug_peer_list = problem.domain'
sudo systemctl reload postfix.service
```

Increasing daemon verbosity

You can increase the verbosity of any Postfix daemon process by editing the /etc/postfix/master.cf and adding a -v after the entry. For example, edit the *smtp* entry:

smtp unix - - - - - - smtp -v

Then, reload the service as usual:

sudo systemctl reload postfix.service

Logging SASL debug info

To increase the amount of information logged when trouble shooting SASL issues you can set the following options in /etc/dovecot/conf.d/10-logging.conf

auth_debug=yes
auth debug passwords=yes

Just like Postfix if you change a Dovecot configuration the process will need to be reloaded:

sudo systemctl reload dovecot.service

Note

Some of the options above can drastically increase the amount of information sent to the log files. Remember to return the log level back to normal after you have corrected the problem. Then reload the appropriate daemon for the new configuration to take affect.

References

Administering a Postfix server can be a very complicated task. At some point you may need to turn to the Ubuntu community for more experienced help.

- The Postfix website documents all available configuration options.
- O'Reilly's Postfix: The Definitive Guide is rather dated but provides deep background information about configuration options.
- The Ubuntu Wiki Postfix page has more information from a Ubuntu context. There is also a Debian Wiki Postfix page that's a bit more up to date; they also have a set of Postfix Tutorials for different Debian versions.
- Info on how to set up mailman3 with postfix

Web Servers

The primary function of a web server is to store, process and deliver Web pages to clients. The clients communicate with the server sending HTTP requests. Clients, mostly via Web Browsers, request for a specific resources and the server responds with the content of that resource or an error message. The response is usually a Web page such as HTML documents which may include images, style sheets, scripts, and the content in form of text.

When accessing a Web Server, every HTTP request that is received is responded to with a content and a HTTP status code. HTTP status codes are three-digit codes, and are grouped into five different classes. The class of a status code can be quickly identified by its first digit:

- **1xx** : *Informational* Request received, continuing process
- + 2xx: Success The action was successfully received, understood, and accepted
- 3xx : Redirection Further action must be taken in order to complete the request
- 4xx: Client Error The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error The server failed to fulfill an apparently valid request

More information about status code check the RFC 2616.

Implementation

Web Servers are heavily used in the deployment of Web sites and in this scenario we can use two different implementations:

- Static Web Server: The content of the server's response will be the hosted files "as-is".
- Dynamic Web Server: Consist in a Web Server plus an extra software, usually an *application server* and a *database*. For example, to produce the Web pages you see in the Web browser, the application server might fill an HTML template with contents from a database. Due to that we say that the content of the server's response is generated dynamically.

HTTPD - Apache2 Web Server

Apache is the most commonly used Web server on Linux systems. Web servers are used to serve Web pages requested by client computers. Clients typically request and view Web pages using Web browser applications such as Firefox, Opera, Chromium, or Internet Explorer.

Users enter a Uniform Resource Locator (URL) to point to a Web server by means of its Fully Qualified Domain Name (FQDN) and a path to the required resource. For example, to view the home page of the Ubuntu Web site a user will enter only the FQDN:

www.ubuntu.com

To view the community sub-page, a user will enter the FQDN followed by a path:

www.ubuntu.com/community

The most common protocol used to transfer Web pages is the Hyper Text Transfer Protocol (HTTP). Protocols such as Hyper Text Transfer Protocol over Secure Sockets Layer (HTTPS), and File Transfer Protocol (FTP), a protocol for uploading and downloading files, are also supported.

Apache Web Servers are often used in combination with the MySQL database engine, the HyperText Preprocessor (PHP) scripting language, and other popular scripting languages such as Python and Perl. This configuration is termed LAMP (Linux, Apache, MySQL and Perl/Python/PHP) and forms a powerful and robust platform for the development and deployment of Web-based applications.

Installation

The Apache2 web server is available in Ubuntu Linux. To install Apache2:

At a terminal prompt enter the following command:

sudo apt install apache2

Configuration

Apache2 is configured by placing *directives* in plain text configuration files. These *directives* are separated between the following files and directories:

- apache2.conf: the main Apache2 configuration file. Contains settings that are global to Apache2.
- *httpd.conf:* historically the main Apache2 configuration file, named after the httpd daemon. In other distributions (or older versions of Ubuntu), the file might be present. In Ubuntu, all configuration options have been moved to *apache2.conf* and the below referenced directories, and this file no longer exists.
- *conf-available:* this directory contains available configuration files. All files that were previously in /etc/apache2/conf.d should be moved to /etc/apache2/conf-available.
- *conf-enabled:* holds *symlinks* to the files in /etc/apache2/conf-available. When a configuration file is symlinked, it will be enabled the next time apache2 is restarted.
- envvars: file where Apache2 environment variables are set.
- *mods-available:* this directory contains configuration files to both load *modules* and configure them. Not all modules will have specific configuration files, however.
- *mods-enabled:* holds *symlinks* to the files in /etc/apache2/mods-available. When a module configuration file is symlinked it will be enabled the next time apache2 is restarted.
- *ports.conf:* houses the directives that determine which TCP ports Apache2 is listening on.

- *sites-available:* this directory has configuration files for Apache2 *Virtual Hosts.* Virtual Hosts allow Apache2 to be configured for multiple sites that have separate configurations.
- *sites-enabled:* like mods-enabled, **sites-enabled** contains symlinks to the /etc/apache2/sites-available directory. Similarly when a configuration file in sites-available is symlinked, the site configured by it will be active once Apache2 is restarted.
- *magic:* instructions for determining MIME type based on the first few bytes of a file.

In addition, other configuration files may be added using the *Include* directive, and wildcards can be used to include many configuration files. Any directive may be placed in any of these configuration files. Changes to the main configuration files are only recognized by Apache2 when it is started or restarted.

The server also reads a file containing mime document types; the filename is set by the *TypesConfig* directive, typically via /etc/apache2/mods-available/mime.conf, which might also include additions and overrides, and is /etc/mime.types by default.

Basic Settings

This section explains Apache2 server essential configuration parameters. Refer to the Apache2 Documentation for more details.

• Apache2 ships with a virtual-host-friendly default configuration. That is, it is configured with a single default virtual host (using the *VirtualHost* directive) which can be modified or used as-is if you have a single site, or used as a template for additional virtual hosts if you have multiple sites. If left alone, the default virtual host will serve as your default site, or the site users will see if the URL they enter does not match the *ServerName* directive of any of your custom sites. To modify the default virtual host, edit the file /etc/apache2/sites-available/000-default.conf.

Note

The directives set for a virtual host only apply to that particular virtual host. If a directive is set server-wide and not defined within the virtual host settings, the default setting is used. For example, you can define a Webmaster email address and not define individual email addresses for each virtual host.

If you wish to configure a new virtual host or site, copy that file into the same directory with a name you choose. For example:

sudo cp /etc/apache2/sites-available/000-default.conf /etc/apache2/sites-available/mynewsite.conf

Edit the new file to configure the new site using some of the directives described below.

- The ServerAdmin directive specifies the email address to be advertised for the server's administrator. The default value is webmaster@localhost. This should be changed to an email address that is delivered to you (if you are the server's administrator). If your website has a problem, Apache2 will display an error message containing this email address to report the problem to. Find this directive in your site's configuration file in /etc/apache2/sites-available.
- The *Listen* directive specifies the port, and optionally the IP address, Apache2 should listen on. If the IP address is not specified, Apache2 will listen on all IP addresses assigned to the machine it runs on. The default value for the Listen directive is 80. Change this to 127.0.0.1:80 to cause Apache2 to listen only on your loopback interface so that it will not be available to the Internet, to (for example) 81 to change the port that it listens on, or leave it as is for normal operation. This directive can be found and changed in its own file, /etc/apache2/ports.conf
- The *ServerName* directive is optional and specifies what FQDN your site should answer to. The default virtual host has no ServerName directive specified, so it will respond to all requests that do not match a ServerName directive in another virtual host. If you have just acquired the domain name mynewsite.com and wish to host it on your Ubuntu server, the value of the ServerName directive in your virtual host configuration file should be mynewsite.com. Add this directive to the new virtual host file you created earlier (/etc/apache2/sites-available/mynewsite.comf).

You may also want your site to respond to www.mynewsite.com, since many users will assume the www prefix is appropriate. Use the *ServerAlias* directive for this. You may also use wildcards in the ServerAlias directive.

For example, the following configuration will cause your site to respond to any domain request ending in *.mynewsite.com*.

ServerAlias *.mynewsite.com

• The *DocumentRoot* directive specifies where Apache2 should look for the files that make up the site. The default value is /var/www/html, as specified in /etc/apache2/sites-available/000-default.conf. If desired, change this value in your site's virtual host file, and remember to create that directory if necessary!

Enable the new *VirtualHost* using the a2ensite utility and restart Apache2:

sudo a2ensite mynewsite
sudo systemctl restart apache2.service

Note

Be sure to replace *mynewsite* with a more descriptive name for the VirtualHost. One method is to name the file after the *ServerName* directive of the VirtualHost.

Similarly, use the a2dissite utility to disable sites. This is can be useful when troubleshooting configuration problems with multiple VirtualHosts:

sudo a2dissite mynewsite
sudo systemctl restart apache2.service

Default Settings

This section explains configuration of the Apache2 server default settings. For example, if you add a virtual host, the settings you configure for the virtual host take precedence for that virtual host. For a directive not defined within the virtual host settings, the default value is used.

• The *DirectoryIndex* is the default page served by the server when a user requests an index of a directory by specifying a forward slash (/) at the end of the directory name.

For example, when a user requests the page http://www.example.com/this_directory/, he or she will get either the DirectoryIndex page if it exists, a server-generated directory list if it does not and the Indexes option is specified, or a Permission Denied page if neither is true. The server will try to find one of the files listed in the DirectoryIndex directive and will return the first one it finds. If it does not find any of these files and if *Options Indexes* is set for that directory, the server will generate and return a list, in HTML format, of the subdirectories and files in the directory. The default value, found in /etc/apache2/mods-available/dir.conf is "index.html index.cgi index.php index.php index.xhtml index.htm". Thus, if Apache2 finds a file in a requested directory matching any of these names, the first will be displayed.

- The *ErrorDocument* directive allows you to specify a file for Apache2 to use for specific error events. For example, if a user requests a resource that does not exist, a 404 error will occur. By default, Apache2 will simply return a HTTP 404 Return code. Read /etc/apache2/conf-available/localized-error-pages.conf for detailed instructions for using ErrorDocument, including locations of example files.
- By default, the server writes the transfer log to the file /var/log/apache2/access.log. You can change this on a per-site basis in your virtual host configuration files with the *CustomLog* directive, or omit it to accept the default, specified in /etc/apache2/conf-available/other-vhosts-access-log.conf. You may also specify the file to which errors are logged, via the *ErrorLog* directive, whose default is /var/log/apache2/error.log. These are kept separate from the transfer logs to aid in troubleshooting problems with your Apache2 server. You may also specify the *LogLevel* (the default value is "warn") and the *LogFormat* (see /etc/apache2/apache2.conf for the default value).
- Some options are specified on a per-directory basis rather than per-server. *Options* is one of these directives. A Directory stanza is enclosed in XML-like tags, like so:

<Directory /var/www/html/mynewsite>

</Directory>

. . .

The *Options* directive within a Directory stanza accepts one or more of the following values (among others), separated by spaces:

- ExecCGI - Allow execution of CGI scripts. CGI scripts are not executed if this option is not chosen.

Caution

Most files should not be executed as CGI scripts. This would be very dangerous. CGI scripts should kept in a directory separate from and outside your DocumentRoot, and only this directory should have the ExecCGI option set. This is the default, and the default location for CGI scripts is /usr/lib/cgi-bin.

- **Includes** - Allow server-side includes. Server-side includes allow an HTML file to *include* other files. See Apache SSI documentation (Ubuntu community) for more information.

- IncludesNOEXEC Allow server-side includes, but disable the *#exec* and *#include* commands in CGI scripts.
- **Indexes** Display a formatted list of the directory's contents, if no *DirectoryIndex* (such as index.html) exists in the requested directory.

Caution

For security reasons, this should usually not be set, and certainly should not be set on your DocumentRoot directory. Enable this option carefully on a per-directory basis only if you are certain you want users to see the entire contents of the directory.

- Multiview Support content-negotiated multiviews; this option is disabled by default for security reasons.
 See the Apache2 documentation on this option.
- SymLinksIfOwnerMatch Only follow symbolic links if the target file or directory has the same owner as the link.

apache2 Settings

This section explains some basic apache2 daemon configuration settings.

LockFile - The LockFile directive sets the path to the lockfile used when the server is compiled with either USE_FCNTL_SERIALIZED_ACCEPT or USE_FLOCK_SERIALIZED_ACCEPT. It must be stored on the local disk. It should be left to the default value unless the logs directory is located on an NFS share. If this is the case, the default value should be changed to a location on the local disk and to a directory that is readable only by root.

PidFile - The PidFile directive sets the file in which the server records its process ID (pid). This file should only be readable by root. In most cases, it should be left to the default value.

User - The User directive sets the userid used by the server to answer requests. This setting determines the server's access. Any files inaccessible to this user will also be inaccessible to your website's visitors. The default value for User is "www-data".

Warning

Unless you know exactly what you are doing, do not set the User directive to root. Using root as the User will create large security holes for your Web server.

Group - The Group directive is similar to the User directive. Group sets the group under which the server will answer requests. The default group is also "www-data".

Apache2 Modules

Apache2 is a modular server. This implies that only the most basic functionality is included in the core server. Extended features are available through modules which can be loaded into Apache2. By default, a base set of modules is included in the server at compile-time. If the server is compiled to use dynamically loaded modules, then modules can be compiled separately, and added at any time using the LoadModule directive. Otherwise, Apache2 must be recompiled to add or remove modules.

Ubuntu compiles Apache2 to allow the dynamic loading of modules. Configuration directives may be conditionally included on the presence of a particular module by enclosing them in an $\langle IfModule \rangle$ block.

You can install additional Apache2 modules and use them with your Web server. For example, run the following command at a terminal prompt to install the Python 3 WSGI module:

sudo apt install libapache2-mod-wsgi-py3

The installation will enable the module automatically, but we can disable it with a2dismod:

sudo a2dismod wsgi
sudo systemctl restart apache2.service
And then use the a2enmod utility to re-enable it:

```
sudo a2enmod wsgi
sudo systemctl restart apache2.service
```

See the /etc/apache2/mods-available directory for additional modules already available on your system.

HTTPS Configuration

The mod_ssl module adds an important feature to the Apache2 server - the ability to encrypt communications. Thus, when your browser is communicating using SSL, the https:// prefix is used at the beginning of the Uniform Resource Locator (URL) in the browser navigation bar.

The mod_ssl module is available in apache2-common package. Execute the following command at a terminal prompt to enable the mod_ssl module:

sudo a2enmod ssl

There is a default HTTPS configuration file in /etc/apache2/sites-available/default-ssl.conf. In order for Apache2 to provide HTTPS, a *certificate* and *key* file are also needed. The default HTTPS configuration will use a certificate and key generated by the ssl-cert package. They are good for testing, but the auto-generated certificate and key should be replaced by a certificate specific to the site or server. For information on generating a key and obtaining a certificate see Certificates.

To configure Apache2 for HTTPS, enter the following:

sudo a2ensite default-ssl

Note

The directories /etc/ssl/certs and /etc/ssl/private are the default locations. If you install the certificate and key in another directory make sure to change *SSLCertificateFile* and *SSLCertificateKeyFile* appropriately.

With Apache2 now configured for HTTPS, restart the service to enable the new settings:

sudo systemctl restart apache2.service

Note

Depending on how you obtained your certificate you may need to enter a passphrase when Apache2 starts.

You can access the secure server pages by typing https://your_hostname/url/ in your browser address bar.

Sharing Write Permission

For more than one user to be able to write to the same directory it will be necessary to grant write permission to a group they share in common. The following example grants shared write permission to /var/www/html to the group "webmasters".

sudo chgrp -R webmasters /var/www/html sudo chmod -R g=rwX /var/www/html/

These commands recursively set the group permission on all files and directories in /var/www/html to allow reading, writing and searching of directories. Many admins find this useful for allowing multiple users to edit files in a directory tree.

Warning

The apache2 daemon will run as the www-data user, which has a corresponding www-data group. These should not be granted write access to the document root, as this would mean that vulnerabilities in Apache or the applications it is serving would allow attackers to overwrite the served content.

References

- Apache2 Documentation contains in depth information on Apache2 configuration directives. Also, see the apache2-doc package for the official Apache2 docs.
- O'Reilly's Apache Cookbook is a good resource for accomplishing specific Apache2 configurations.
- For Ubuntu specific Apache2 questions, ask in the *#ubuntu-server* IRC channel on libera.chat.

Squid

Squid is a full-featured web proxy cache server application which provides proxy and cache services for Hyper Text Transport Protocol (HTTP), File Transfer Protocol (FTP), and other popular network protocols. Squid can implement caching and proxying of Secure Sockets Layer (SSL) requests and caching of Domain Name Server (DNS) lookups, and perform transparent caching. Squid also supports a wide variety of caching protocols, such as Internet Cache Protocol

(ICP), the Hyper Text Caching Protocol (HTCP), the Cache Array Routing Protocol (CARP), and the Web Cache Coordination Protocol (WCCP).

The Squid proxy cache server is an excellent solution to a variety of proxy and caching server needs, and scales from the branch office to enterprise level networks while providing extensive, granular access control mechanisms, and monitoring of critical parameters via the Simple Network Management Protocol (SNMP). When selecting a computer system for use as a dedicated Squid caching proxy server for many users ensure it is configured with a large amount of physical memory as Squid maintains an in-memory cache for increased performance.

Installation

At a terminal prompt, enter the following command to install the Squid server:

sudo apt install squid

Configuration

Squid is configured by editing the directives contained within the /etc/squid/squid.conf configuration file. The following examples illustrate some of the directives which may be modified to affect the behavior of the Squid server. For more in-depth configuration of Squid, see the References section.

Tip

Prior to editing the configuration file, you should make a copy of the original file and protect it from writing so you will have the original settings as a reference, and to re-use as necessary. Make this copy and protect it from writing using the following commands:

sudo cp /etc/squid/squid.conf /etc/squid/squid.conf.original sudo chmod a-w /etc/squid/squid.conf.original

• To set your Squid server to listen on TCP port 8888 instead of the default TCP port 3128, change the http_port directive as such:

http_port 8888

• Change the visible_hostname directive in order to give the Squid server a specific hostname. This hostname does not necessarily need to be the computer's hostname. In this example it is set to *weezie*

visible_hostname weezie

• The cache_dir option allows one to configure an on-disk cache, the default option is on-memory cache. The cache_dir directive takes the following arguments:

cache_dir <Type> <Directory-Name> <Fs-specific-data> [options]

In the config file you can find the default cache_dir directive commented out:

Uncomment and adjust the following to add a disk cache directory. #cache dir ufs /var/spool/squid 100 16 256

You can just use the default option but you can also customize your cache directory, basically changing the <Type> of this directory, it can be:

- ufs: the old well-known Squid storage format that has always been there.
- aufs: uses the same storage format as ufs, utilizing POSIX-threads to avoid blocking the main Squid process on disk-I/O. This was formerly known in Squid as async-io.
- diskd: uses the same storage format as ufs, utilizing a separate process to avoid blocking the main Squid process on disk-I/O.
- rock: is a database-style storage. All cached entries are stored in a "database" file, using fixed-size slots. A single entry occupies one or more slots.

If you want to use a different directory type please take a look at their different options.

• Using Squid's access control, you may configure use of Internet services proxied by Squid to be available only users with certain Internet Protocol (IP) addresses. For example, we will illustrate access by users of the 192.168.42.0/24 subnetwork only:

Add the following to the **bottom** of the ACL section of your /etc/squid.conf file:

acl fortytwo_network src 192.168.42.0/24

Then, add the following to the top of the http_access section of your /etc/squid.conf file:

http_access allow fortytwo_network

• Using the excellent access control features of Squid, you may configure use of Internet services proxied by Squid to be available only during normal business hours. For example, we'll illustrate access by employees of a business which is operating between 9:00AM and 5:00PM, Monday through Friday, and which uses the 10.1.42.0/24 subnetwork:

Add the following to the **bottom** of the ACL section of your /etc/squid.conf file:

acl biz_network src 10.1.42.0/24 acl biz_hours time M T W T F 9:00-17:00

Then, add the following to the top of the http_access section of your /etc/squid.conf file:

http_access allow biz_network biz_hours

Note

After making changes to the /etc/squid.conf file, save the file and restart the squid server application to effect the changes using the following command entered at a terminal prompt:

sudo systemctl restart squid.service

Note

If formerly a customized squid3 was used that set up the spool at /var/log/squid3 to be a mountpoint, but otherwise kept the default configuration the upgrade will fail. The upgrade tries to rename/move files as needed, but it can't do so for an active mountpoint. In that case please either adapt the mountpoint or the config in /etc/squid.conf so that they match.

The same applies if the **include** config statement was used to pull in more files from the old path at /etc/squid3/. In those cases you should move and adapt your configuration accordingly.

References

Squid Website

Ubuntu Wiki Squid page.

LAMP Applications

Overview

LAMP installations (Linux + Apache + MySQL + PHP/Perl/Python) are a popular setup for Ubuntu servers. There is a plethora of Open Source applications written using the LAMP application stack. Some popular LAMP applications are Wiki's, Content Management Systems, and Management Software such as phpMyAdmin.

One advantage of LAMP is the substantial flexibility for different database, web server, and scripting languages. Popular substitutes for MySQL include PostgreSQL and SQLite. Python, Perl, and Ruby are also frequently used instead of PHP. While Nginx, Cherokee and Lightpd can replace Apache.

The fastest way to get started is to install LAMP using tasksel. Tasksel is a Debian/Ubuntu tool that installs multiple related packages as a co-ordinated "task" onto your system. To install a LAMP server:

At a terminal prompt enter the following command:

sudo tasksel install lamp-server

After installing it you'll be able to install most *LAMP* applications in this way:

- Download an archive containing the application source files.
- Unpack the archive, usually in a directory accessible to a web server.
- Depending on where the source was extracted, configure a web server to serve the files.
- Configure the application to connect to the database.
- Run a script, or browse to a page of the application, to install the database needed by the application.
- Once the steps above, or similar steps, are completed you are ready to begin using the application.

A disadvantage of using this approach is that the application files are not placed in the file system in a standard way, which can cause confusion as to where the application is installed. Another larger disadvantage is updating the application. When a new version is released, the same process used to install the application is needed to apply updates.

Fortunately, a number of LAMP applications are already packaged for Ubuntu, and are available for installation in the same way as non-LAMP applications. Depending on the application some extra configuration and setup steps may be needed, however.

This section covers how to install some LAMP applications.

phpMyAdmin

phpMyAdmin is a LAMP application specifically written for administering MySQL servers. Written in PHP, and accessed through a web browser, phpMyAdmin provides a graphical interface for database administration tasks.

Installation

Before installing phpMyAdmin you will need access to a MySQL database either on the same host as that phpMyAdmin is installed on, or on a host accessible over the network. For more information see MySQL documentation. From a terminal prompt enter:

sudo apt install phpmyadmin

At the prompt choose which web server to be configured for phpMyAdmin. The rest of this section will use Apache2 for the web server.

In a browser go to http://servername/phpmyadmin, replacing *servername* with the server's actual hostname. At the login, page enter *root* for the *username*, or another MySQL user, if you have any setup, and enter the MySQL user's password.

Once logged in you can reset the root password if needed, create users, create/destroy databases and tables, etc.

Configuration

The configuration files for phpMyAdmin are located in /etc/phpmyadmin. The main configuration file is /etc/phpmyadmin/config.inc.php. This file contains configuration options that apply globally to phpMyAdmin.

To use phpMyAdmin to administer a MySQL database hosted on another server, adjust the following in /etc/phpmyadmin/config.inc.php:

\$cfg['Servers'][\$i]['host'] = 'db_server';

Note

Replace db_server with the actual remote database server name or IP address. Also, be sure that the phpMyAdmin host has permissions to access the remote database.

Once configured, log out of phpMyAdmin and back in, and you should be accessing the new server.

The config.header.inc.php and config.footer.inc.php files in /etc/phpmyadmin directory are used to add a HTML header and footer to phpMyAdmin.

Another important configuration file is /etc/phpmyadmin/apache.conf, this file is symlinked to /etc/apache2/confavailable/phpmyadmin.conf, and, once enabled, is used to configure Apache2 to serve the phpMyAdmin site. The file contains directives for loading PHP, directory permissions, etc. From a terminal type:

```
sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/conf-available/phpmyadmin.conf
sudo a2enconf phpmyadmin.conf
sudo systemctl reload apache2.service
```

For more information on configuring Apache2 see this documentation.

References

- The phpMyAdmin documentation comes installed with the package and can be accessed from the *phpMyAdmin Documentation* link (a question mark with a box around it) under the phpMyAdmin logo. The official docs can also be access on the phpMyAdmin site.
- Another resource is the phpMyAdmin Ubuntu Wiki page.

WordPress

Wordpress is a blog tool, publishing platform and CMS implemented in PHP and licensed under the GNU GPLv2.

Installation

To install WordPress, run the following comand in the command prompt:

sudo apt install wordpress

You should also install apache2 web server and mysql server. For installing apache2 web server, please refer to MySQL documentation.

Configuration

For configuring your first WordPress application, configure an apache site. Open /etc/apache2/sites-available/wordpress.conf and write the following lines:

```
Alias /blog /usr/share/wordpress
<Directory /usr/share/wordpress>
    Options FollowSymLinks
    AllowOverride Limit Options FileInfo
    DirectoryIndex index.php
    Order allow,deny
    Allow from all
</Directory>
<Directory /usr/share/wordpress/wp-content>
    Options FollowSymLinks
    Order allow,deny
    Allow from all
</Directory>
```

Enable this new WordPress site

sudo a2ensite wordpress

Once you configure the apache2 web server and make it ready for your WordPress application, you should restart it. You can run the following command to restart the apache2 web server:

sudo systemctl reload apache2.service

To facilitate multiple WordPress installations, the name of this configuration file is based on the Host header of the HTTP request. This means that you can have a configuration per VirtualHost by simply matching the host-name portion of this configuration with your Apache Virtual Host. e.g. /etc/wordpress/config-10.211.55.50.php, /etc/wordpress/config-hostalias1.php, etc. These instructions assume you can access Apache via the localhost host-name (perhaps by using an ssh tunnel) if not, replace /etc/wordpress/config-localhost.php with /etc/wordpress/config-NAME_OF_YOUR_VIRTUAL_HOST.php.

Once the configuration file is written, it is up to you to choose a convention for username and password to mysql for each WordPress database instance. This documentation shows only one, localhost, example.

Now configure WordPress to use a mysql database. Open /etc/wordpress/config-localhost.php file and write the following lines:

```
<?php
define('DB_NAME', 'wordpress');
define('DB_USER', 'wordpress');
define('DB_PASSWORD', 'yourpasswordhere');
define('DB_HOST', 'localhost');
define('WP_CONTENT_DIR', '/usr/share/wordpress/wp-content');
?>
```

Now create this mysql database. Open a temporary file with mysql commands ${\tt wordpress.sql}$ and write the following lines:

```
CREATE DATABASE wordpress;
CREATE USER 'wordpress'@'localhost'
IDENTIFIED BY 'yourpasswordhere';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER
ON wordpress.*
T0 wordpress@localhost;
FLUSH PRIVILEGES;
```

```
Execute these commands.
```

cat wordpress.sql | sudo mysql --defaults-extra-file=/etc/mysql/debian.cnf

Your new WordPress can now be configured by visiting http://localhost/blog/wp-admin/install.php. (Or http://NAME_OF_YOUR_VIRTUAL_HOST/blog/wp-admin/install.php if your server has no GUI and you are completing WordPress configuration via a web browser running on another computer.) Fill out the Site Title, username, password, and E-mail and click Install WordPress.

Note the generated password (if applicable) and click the login password. Your WordPress is now ready for use.

References

- WordPress.org Codex
- Ubuntu Wiki WordPress

PHP - Scripting Language

PHP is a general-purpose scripting language suited for Web development. PHP scripts can be embedded into HTML. This section explains how to install and configure PHP in an Ubuntu System with Apache2 and MySQL.

This section assumes you have installed and configured Apache2 Web Server and MySQL Database Server. You can refer to the Apache2 and MySQL sections in this document to install and configure Apache2 and MySQL respectively.

Installation

PHP is available in Ubuntu Linux. Unlike Python, which is installed in the base system, PHP must be added.

To install PHP and the Apache PHP module you can enter the following command at a terminal prompt:

sudo apt install php libapache2-mod-php

You can run PHP scripts at a terminal prompt. To run PHP scripts at a terminal prompt you should install the php-cli package. To install php-cli you can enter the following command:

sudo apt install php-cli

You can also execute PHP scripts without installing the Apache PHP module. To accomplish this, you should install the php-cgi package via this command:

sudo apt install php-cgi

To use MySQL with PHP you should install the php-mysql package, like so:

sudo apt install php-mysql

Similarly, to use PostgreSQL with PHP you should install the php-pgsql package:

sudo apt install php-pgsql

Configuration

If you have installed the libapache2-mod-php or php-cgi packages, you can run PHP scripts from your web browser. If you have installed the php-cli package, you can run PHP scripts at a terminal prompt.

By default, when libapache2-mod-php is installed, the Apache 2 Web server is configured to run PHP scripts using this module. Please verify if the files /etc/apache2/mods-enabled/php8.*.load exist. If they do not exist, you can enable the module using the a2enmod command.

Once you have installed the PHP related packages and enabled the Apache PHP module, you should restart the Apache2 Web server to run PHP scripts, by running the following command:

sudo systemctl restart apache2.service

Testing

To verify your installation, you can run the following PHP phpinfo script:

```
<?php
phpinfo();
?>
```

You can save the content in a file phpinfo.php and place it under the DocumentRoot directory of the Apache2 Web server. Pointing your browser to http://hostname/phpinfo.php will display the values of various PHP configuration parameters.

References

- For more in depth information see the php.net documentation.
- There are a plethora of books on PHP 7 and 8. A good book from O'Reilly is Learning PHP, which includes an exploration of PHP 7's enhancements to the language.
- Also, see the Apache MySQL PHP Ubuntu Wiki page for more information.

Ruby on Rails

Ruby on Rails is an open source web framework for developing database backed web applications. It is optimized for sustainable productivity of the programmer since it lets the programmer to write code by favouring convention over configuration.

Installation

Before installing Rails you should install Apache (or a preferred web server) and a database service such as MySQL. To install the Apache package, please refer to MySQL documentation for example.

Once you have a web server (e.g. Apache) and a database service (e.g. MySQL) installed and configured, you are ready to install Ruby on Rails package.

To install the Ruby base packages and Ruby on Rails, you can enter the following command in the terminal prompt:

sudo apt install rails

Configuration

Web Server

Modify the /etc/apache2/sites-available/000-default.conf configuration file to setup your domains.

The first thing to change is the *DocumentRoot* directive:

DocumentRoot /path/to/rails/application/public

Next, change the <Directory "/path/to/rails/application/public"> directive:

```
<Directory "/path/to/rails/application/public">
    Options Indexes FollowSymLinks MultiViews ExecCGI
    AllowOverride All
    Order allow,deny
    allow from all
    AddHandler cgi-script .cgi
```

</Directory>

You should also enable the mod_rewrite module for Apache. To enable mod_rewrite module, please enter the following command in a terminal prompt:

```
sudo a2enmod rewrite
```

Finally you will need to change the ownership of the /path/to/rails/application/public and /path/to/rails/application/tmp directories to the user used to run the Apache process:

sudo chown -R www-data:www-data /path/to/rails/application/public sudo chown -R www-data:www-data /path/to/rails/application/tmp

If you need to compile your application assets run the following command in your application directory:

RAILS_ENV=production rake assets:precompile

Database

With your database service in place you need to make sure your app database configuration is also correct. For example, if you are using MySQL your config/database.yml should look like this:
Mysql
production:
 adapter: mysql2
 username: user
 password: password
 host: 127.0.0.1
 database: app

To finally create your application database and apply its migrations you can run the following commands from your app directory:

RAILS_ENV=production rake db:create RAILS_ENV=production rake db:migrate

That's it! Now you have your Server ready for your Ruby on Rails application. You can daemonize your application as you want.

References

- See the Ruby on Rails website for more information.
- Also Agile Development with Rails is a great resource.

OpenLDAP Introduction

The Lightweight Directory Access Protocol, or LDAP, is a protocol for querying and modifying a X.500-based directory service running over TCP/IP. The current LDAP version is LDAPv3, as defined in RFC4510, and the implementation used in Ubuntu is OpenLDAP.

The LDAP protocol accesses directories. A common mistake is to call a directory an *LDAP directory*, or *LDAP database*, but it's really so common, and we all know what we are talking about, that it's ok. Here are some key concepts and terms:

- A directory is a tree of data *entries* that is hierarchical in nature and is called the Directory Information Tree (DIT).
- An entry consists of a set of *attributes*.
- An attribute has a key (a name/description) and one or more values.
- Every attribute must be defined in at least one *objectClass*.
- Attributes and object classes are defined in *schemas* (an object class is actually considered as a special kind of attribute).
- Each entry has a unique identifier: its *Distinguished Name* (DN or dn). This, in turn, consists of a *Relative Distinguished Name* (RDN) followed by the parent entry's DN.
- The entry's DN is not an attribute. It is not considered part of the entry itself.

Note

The terms *object*, *container*, and *node* have certain connotations but they all essentially mean the same thing as *entry*, the technically correct term.

For example, below we have a single entry consisting of 11 attributes where the following is true:

- DN is "cn=John Doe,dc=example,dc=com"
- RDN is "cn=John Doe"
- parent DN is "dc=example,dc=com"

dn: cn=John Doe,dc=example,dc=com cn: John Doe givenName: John sn: Doe telephoneNumber: +1 888 555 6789 telephoneNumber: +1 888 555 1232 mail: john@example.com manager: cn=Larry Smith,dc=example,dc=com objectClass: inetOrgPerson objectClass: organizationalPerson
objectClass: person
objectClass: top

The above entry is in *LDIF* format (LDAP Data Interchange Format). Any information that you feed into your DIT must also be in such a format. It is defined in RFC2849.

Such a directory accessed via LDAP is good for anything that involves a large number of access requests to a mostlyread, attribute-based (name:value) backend, and that can benefit from a hierarchical structure. Examples include an address book, company directory, a list of email addresses, and a mail server's configuration.

References

- The OpenLDAP administrators guide
- RFC 4515: LDAP string representation of search filters
- Zytrax's LDAP for Rocket Scientists; a less pedantic but comprehensive treatment of LDAP

Older references that might still be useful:

- O'Reilly's LDAP System Administration (textbook; 2003)
- Packt's Mastering OpenLDAP (textbook; 2007)

LDAP Installation

The installation of slapd will create a minimal working configuration with a top level entry, and an administrator's DN. In particular, it will create a database instance that you can use to store your data. However, the suffix (or base DN) of this instance will be determined from the domain name of the host. If you want something different, you can change it right after the installation when you still don't have any useful data.

Note

This guide will use a database suffix of dc=example, dc=com.

Proceed with the install of the server and the main command line utilities:

sudo apt install slapd ldap-utils

If you want to change your DIT suffix, now would be a good time, because changing it discards your existing one. To change the suffix, run the following command:

sudo dpkg-reconfigure slapd

To switch your DIT suffix to dc=example, dc=com, for example, so you can follow this guide more closely, answer example.com when asked about the DNS domain name.

Throughout this guide we will issue many commands with the LDAP utilities. To save some typing, we can configure the OpenLDAP libraries with certain defaults in /etc/ldap/ldap.conf:

```
BASE dc=example,dc=com
URI ldap://ldap01.example.com
```

Note

Adjust for your server name and directory suffix

The Default DIT

The packaging of *slapd* is designed to be configured within the service itself by dedicating a separate DIT for that purpose. This allows one to dynamically configure *slapd* without the need to restart the service or edit config files. This configuration database consists of a collection of text-based LDIF files located under /etc/ldap/slapd.d, but these should never be edited directly. This way of working is known by several names: the *slapd-config* method, the RTC method (Real Time Configuration), or the *cn=config* method. You can still use the traditional flat-file method (*slapd.conf*) but it's not going to be covered in this guide.

Right after installation, you will get two databases, or suffixes: one for your data, based on your host's domain (dc=example, dc=com), and one for your configuration, with its root at cn=config. To change the data on each we need different credentials and access methods:

• dc=example, dc=com: the administrative user for this suffix is cn=admin, dc=example, dc=com and its password is the one selected during the installation of the slapd package

- cn=config: the configuration slapd itself is stored under this suffix. Changes to it can be made by the special DN gidNumber=0+uidNumber=0, cn=peercred, cn=external, cn=auth. This is how the local system's root user (uid=0/gid=0) is seen by the directory when using SASL EXTERNAL authentication through the ldapi:/// transport via the /run/slapd/ldapi unix socket. Essentially what this means is that only the local root user can update the cn=config database. More details later.
- This is what the slapd-config DIT looks like via the LDAP protocol (listing only the *dns*):

 $\$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=config dn dn: cn=config

- dn: cn=module{0},cn=config
- dn: cn=schema,cn=config
- dn: cn={0}core,cn=schema,cn=config
- dn: cn={1}cosine,cn=schema,cn=config
- dn: cn={2}nis,cn=schema,cn=config
- dn: cn={3}inetorgperson,cn=schema,cn=config
- dn: olcDatabase={-1}frontend,cn=config
- dn: olcDatabase={0}config,cn=config
- dn: olcDatabase={1}mdb,cn=config

Explanation of entries:

- cn=config: global settings
- cn=module{0}, cn=config: a dynamically loaded module
- cn=schema,cn=config: contains hard-coded system-level schema
- $cn = \{0\}$ core, cn = schema, cn = config: the hard-coded core schema
- $cn = \{1\} cosine, cn = schema, cn = config:$ the cosine schema
- $cn = \{2\}$ nis, cn = schema, cn = config: the nis schema
- $cn = \{3\}$ inetorgperson, cn = schema, cn = config: the inetorgperson schema
- olcDatabase={-1}frontend, cn=config: frontend database, default settings for other databases
- olcDatabase={0}config.cn=config: slapd configuration database (cn=config)
- $olcDatabase = \{1\} mdb, cn = config:$ your database instance (dc = example, dc = com)
- This is what the dc=example, dc=com DIT looks like:

\$ ldapsearch -x -LLL -H ldap:/// -b dc=example,dc=com dn

dn: dc=example,dc=com

dn: cn=admin,dc=example,dc=com

Explanation of entries:

- dc=example,dc=com: base of the DIT
- cn = admin, dc = example, dc = com: administrator (rootDN) for this DIT (set up during package install)

Notice how we used two different authentication mechanisms:

- -x: this is called a *simple bind*, and is essentially a plain text authentication. Since no *binddn* was provided (via -D), this became an *anonymous* bind. Without -x, the default is to use a SASL bind.
- -Y EXTERNAL: this is using a SASL bind (no -x was provided), and further specifying the EXTERNAL type. Together with -H ldapi:///, this uses a local unix socket connection

In both cases we only got the results that the server ACLs allowed us to see, based on who we are. A very handy tool to verify the authentication is *ldapwhoami*:

\$ ldapwhoami -x
anonymous

```
$ ldapwhoami -x -D cn=admin,dc=example,dc=com -W
Enter LDAP Password:
dn:cn=admin,dc=example,dc=com
```

When you use simple bind (-x) and specify a *binddn* with -D as your authentication dn, the server will look for a *userPassword* attribute in that entry, and use that to verify the credentials. In this particular case above, we used the database *rootDN* entry, i.e., the actual administrator, and that is a special case whose password is set in the configuration when the package is installed.

Note

A simple bind without some sort of transport security mechanism is **clear text**, meaning the credentials are transmitted in the clear. You should add TLS support to your OpenLDAP server as soon as possible.

Here are the SASL EXTERNAL examples:

```
$ ldapwhoami -Y EXTERNAL -H ldapi:/// -Q
dn:gidNumber=1000+uidNumber=1000,cn=peercred,cn=external,cn=auth
```

```
$ sudo ldapwhoami -Y EXTERNAL -H ldapi:/// -Q
dn:gidNumber=0+uidNumber=0, cn=peercred, cn=external, cn=auth
```

When using SASL EXTERNAL via the ldapi:/// transport, the *binddn* becomes a combination of the *uid* and *gid* of the connecting user, followed by the suffix cn=peercred, cn=external, cn=auth. The server ACLs know about this, and grant the local root user complete write access to cn=config via the SASL mechanism.

Populating the Directory

Let's introduce some content to our directory. We will add the following:

- a node called *People* (to store users)
- a node called *Groups* (to store groups)
- a group called *miners*
- a user called *john*

Create the following LDIF file and call it add_content.ldif:

```
dn: ou=People,dc=example,dc=com
objectClass: organizationalUnit
ou: People
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
ou: Groups
dn: cn=miners,ou=Groups,dc=example,dc=com
objectClass: posixGroup
cn: miners
gidNumber: 5000
dn: uid=john,ou=People,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: john
sn: Doe
givenName: John
cn: John Doe
displayName: John Doe
uidNumber: 10000
gidNumber: 5000
```

userPassword: {CRYPT}x
gecos: John Doe
loginShell: /bin/bash
homeDirectory: /home/john

Note

It's important that uid and gid values in your directory do not collide with local values. You can use high number ranges, such as starting at 5000 or even higher.

Add the content:

\$ ldapadd -x -D cn=admin,dc=example,dc=com -W -f add_content.ldif Enter LDAP Password: ******* adding new entry "ou=People,dc=example,dc=com"

adding new entry "ou=Groups,dc=example,dc=com"

adding new entry "cn=miners,ou=Groups,dc=example,dc=com"

adding new entry "uid=john,ou=People,dc=example,dc=com"

We can check that the information has been correctly added with the *ldapsearch* utility. For example, let's search for the *john* entry, and request the *cn* and *gidnumber* attributes:

\$ ldapsearch -x -LLL -b dc=example,dc=com '(uid=john)' cn gidNumber dn: uid=john,ou=People,dc=example,dc=com cn: John Doe gidNumber: 5000

Here we used an LDAP "filter": (*uid=john*). LDAP filters are very flexible and can become complex. For example, to list the group names of which *john* is a member, we could use the filter:

(&(objectClass=posixGroup)(memberUid=john))

That is a logical AND between two attributes. Filters are very important in LDAP and mastering their syntax will help a long way. They are used for simple queries like this, but can also select what content is to be replicated to a secondary server, or even in complex ACLs. The full specification is defined in RFC 4515.

Notice we set the *userPassword* field for the *john* entry to the cryptic value {CRYPT}x. This essentially is an invalid password, because no hashing will produce just x. It's a common pattern when adding a user entry without a default password. To change the password to something valid, you can now use *ldappasswd*:

\$ ldappasswd -x -D cn=admin,dc=example,dc=com -W -S uid=john,ou=people,dc=example,dc=com New password: Re-enter new password: Enter LDAP Password:

Note

Remember that simple binds are insecure and you should add TLS support to your server!

Changing the Configuration

The *slapd-config* DIT can also be queried and modified. Here are some common operations.

Adding an index

Use *ldapmodify* to add an "Index" to your $\{1\}mdb, cn=config$ database definition (for dc=example, dc=com). Create a file, call it uid_index.ldif, with the following contents:

dn: olcDatabase={1}mdb,cn=config
add: olcDbIndex
olcDbIndex: mail eq,sub

Then issue the command:

\$ sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f uid_index.ldif

modifying entry "olcDatabase={1}mdb,cn=config"

You can confirm the change in this way:

```
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={1}mdb)' olcDbIndex
```

```
dn: olcDatabase={1}mdb,cn=config
olcDbIndex: objectClass eq
olcDbIndex: cn,uid eq
olcDbIndex: uidNumber,gidNumber eq
olcDbIndex: member,memberUid eq
olcDbIndex: mail eq,sub
```

Change the rootDN password:

First, run *slappasswd* to get the hash for the new password you want:

\$ slappasswd
New password:
Re-enter new password:
{SSHA}VKrYMxlSKh0NGRpC6rnASKNmXG2xHXFo

Now prepare a changerootpw.ldif file with this content:

```
dn: olcDatabase={1}mdb,cn=config
changetype: modify
replace: olcRootPW
olcRootPW: {SSHA}VKrYMxlSKh0NGRpC6rnASKNmXG2xHXFo
```

Finally, run the *ldapmodify* command:

```
$ sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f changerootpw.ldif
modifying entry "olcDatabase={1}mdb,cn=config"
```

We still have the actual cn=admin, dc=example, dc=com dn in the dc=example, dc=com database, so let's change it too. Since this is a regular entry in this database suffix, we can use ldappasswd:

```
$ ldappasswd -x -D cn=admin,dc=example,dc=com -W -S
New password:
Re-enter new password:
Enter LDAP Password: <-- current password, about to be changed</pre>
```

Adding a schema

Schemas can only be added to cn=config if they are in LDIF format. If not, they will first have to be converted. You can find unconverted schemas in addition to converted ones in the /etc/ldap/schema directory.

Note

It is not trivial to remove a schema from the slapd-config database. Practice adding schemas on a test system.

In the following example we'll add the password policy (*ppolicy*) schema. This schema exists in both converted (*.ldif*) and native (*.schema*) formats, so we don't have to convert it and can use *ldapadd* directly:

\$ sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/ppolicy.ldif adding new entry "cn=ppolicy,cn=schema,cn=config"

If the schema you want to add does not exist in LDIF format, a nice conversion tool that can be used is provided in the schema2ldif package.

Logging

Activity logging for slapd is very useful when implementing an OpenLDAP-based solution yet it must be manually enabled after software installation. Otherwise, only rudimentary messages will appear in the logs. Logging, like any other such configuration, is enabled via the *slapd-config* database.

OpenLDAP comes with multiple logging levels with each one containing the lower one (additive). A good level to try is *stats*. The slapd-config man page has more to say on the different subsystems.

Create the file logging.ldif with the following contents:

dn: cn=config
changetype: modify

replace: olcLogLevel
olcLogLevel: stats

Implement the change:

sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f logging.ldif

This will produce a significant amount of logging and you will want to throttle back to a less verbose level once your system is in production. While in this verbose mode your host's syslog engine (rsyslog) may have a hard time keeping up and may drop messages:

rsyslogd-2177: imuxsock lost 228 messages from pid 2547 due to rate-limiting

You may consider a change to rsyslog's configuration. In /etc/rsyslog.conf, put:

```
# Disable rate limiting
# (default is 200 messages in 5 seconds; below we make the 5 become 0)
$SystemLogRateLimitInterval 0
```

And then restart the rsyslog daemon:

sudo systemctl restart syslog.service

LDAP Access Control

The management of what type of access (read, write, etc) users should be granted to resources is known as *access* control. The configuration directives involved are called *access control lists* or ACL.

When we installed the slapd package various ACL were set up automatically. We will look at a few important consequences of those defaults and, in so doing, we'll get an idea of how ACLs work and how they're configured.

To get the effective ACL for an LDAP query we need to look at the ACL entries of the database being queried as well as those of the special frontend database instance. The ACLs belonging to the frontend database are always appended to the database-specific ACLs, and the first match wins.

The following commands will give, respectively, the ACLs of the mdb database ("dc=example,dc=com") and those of the frontend database:

```
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={1}mdb)' olcAccess
```

```
dn: olcDatabase={1}mdb,cn=config
olcAccess: {0}to attrs=userPassword by self write by anonymous auth by * none
olcAccess: {1}to attrs=shadowLastChange by self write by * read
olcAccess: {2}to * by * read
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={-1}frontend)' olcAccess
dn: olcDatabase={-1}frontend,cn=config
olcAccess: {0}to * by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external
,cn=auth manage by * break
olcAccess: {1}to dn.exact="" by * read
```

Note

The rootDN always has full rights to its database and does not need to be included in any ACL.

The very first two ACLs are crucial:

```
olcAccess: {0}to attrs=userPassword by self write by anonymous auth by * none
olcAccess: {1}to attrs=shadowLastChange by self write by * read
```

This can be represented differently for easier reading:

olcAccess: {2}to dn.base="cn=Subschema" by * read

```
to attrs=userPassword
    by self write
    by anonymous auth
    by * none
```

to attrs=shadowLastChange by self write bv * read

These ACLs enforce the following:

- Anonymous 'auth' access is provided to the *userPassword* attribute so that users can authenticate, or *bind*. Perhaps counter-intuitively, 'by anonymous auth' is needed even when anonymous access to the DIT is unwanted, otherwise this would be a chicken and egg problem: before authentication, all users are anonymous.
- The by self write ACL grants write access to the userPassword attribute to users who authenticated as the dn where the attribute lives. In other words, users can update the userPassword attribute of their own entries.
- The *userPassword* attribute is otherwise unaccessible by all other users, with the exception of the rootDN, who always has access and doesn't need to be mentioned explicitly.
- In order for users to change their own password, using passwd or other utilities, the user's own *shadowLastChange* attribute needs to be writable. All other directory users get to read this attribute's contents.

This DIT can be searched anonymously because of 'to * by * read' in this ACL, which grants read access to everything else, by anyone (including anonymous):

to *

by * read

If this is unwanted then you need to change the ACLs. To force authentication during a bind request you can alternatively (or in combination with the modified ACL) use the 'olcRequire: authc' directive.

As previously mentioned, there is no administrative account ("rootDN") created for the slapd-config database. There is, however, a SASL identity that is granted full access to it. It represents the localhost's superuser (root/sudo). Here it is:

dn.exact=gidNumber=0+uidNumber=0, cn=peercred, cn=external, cn=auth

The following command will display the ACLs of the slapd-config database:

```
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcDatabase={0}config)' olcAccess
```

Since this is a SASL identity we need to use a SASL *mechanism* when invoking the LDAP utility in question and we have seen it plenty of times in this guide. It is the EXTERNAL mechanism. See the previous command for an example. Note that:

- You must use *sudo* to become the root identity in order for the ACL to match.
- The EXTERNAL mechanism works via IPC (UNIX domain sockets). This means you must use the ldapi URI format.

A succinct way to get all the ACLs is like this:

```
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b \
cn=config '(olcAccess=*)' olcAccess olcSuffix
```

References

- See the man page for slapd.access.
- Access Control topic in the OpenLDAP Administrator's guide

OpenLDAP Replication

The LDAP service becomes increasingly important as more networked systems begin to depend on it. In such an environment, it is standard practice to build redundancy (high availability) into LDAP to prevent havoc should the LDAP server become unresponsive. This is done through *LDAP replication*.

Replication is achieved via the *Syncrepl* engine. This allows changes to be synchronized using a *Consumer - Provider* model. A detailed description this replication mechanism can be found in the OpenLDAP Administrator's Guide and in its defining RFC 4533.

There are two ways to use this replication:

- standard replication: changed entries are sent to the consumer in their entirety. For example, if the userPassword attribute of the uid=john,ou=people,dc=example,dc=com entry changed, then the whole entry is sent to the consumer
- delta replication: only the actual change is sent, instead of the whole entry

The delta replication sends less data over the network, but is more complex to setup. We will show both in this guide.

Important

You must have TLS enabled already. Please consult the LDAP with TLS guide

Provider Configuration - replication user

Both replication strategies will need a replication user and updates to the ACLs and limits regarding this user.

To create the replication user, save the following contents to a file called replicator.ldif:

dn: cn=replicator,dc=example,dc=com
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: replicator
description: Replication user
userPassword: {CRYPT}x

Then add it with *ldapadd*:

\$ ldapadd -x -ZZ -D cn=admin,dc=example,dc=com -W -f replicator.ldif Enter LDAP Password: adding new entry "cn=replicator,dc=example,dc=com"

Now set a password for it with *ldappasswd*:

\$ ldappasswd -x -ZZ -D cn=admin,dc=example,dc=com -W -S cn=replicator,dc=example,dc=com New password: Re-enter new password: Enter LDAP Password:

The next step is to give this replication user the correct privileges:

- read access to the content that we want replicated
- no search limits on this content

For that we need to update the ACLs on the provider. Since ordering matters, first check what the existing ACLs look like on the dc=example, dc=com tree:

```
$ sudo ldapsearch -Q -Y EXTERNAL -H ldapi:/// -LLL -b cn=config '(olcSuffix=dc=example,dc=com)' olcAccess
dn: olcDatabase={1}mdb,cn=config
olcAccess: {0}to attrs=userPassword by self write by anonymous auth by * none
olcAccess: {1}to attrs=shadowLastChange by self write by * read
olcAccess: {2}to * by * read
```

What we need is to insert a new rule before the first one, and also adjust the limits for the replicator user. Prepare the replicator-acl-limits.ldif file with this content:

```
dn: olcDatabase={1}mdb,cn=config
changetype: modify
add: olcAccess
olcAccess
olcAccess: {0}to *
    by dn.exact="cn=replicator,dc=example,dc=com" read
    by * break
-
add: olcLimits
olcLimits
olcLimits: dn.exact="cn=replicator,dc=example,dc=com"
    time.soft=unlimited time.hard=unlimited
    size.soft=unlimited size.hard=unlimited
And add it to the server:
```

```
$ sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f replicator-acl-limits.ldif
modifying entry "olcDatabase={1}mdb,cn=config"
```

Provider Configuration - standard replication

The remaining configuration for the provider using standard replication is to add the *syncprov* overlay on top of the dc=example, dc=com database.o

Create a file called provider_simple_sync.ldif with this content:

Add indexes to the frontend db. dn: olcDatabase={1}mdb,cn=config changetype: modify add: olcDbIndex olcDbIndex: entryCSN eq -

add: olcDbIndex
olcDbIndex: entryUUID eq

#Load the syncprov module. dn: cn=module{0},cn=config changetype: modify add: olcModuleLoad olcModuleLoad: syncprov

syncrepl Provider for primary db
dn: olcOverlay=syncprov,olcDatabase={1}mdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
olcSpCheckpoint: 100 10
olcSpSessionLog: 100

Customization warning

The LDIF above has some parameters that you should review before deploying in production on your directory. In particular:

• *olcSpCheckpoint*, *olcSpSessionLog*: please see the slapo-syncprov(5) manpage. In general, *olcSpSessionLog* should be equal to, or preferably larger, than the number of entries in your directory. Also see ITS #8125 for details on an existing bug.

Add the new content:

sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f provider_simple_sync.ldif

The Provider is now configured.

Consumer Configuration - standard replication

Install the software by going through enable TLS.

Create an LDIF file with the following contents and name it consumer_simple_sync.ldif:

```
dn: cn=module{0},cn=config
changetype: modify
add: olcModuleLoad
olcModuleLoad: syncprov
dn: olcDatabase={1}mdb,cn=config
changetype: modify
add: olcDbIndex
olcDbIndex: entryUUID eq
-
add: olcSyncrepl
olcSyncrepl: rid=0
provider=ldap://ldap01.example.com
bindmethod=simple
binddn="cn=replicator,dc=example,dc=com" credentials=<secret>
searchbase="dc=example,dc=com"
schemachecking=on
```

```
type=refreshAndPersist retry="60 +"
starttls=critical tls_reqcert=demand
```

```
add: olcUpdateRef
```

olcUpdateRef: ldap://ldap01.example.com

Ensure the following attributes have the correct values:

- *provider* (Provider server's hostname ldap01.example.com in this example or IP address). It must match what is presented in the provider's SSL certificate.
- *binddn* (the bind DN for the *replicator* user)
- credentials (the password you selected for the replicator user)
- searchbase (the database suffix you're using, i.e., content that is to be replicated)
- *olcUpdateRef* (Provider server's hostname or IP address, given to clients if they try to write to this consumer)
- rid (Replica ID, an unique 3-digit that identifies the replica. Each consumer should have at least one rid)

Note

Note that a successful encrypted connection via *START_TLS* is being enforced in this configuration, to avoid sending the credentials in the clear across the network. See LDAP with TLS for details on how to setup OpenLDAP with trusted SSL certificates.

Add the new configuration:

sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f consumer_simple_sync.ldif

You're done. The dc=example, dc=com tree should now be synchronizing.

Provider Configuration - delta replication

The remaining provider configuration for delta replication is:

- create a new database called *accesslog*
- add the syncprov overlay on top of the accesslog and dc=example, dc=com databases
- add the *accesslog* overlay on top of the dc=example, dc=com database

Add syncprov and accesslog overlays and DBs

Create an LDIF file with the following contents and name it provider_sync.ldif:

Add indexes to the frontend db. dn: olcDatabase={1}mdb,cn=config changetype: modify add: olcDbIndex olcDbIndex: entryCSN eq add: olcDbIndex olcDbIndex: entryUUID eq #Load the syncprov and accesslog modules. dn: cn=module{0},cn=config changetype: modify add: olcModuleLoad olcModuleLoad: syncprov add: olcModuleLoad olcModuleLoad: accesslog # Accesslog database definitions

dn: olcDatabase={2}mdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcMdbConfig
olcDatabase: {2}mdb
olcDbDirectory: /var/lib/ldap/accesslog
olcSuffix: cn=accesslog

olcRootDN: cn=admin,dc=example,dc=com olcDbIndex: default eq olcDbIndex: entryCSN,objectClass,reqEnd,reqResult,reqStart olcAccess: {0}to * by dn.exact="cn=replicator,dc=example,dc=com" read by * break olcLimits: dn.exact="cn=replicator,dc=example,dc=com" time.soft=unlimited time.hard=unlimited size.soft=unlimited size.hard=unlimited

Accesslog db syncprov. dn: olcOverlay=syncprov,olcDatabase={2}mdb,cn=config changetype: add objectClass: olcOverlayConfig objectClass: olcSyncProvConfig olcOverlay: syncprov olcSpNoPresent: TRUE olcSpReloadHint: TRUE

syncrepl Provider for primary db
dn: olcOverlay=syncprov,olcDatabase={1}mdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
olcSpCheckpoint: 100 10
olcSpSessionLog: 100

accesslog overlay definitions for primary db dn: olcOverlay=accesslog,olcDatabase={1}mdb,cn=config objectClass: olcOverlayConfig objectClass: olcAccessLogConfig olcOverlay: accesslog olcAccessLogDB: cn=accesslog olcAccessLogOps: writes olcAccessLogSuccess: TRUE # scan the accesslog DB every day, and purge entries older than 7 days olcAccessLogPurge: 07+00:00 01+00:00

Customization warning

The LDIF above has some parameters that you should review before deploying in production on your directory. In particular:

- *olcSpCheckpoint*, *olcSpSessionLog*: please see the slapo-syncprov(5) manpage. In general, *olcSpSessionLog* should be equal to, or preferably larger, than the number of entries in your directory. Also see ITS #8125 for details on an existing bug.
- *olcAccessLogPurge*: check the slapo-accesslog(5) manpage

Create a directory:

sudo -u openldap mkdir /var/lib/ldap/accesslog

Add the new content:

sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f provider_sync.ldif

The Provider is now configured.

Consumer Configuration

Install the software by going through enable TLS.

Create an LDIF file with the following contents and name it consumer_sync.ldif:

dn: cn=module{0}, cn=config
changetype: modify
add: olcModuleLoad
olcModuleLoad: syncprov

```
dn: olcDatabase={1}mdb,cn=config
changetype: modify
add: olcDbIndex
olcDbIndex: entryUUID eq
add: olcSyncrepl
olcSyncrepl: rid=0
  provider=ldap://ldap01.example.com
  bindmethod=simple
  binddn="cn=replicator,dc=example,dc=com" credentials=<secret>
  searchbase="dc=example,dc=com"
  logbase="cn=accesslog"
  logfilter="(&(objectClass=auditWriteObject)(reqResult=0))"
  schemachecking=on
  type=refreshAndPersist retry="60 +"
  syncdata=accesslog
  starttls=critical tls_reqcert=demand
```

add: olcUpdateRef
olcUpdateRef: ldap://ldap01.example.com

Ensure the following attributes have the correct values:

- *provider* (Provider server's hostname ldap01.example.com in this example or IP address). It must match what is presented in the provider's SSL certificate.
- *binddn* (the bind DN for the *replicator* user)
- credentials (the password you selected for the replicator user)
- searchbase (the database suffix you're using, i.e., content that is to be replicated)
- *olcUpdateRef* (Provider server's hostname or IP address, given to clients if they try to write to this consumer)
- rid (Replica ID, an unique 3-digit that identifies the replica. Each consumer should have at least one rid)
 Note

Note that a successful encrypted connection via *START_TLS* is being enforced in this configuration, to avoid sending the credentials in the clear across the network. See LDAP with TLS for details on how to setup OpenLDAP with trusted SSL certificates.

Add the new configuration:

sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f consumer_sync.ldif

You're done. The dc=example, dc=com tree should now be synchronizing.

Testing

Once replication starts, you can monitor it by running

\$ ldapsearch -z1 -LLL -x -s base -b dc=example,dc=com contextCSN
dn: dc=example,dc=com
contextCSN: 20200423222317.722667Z#000000#000#0000#0000000

on both the provider and the consumer. Once the *contextCSN* value for both match, both trees are in sync. Every time a change is done in the provider, this value will change and so should the one in the consumer(s).

If your connection is slow and/or your ldap database large, it might take a while for the consumer's *contextCSN* match the provider's. But, you will know it is progressing since the consumer's *contextCSN* will be steadly increasing.

If the consumer's *contextCSN* is missing or does not match the provider, you should stop and figure out the issue before continuing. Try checking the slapd entries in /var/log/syslog in the provider to see if the consumer's authentication requests were successful or its requests to retrieve data return no errors. In particular, verify that you can connect to the provider from the consumer as the replicator binddn using $START_TLS$:

ldapwhoami -x -ZZ -D cn=replicator,dc=example,dc=com -W -h ldap01.example.com

For our example, you should now see the *john* user in the replicated tree:

\$ ldapsearch -x -LLL -b dc=example,dc=com -h ldap02.example.com '(uid=john)' uid dn: uid=john,ou=People,dc=example,dc=com uid: john

References

- Replication types, OpenLDAP Administrator's Guide
- LDAP Sync Replication OpenLDAP Administrator's Guide
- RFC 4533.

LDAP Workstation Authentication

Once you have a working LDAP server, you will need to install libraries on the client that will know how and when to contact it. On Ubuntu, this has been traditionally accomplished by installing the libras-ldap package, but nowadays you should use SSSD. Please refer to SSSD and LDAP.

User and Group Management - Idapscripts

Another very common usage case for having an LDAP server is to store unix user and group information in the directory. There are many tools out there, but usually big deployments will have developed their own. Here we will briefly show how to use the ldapscripts package for an easy and quick way to start storing user and group information in OpenLDAP.

Install the package:

sudo apt install ldapscripts

Then edit the file /etc/ldapscripts.conf to arrive at something similar to the following:

SERVER=ldap://ldap01.example.com LDAPBINOPTS="-ZZ" BINDDN='cn=admin,dc=example,dc=com' BINDPWDFILE="/etc/ldapscripts/ldapscripts.passwd" SUFFIX='dc=example,dc=com' GSUFFIX='ou=Groups' USUFFIX='ou=Groups' USUFFIX='ou=Computers'

Note

- Adjust SERVER and related SUFFIX options to suit your directory structure.
- Note we are forcing *START_TLS* usage here (-*ZZ* parameter), please refer to LDAP with TLS for details on how to set the server up with TLS support

Store the cn=admin password in the /etc/ldapscripts/ldapscripts.passwd file and make sure it's only readable by the *root* local user:

\$ sudo chmod 400 /etc/ldapscripts/ldapscripts.passwd

The scripts are now ready to help manage your directory. Here are some examples of how to use them:

• Create a new user:

sudo ldapaddgroup george sudo ldapadduser george george

This will create a group and user with name george and set the user's primary group (gid) to george

• Change a user's password:

\$ sudo ldapsetpasswd george Changing password for user uid=george,ou=People,dc=example,dc=com New Password: Retype New Password: Successfully set password for user uid=george,ou=People,dc=example,dc=com

• Delete a user:

sudo ldapdeleteuser george

Note

This won't delete the user's primary group, but will remove the user from supplementary ones.

• Add a group:

sudo ldapaddgroup qa

• Delete a group:

sudo ldapdeletegroup qa

• Add a user to a group:

sudo ldapaddusertogroup george qa

You should now see a *memberUid* attribute for the qa group with a value of george.

• Remove a user from a group:

sudo ldapdeleteuserfromgroup george qa

The *memberUid* attribute should now be removed from the qa group.

• The ldapmodifyuser script allows you to add, remove, or replace a user's attributes. The script uses the same syntax as the ldapmodify utility. For example:

sudo ldapmodifyuser george
About to modify the following entry :
dn: uid=george,ou=People,dc=example,dc=com
objectClass: account
objectClass: posixAccount
cn: george
uid: george
uid.george
uidNumber: 10001
gidNumber: 10001
homeDirectory: /home/george
loginShell: /bin/bash
gecos: george
description: User account
userPassword:: e1NTSEF9eXFsTFcyWlhwWkF1eGUybVdFWHZKRzJVMjFTSG9vcHk=

```
# Enter your modifications here, end with CTRL-D.
dn: uid=george,ou=People,dc=example,dc=com
replace: gecos
gecos: George Carlin
```

The user's gecos should now be "George Carlin".

• A nice feature of ldapscripts is the template system. Templates allow you to customize the attributes of user, group, and machine objects. For example, to enable the *user* template edit /etc/ldapscripts/ldapscripts.conf changing:

UTEMPLATE="/etc/ldapscripts/ldapadduser.template"

There are *sample* templates in the /usr/share/doc/ldapscripts/examples directory. Copy or rename the lda-padduser.template.sample file to /etc/ldapscripts/ldapadduser.template:

sudo cp /usr/share/doc/ldapscripts/examples/ldapadduser.template.sample \
/etc/ldapscripts/ldapadduser.template

Edit the new template to add the desired attributes. The following will create new users with an objectClass of inetOrgPerson:

dn: uid=<user>,<usuffix>,<suffix>
objectClass: inetOrgPerson
objectClass: posixAccount
cn: <user>
sn: <ask>
uid: <user>
uidNumber: <uid>
gidNumber: <gid>
homeDirectory: <home>

loginShell: <shell>
gecos: <user>
description: User account
title: Employee

Notice the *<ask>* option used for the *sn* attribute. This will make *ldapadduser* prompt you for its value.

There are utilities in the package that were not covered here. This command will output a list:

dpkg -L ldapscripts | grep /usr/sbin

LDAP & TLS

When authenticating to an OpenLDAP server it is best to do so using an encrypted session. This can be accomplished using Transport Layer Security (TLS).

Here, we will be our own *Certificate Authority* and then create and sign our LDAP server certificate as that CA. This guide will use the *certtool* utility to complete these tasks. For simplicity, this is being done on the OpenLDAP server itself, but your real internal CA should be elsewhere.

Install the *gnutls-bin* and *ssl-cert* packages:

sudo apt install gnutls-bin ssl-cert

Create a private key for the Certificate Authority:

```
sudo certtool --generate-privkey --bits 4096 --outfile /etc/ssl/private/mycakey.pem
```

Create the template/file /etc/ssl/ca.info to define the CA:

cn = Example Company
ca

cert_signing_key
expiration_days = 3650

Create the self-signed CA certificate:

sudo certtool --generate-self-signed \
--load-privkey /etc/ssl/private/mycakey.pem \
--template /etc/ssl/ca.info \
--outfile /usr/local/share/ca-certificates/mycacert.crt

Note

Yes, the *-outfile* path is correct, we are writing the CA certificate to */usr/local/share/ca-certificates*. This is where *update-ca-certificates* will pick up trusted local CAs from. To pick up CAs from */usr/share/ca-certificates*, a call to dpkg-reconfigure ca-certificates is necessary.

Run update-ca-certificates to add the new CA certificate to the list of trusted CAs. Note the one added CA:

```
$ sudo update-ca-certificates
Updating certificates in /etc/ssl/certs...
1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
```

This also creates a /etc/ssl/certs/mycacert.pem symlink pointing to the real file in /usr/local/share/ca-certificates.

Make a private key for the server:

```
sudo certtool --generate-privkey \
--bits 2048 \
--outfile /etc/ldap/ldap01_slapd_key.pem
```

Note

Replace ldap01 in the filename with your server's hostname. Naming the certificate and key for the host and service that will be using them will help keep things clear.

Create the /etc/ssl/ldap01.info info file containing:

```
organization = Example Company
cn = ldap01.example.com
tls_www_server
```

encryption_key
signing_key
expiration_days = 365

The above certificate is good for 1 year, and it's valid only for the ldap01.example.com hostname. Adjust accordingly.

Create the server's certificate:

```
sudo certtool --generate-certificate \
--load-privkey /etc/ldap/ldap01_slapd_key.pem \
--load-ca-certificate /etc/ssl/certs/mycacert.pem \
--load-ca-privkey /etc/ssl/private/mycakey.pem \
--template /etc/ssl/ldap01.info \
--outfile /etc/ldap/ldap01_slapd_cert.pem
```

Adjust permissions and ownership:

sudo chgrp openldap /etc/ldap/ldap01_slapd_key.pem sudo chmod 0640 /etc/ldap/ldap01_slapd_key.pem

Your server is now ready to accept the new TLS configuration.

Create the file certinfo.ldif with the following contents (adjust paths and filenames accordingly):

```
dn: cn=config
add: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ssl/certs/mycacert.pem
-
add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/ldap01 slapd cert.pem
```

```
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/ldap01_slapd_key.pem
```

Use the ldapmodify command to tell slapd about our TLS work via the slapd-config database:

sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f certinfo.ldif

If you need access to *LDAPS* (LDAP over SSL), then you need to edit /etc/default/slapd and include ldaps:/// in SLAPD_SERVICES like below:

SLAPD_SERVICES="ldap:/// ldapi:/// ldaps:///"

And restart slapd with: sudo systemctl restart slapd.

Note that *StartTLS* will be available without the change above, and does NOT need a slapd restart.

Test *StartTLS*:

```
$ ldapwhoami -x -ZZ -H ldap://ldap01.example.com
anonymous
```

Test *LDAPS*:

\$ ldapwhoami -x -H ldaps://ldap01.example.com
anonymous

Certificate for an OpenLDAP replica

To generate a certificate pair for an OpenLDAP replica (consumer), create a holding directory (which will be used for the eventual transfer) and:

```
mkdir ldap02-ssl
cd ldap02-ssl
certtool --generate-privkey \
    --bits 2048 \
    --outfile ldap02_slapd_key.pem
```

Create an info file, ldap02.info, for the Consumer server, adjusting its values accordingly:

organization = Example Company cn = ldap02.example.com tls_www_server encryption_key signing_key
expiration_days = 365

Create the Consumer's certificate:

```
sudo certtool --generate-certificate \
--load-privkey ldap02_slapd_key.pem \
--load-ca-certificate /etc/ssl/certs/mycacert.pem \
--load-ca-privkey /etc/ssl/private/mycakey.pem \
--template ldap02.info \
--outfile ldap02 slapd cert.pem
```

Note

We had to use sudo to get access to the CA's private key. This means the generated certificate file is owned by root. You should change that ownership back to your regular user before copying these files over to the Consumer.

Get a copy of the CA certificate:

cp /etc/ssl/certs/mycacert.pem .

We're done. Now transfer the ldap02-ssl directory to the Consumer. Here we use scp (adjust accordingly):

cd ..

scp -r ldap02-ssl user@consumer:

On the Consumer side, install the certificate files you just transferred:

```
sudo cp ldap02_slapd_cert.pem ldap02_slapd_key.pem /etc/ldap
sudo chgrp openldap /etc/ldap/ldap02_slapd_key.pem
sudo chmod 0640 /etc/ldap/ldap02_slapd_key.pem
sudo cp mycacert.pem /usr/local/share/ca-certificates/mycacert.crt
sudo update-ca-certificates
```

Create the file certinfo.ldif with the following contents (adjust accordingly regarding paths and filenames, if needed):

```
dn: cn=config
add: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ssl/certs/mycacert.pem
```

```
add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/ldap02_slapd_cert.pem
```

```
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/ldap02_slapd_key.pem
```

Configure the slapd-config database:

```
sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f certinfo.ldif
```

Like before, if you want to enable *LDAPS*, edit /etc/default/slapd and add ldaps:/// to SLAPD_SERVICES, and then restart slapd.

```
Test StartTLS:
```

\$ ldapwhoami -x -ZZ -H ldap://ldap02.example.com
anonymous

Test LDAPS:

```
$ ldapwhoami -x -H ldaps://ldap02.example.com
anonymous
```

OpenLDAP - Backup and Restore

Now we have ldap running just the way we want, it is time to ensure we can save all of our work and restore it as needed.

What we need is a way to backup the directory database(s), specifically the configuration backend (cn=config) and the DIT (dc=example, dc=com). If we are going to backup those databases into, say, /export/backup, we could use *slapcat* as shown in the following script, called /usr/local/bin/ldapbackup:

#!/bin/bash

set -e

BACKUP_PATH=/export/backup SLAPCAT=/usr/sbin/slapcat

```
nice ${SLAPCAT} -b cn=config > ${BACKUP_PATH}/config.ldif
nice ${SLAPCAT} -b dc=example,dc=com > ${BACKUP_PATH}/example.com.ldif
chown root:root ${BACKUP_PATH}/*
chmod 600 ${BACKUP_PATH}/*.ldif
```

Note

These files are uncompressed text files containing everything in your directory including the tree layout, usernames, and every password. So, you might want to consider making /export/backup an encrypted partition and even having the script encrypt those files as it creates them. Ideally you should do both, but that depends on your security requirements.

Then, it is just a matter of having a cron script to run this program as often as you feel comfortable with. For many, once a day suffices. For others, more often is required. Here is an example of a cron script called /etc/cron.d/ldapbackup that is run every night at 22:45h:

MAILTO=backup-emails@domain.com
45 22 * * * root /usr/local/bin/ldapbackup

Now the files are created, they should be copied to a backup server.

Assuming we did a fresh reinstall of ldap, the restore process could be something like this:

```
#!/bin/bash
```

set -e

```
BACKUP_PATH=/export/backup
SLAPADD=/usr/sbin/slapadd
```

```
if [ -n "$(ls -l /var/lib/ldap/* 2>/dev/null)" -o -n "$(ls -l /etc/ldap/slapd.d/* 2>/dev/null)" ]; then
    echo Run the following to remove the existing db:
    echo sudo systemctl stop slapd.service
    echo sudo rm -rf /etc/ldap/slapd.d/* /var/lib/ldap/*
    exit 1
fi
sudo systemctl stop slapd.service || :
sudo slapadd -F /etc/ldap/slapd.d -b cn=config -l /export/backup/config.ldif
sudo slapadd -F /etc/ldap/slapd.d -b dc=example,dc=com -l /export/backup/example.com.ldif
sudo chown -R openldap:openldap /etc/ldap/slapd.d/
sudo systemctl start slapd.service
```

This is a simplistic backup strategy, of course. It's being shown here as a reference for the basic tooling you can use for backups and restores.

Samba

Computer networks are often comprised of diverse systems, and while operating a network made up entirely of Ubuntu desktop and server computers would certainly be fun, some network environments must consist of both Ubuntu and Microsoft Windows systems working together in harmony. This section of the Samba guide introduces principles and tools used in configuring your Ubuntu Server for sharing network resources with Windows computers.

Samba Introduction

Successfully networking your Ubuntu system with Windows clients involves providing and integrating with services common to Windows environments. Such services assist the sharing of data and information about the computers and users involved in the network, and maybe classified under three major categories of functionality:

- File and Printer Sharing Services. Using the Server Message Block (SMB) protocol to facilitate the sharing of files, folders, volumes, and the sharing of printers throughout the network.
- **Directory Services**. Sharing vital information about the computers and users of the network with such technologies as the Lightweight Directory Access Protocol (LDAP) and Microsoft Active Directory.
- Authentication and Access. Establishing the identity of a computer or user of the network and determining the information the computer or user is authorized to access using such principles and technologies as file permissions, group policies, and the Kerberos authentication service.

Fortunately, your Ubuntu system may provide all such facilities to Windows clients and share network resources among them. One of the principal pieces of software your Ubuntu system includes for Windows networking is the Samba suite of SMB server applications and tools.

This section of the Samba guide will introduce some of the common Samba use cases, and how to install and configure the necessary packages. Additional detailed documentation and information on Samba can be found on the Samba website.

Member Server in an Active Directory Domain

In order to have a Samba server serve files and printers to Active Directory users, this Samba server needs to join the AD domain. This is slightly different from what is explained in Network User Authentication with SSSD There, we integrate the AD users and groups into the local Ubuntu system, as if they were local. In order for Samba to authenticate these users via SMB authentication protocols, not only do we need the remote users to be "seen", but Samba itself needs to be aware of the domain. In this scenario, Samba is called a Member Server or Domain Member.

Methods to join an Active Directory Domain

Samba itself has the necessary tooling to join an Active Directory domain. It will require a sequence of manual steps and configuration file editing, and it's documented upstream. It's useful to read that documentation to get an idea of the steps necessary, and decisions that have to be made.

For this guide, though, we are going to use the realmd package and instruct it to use the Samba tooling for joining the domain. This package will make certain decisions for us which will work for most cases, but more complex setups involving multiple or very large domains might require some tweaking.

Using realmd

First, let's install the necessary packages:

```
$ sudo apt install realmd samba
```

In order to have the joined machine registered in the AD DNS, it needs to have an FQDN set. You might have that already, if this command returns a full hostname with domain:

```
$ hostname -f
```

If it doesn't, then set it:

\$ sudo hostnamectl hostname <yourfqdn>

For this guide, we will be using jl.internal.example.fake, and the AD domain will be internal.example.fake.

Verify that the AD server is reachable and known:

\$ sudo realm discover internal.example.fake

```
internal.example.fake
type: kerberos
realm-name: INTERNAL.EXAMPLE.FAKE
domain-name: internal.example.fake
configured: no
server-software: active-directory
client-software: sssd
required-package: sssd-tools
required-package: libnss-sss
required-package: libpam-sss
required-package: adcli
required-package: samba-common-bin
```

realm is suggesting a set of packages for the discovered domain, but we will override that and select the Samba tooling for this join, because we want Samba to become a Member Server. Let's join the domain (in verbose mode so we can see all the steps):

\$ sudo realm join -v --membership-software=samba --client-software=winbind internal.example.fake

- * Resolving: _ldap._tcp.internal.example.fake
- * Performing LDAP DSE lookup on: 10.0.16.5
- * Successfully discovered: internal.example.fake

Password for Administrator:

- * Unconditionally checking packages
- * Resolving required packages
- * Installing necessary packages: libnss-winbind samba-common-bin libpam-winbind winbind

* LANG=C LOGNAME=root /usr/bin/net --configfile /var/cache/realmd/realmd-smb-conf.A53N01 -U Administrator -

-use-kerberos=required ads join internal.example.fake

Password for [INTEXAMPLE\Administrator]:

Using short domain name -- INTEXAMPLE

Joined 'J1' to dns domain 'internal.example.fake'

* LANG=C LOGNAME=root /usr/bin/net --configfile /var/cache/realmd/realmd-smb-conf.A53N01 -U Administrator ads keytab cre Password for [INTEXAMPLE\Administrator]:

- * /usr/sbin/update-rc.d winbind enable
- * /usr/sbin/service winbind restart
- * Successfully enrolled machine in realm

NOTE

This procedure also installed the libpam-winbind package, which allows AD users to authenticate to other services on this system via PAM, like ssh or console logins. For example, if your SSH server allows password authentication (PasswordAuthentication yes in /etc/ssh/sshd_config), then the domain users will be allowed to login remotely on this system via SSH.

If you don't expect or need AD users to login on this system unless it's via Samba or Windows, then it's safe (and probably best) to remove the libpam-winbind package.

Until bug #1980246 is fixed, though, one extra step is needed: configure /etc/nsswitch.conf. Please add the winbind word to the passwd and group lines like shown below:

passwd: files systemd winbind group: files systemd winbind

Now you will be able to query users from the AD domain. Winbind adds the short domain name as a prefix to domain users and groups:

\$ getent passwd INTEXAMPLE\\Administrator

You can find out the short domain name in the realm output shown earlier, or inspect the workgroup parameter of /etc/samba/smb.conf.

When domain users and groups are brought to the Linux world, a bit of translation needs to happen, and sometimes new values need to be created. For example, there is no concept of a "login shell" for AD users, but it exists on the Linux side.

The following are some common /etc/samba/smb.conf options that many installations might want to tweak. The smb.conf(5) manpage explains the % variable substitutions and other details:

- home directory: template homedir = /home/%U@%D. Another popular choice is /home/%D/%U
- login shell: template shell = /bin/bash
- winbind separator = \: this is the \ character between the short domain name and the user or group name that we saw in the getent passwd output above.
- winbind use default domain: if this is set to yes, then the domain name will not be part of the users and groups. Setting this to yes makes the system more friendly towards Linux users, as they won't have to remember to include the domain name everytime a user or group is referenced. If multiple domains are involved, though, like in an AD forest or some other form of domain trust relationship, then leave this setting at its default of no.

To have the home directory created automatically the first time a user logs in on the system, and if you haven't removed libpam-winbind, enable the pam_mkhomedir module via this command:

\$ sudo pam-auth-update --enable mkhomedir

Note that this won't apply to logins via Samba: this only creates the home directory for system logins like those via ssh or the console.

Exporting shares

Shares can be exported as usual. Since this is now a Member Server, there is no need to deal with user and group management. All of this is integrated with the Active Directory server we joined.

For example, let's create a simple [storage] share. Add this to the /etc/samba/smb.conf file:

[storage]

```
path = /storage
comment = Storage share
writable = yes
guest ok = no
```

Then create the /storage directory. Let's also make it 1777 so all users can use it, and then ask samba to reload its configuration:

\$ sudo mkdir -m 1777 /storage \$ sudo smbcontrol smbd reload-config

With this, users from the AD domain will be able to access this share. For example, if there is an user ubuntu the following command would access the share from another system, using the domain credentials:

```
$ smbclient //j1.internal.example.fake/storage -U INTEXAMPLE\\ubuntu
Enter INTEXAMPLE\ubuntu's password:
Try "help" to get a list of possible commands.
smb: \>
```

And smbstatus on the member server will show the connected user:

\$ sudo smbstatus

Samba version 4.15.5-Ubuntu							
PID	Username	Group	Machine	Protocol Ve	Protocol Version Encrypt		
3631	ΤΝΤΕΧΔΜΡΙ	E\ubuptu TN	- FEXAMPLE\domain users	10 0 16 1 (ipv4·10 0 16 1·3	80534)	SMB3 11	
partia	1 (AES-128-0	CMAC)		10.0.10.1 (1904.10.0.10.1.)	(,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	5105_11	
Servio	e pid	Machine	Connected at	Encrypti	on Signir	ng	
storad	ie 3631	10.0.16	.1 Wed Jun 29 17	:42:54 2022 UTC -	-		

No locked files

You can also restrict access to the share as usual, just keep in mind the syntax for the domain users. For example, to restrict access to the [storage] share we just created to only members of the LTS Releases domain group, add the valid users parameter like below:

```
[storage]
   path = /storage
   comment = Storage share
   writable = yes
   guest ok = no
   valid users = "@INTEXAMPLE\LTS Releases"
```

Choosing an idmap backend

realm made some choices for us when we joined the domain. A very important one is the idmap backend, and it might need changing for more complex setups.

User and group identifiers on the AD side are not directly usable as identifier on the Linux site. A *mapping* needs to be performed.

Winbind supports several idmap backends, and each one has its own manpage. The three main ones are:

- idmap_ad
- idmap autorid
- idmap_rid

Choosing the correct backend for each deployment type needs careful planing. Upstream has some guidelines at Choosing an idmap backend, and each manpage has more details and recommendations.

The realm tool selects by default the rid backend. This backend uses an algorithm to calculate the unix user and group ids from the respective RID value on the AD side. You might need to review the idmap config settings in /etc/samba/smb.conf and make sure they can accomodate the number of users and groups that exist in the domain, and that the range does not overlap with users from other sources.

For example, these settings:

```
idmap config * : range = 10000-999999
idmap config intexample : backend = rid
idmap config intexample : range = 2000000-2999999
idmap config * : backend = tdb
```

Will reserve the 2,000,000 through 2,999,999 range for user and group ids allocations on the Linux side for the intexample domain. The default backend (*, which acts as a "globbing" catch-all rule)) is used for the BUILTIN user and groups, and other domains should they exist. It's important that these ranges do not overlap.

The Administrator user we inspected before with getent passwd can give us a glimpse of how these ranges are used (output format changed for clarity):

```
$ id INTEXAMPLE\\Administrator
uid=2000500(INTEXAMPLE\administrator)
gid=2000513(INTEXAMPLE\domain users)
groups=2000513(INTEXAMPLE\domain users),
        2000500(INTEXAMPLE\doministrator),
        2000572(INTEXAMPLE\denied rodc password replication group),
        2000519(INTEXAMPLE\denied rodc password replication group),
        2000512(INTEXAMPLE\denied rodc password replication group),
        2000512(INTEXAMPLE\domain admins),
        10001(BUILTIN\users),
        10000(BUILTIN\administrators)
```

Resources

• The Samba Wiki

NOTE

This section is flagged as *legacy* because nowadays Samba can be deployed in full Active Directory Domain Controller mode, and the old style NT4 Primary Domain Controller is deprecated.

As an NT4 Domain Controller

A Samba server can be configured to appear as a Windows NT4-style domain controller. A major advantage of this configuration is the ability to centralize user and machine credentials. Samba can also use multiple backends to store the user information.

Primary Domain Controller

This section covers configuring Samba as a Primary Domain Controller (PDC) using the default smbpasswd backend.

First, install Samba, and libpam-winbind to sync the user accounts, by entering the following in a terminal prompt:

sudo apt install samba libpam-winbind

Next, configure Samba by editing /etc/samba/smb.conf. The *security* mode should be set to *user*, and the *workgroup* should relate to your organization:

```
workgroup = EXAMPLE
...
security = user
```

In the commented "Domains" section add or uncomment the following (the last line has been split to fit the format of this document):

```
domain logons = yes
logon path = \\%N\%U\profile
logon drive = H:
logon home = \\%N\%U
```

```
logon script = logon.cmd
add machine script = sudo /usr/sbin/useradd -N -g machines -c Machine -d
/var/lib/samba -s /bin/false %u
```

Note

If you wish to not use *Roaming Profiles* leave the logon home and logon path options commented.

- domain logons: provides the netlogon service causing Samba to act as a domain controller.
- *logon path:* places the user's Windows profile into their home directory. It is also possible to configure a *[profiles]* share placing all profiles under a single directory.
- logon drive: specifies the home directory local path.
- logon home: specifies the home directory location.
- *logon script:* determines the script to be run locally once a user has logged in. The script needs to be placed in the *[netlogon]* share.
- *add machine script:* a script that will automatically create the *Machine Trust Account* needed for a workstation to join the domain.

In this example the *machines* group will need to be created using the addgroup utility see Security - Users: Adding and Deleting Users for details.

Uncomment the *[homes]* share to allow the *logon home* to be mapped:

```
[homes]
```

```
comment = Home Directories
browseable = no
read only = no
create mask = 0700
directory mask = 0700
valid users = %S
```

When configured as a domain controller a *[netlogon]* share needs to be configured. To enable the share, uncomment:

[netlogon]

```
comment = Network Logon Service
path = /srv/samba/netlogon
guest ok = yes
read only = yes
share modes = no
```

Note

The original *netlogon* share path is /home/samba/netlogon, but according to the Filesystem Hierarchy Standard (FHS), /srv is the correct location for site-specific data provided by the system.

Now create the netlogon directory, and an empty (for now) logon.cmd script file:

sudo mkdir -p /srv/samba/netlogon
sudo touch /srv/samba/netlogon/logon.cmd

You can enter any normal Windows logon script commands in logon.cmd to customize the client's environment.

Restart Samba to enable the new domain controller:

sudo systemctl restart smbd.service nmbd.service

Lastly, there are a few additional commands needed to setup the appropriate rights.

With *root* being disabled by default, in order to join a workstation to the domain, a system group needs to be mapped to the Windows *Domain Admins* group. Using the net utility, from a terminal enter:

sudo net groupmap add ntgroup="Domain Admins" unixgroup=sysadmin rid=512 type=d

Note

Change *sysadmin* to whichever group you prefer. Also, the user used to join the domain needs to be a member of the *sysadmin* group, as well as a member of the system *admin* group. The *admin* group allows sudo use.

If the user does not have Samba credentials yet, you can add them with the smbpasswd utility, change the *sysadmin* username appropriately:

sudo smbpasswd -a sysadmin

Also, rights need to be explicitly provided to the *Domain Admins* group to allow the *add machine script* (and other admin functions) to work. This is achieved by executing:

net rpc rights grant -U sysadmin "EXAMPLE\Domain Admins" SeMachineAccountPrivilege \
SePrintOperatorPrivilege SeAddUsersPrivilege SeDiskOperatorPrivilege \
SeRemoteShutdownPrivilege

You should now be able to join Windows clients to the Domain in the same manner as joining them to an NT4 domain running on a Windows server.

Backup Domain Controller

With a Primary Domain Controller (PDC) on the network it is best to have a Backup Domain Controller (BDC) as well. This will allow clients to authenticate in case the PDC becomes unavailable.

When configuring Samba as a BDC you need a way to sync account information with the PDC. There are multiple ways of accomplishing this scp, rsync, or by using LDAP as the *passdb backend*.

Using LDAP is the most robust way to sync account information, because both domain controllers can use the same information in real time. However, setting up a LDAP server may be overly complicated for a small number of user and computer accounts. See Samba - OpenLDAP Backend for details.

First, install samba and libpam-winbind. From a terminal enter:

sudo apt install samba libpam-winbind

Now, edit /etc/samba/smb.conf and uncomment the following in the [global]:

```
workgroup = EXAMPLE
...
security = user
```

In the commented *Domains* uncomment or add:

domain logons = yes
domain master = no

Make sure a user has rights to read the files in /var/lib/samba. For example, to allow users in the *admin* group to scp the files, enter:

sudo chgrp -R admin /var/lib/samba

Next, sync the user accounts, using scp to copy the /var/lib/samba directory from the PDC:

sudo scp -r username@pdc:/var/lib/samba /var/lib

Note

Replace *username* with a valid username and *pdc* with the hostname or IP Address of your actual PDC.

Finally, restart samba:

sudo systemctl restart smbd.service nmbd.service

You can test that your Backup Domain controller is working by stopping the Samba daemon on the PDC, then trying to login to a Windows client joined to the domain.

Another thing to keep in mind is if you have configured the *logon home* option as a directory on the PDC, and the PDC becomes unavailable, access to the user's *Home* drive will also be unavailable. For this reason it is best to configure the *logon home* to reside on a separate file server from the PDC and BDC.

Resources

- For in depth Samba configurations see the Samba HOWTO Collection
- The guide is also available in printed format.
- O'Reilly's Using Samba is also a good reference.
- Chapter 4 of the Samba HOWTO Collection explains setting up a Primary Domain Controller.
- Chapter 5 of the Samba HOWTO Collection explains setting up a Backup Domain Controller.
- The Ubuntu Wiki Samba page.

File Server

One of the most common ways to network Ubuntu and Windows computers is to configure Samba as a File Server. This section covers setting up a Samba server to share files with Windows clients.

The server will be configured to share files with any client on the network without prompting for a password. If your environment requires stricter Access Controls see Share Access Control.

Installation

The first step is to install the samba package. From a terminal prompt enter:

```
$ sudo apt install samba
```

That's all there is to it; you are now ready to configure Samba to share files.

Configuration

The main Samba configuration file is located in /etc/samba/smb.conf. The default configuration file has a significant number of comments in order to document various configuration directives.

Note

Not all the available options are included in the default configuration file. See the smb.conf man page or the Samba HOWTO Collection for more details.

First, edit the workgroup parameter in the [global] section of /etc/samba/smb.conf and change it to better match your environment:

workgroup = EXAMPLE

Create a new section at the bottom of the file, or uncomment one of the examples, for the directory to be shared:

[share]

```
comment = Ubuntu File Server Share
path = /srv/samba/share
browsable = yes
guest ok = yes
read only = no
create mask = 0755
```

- comment: a short description of the share. Adjust to fit your needs.
- *path:* the path to the directory to share.

This example uses /srv/samba/sharename because, according to the *Filesystem Hierarchy Standard (FHS)*, /srv is where site-specific data should be served. Technically Samba shares can be placed anywhere on the filesystem as long as the permissions are correct, but adhering to standards is recommended.

- browsable: enables Windows clients to browse the shared directory using Windows Explorer.
- guest ok: allows clients to connect to the share without supplying a password.
- *read only:* determines if the share is read only or if write privileges are granted. Write privileges are allowed only when the value is *no*, as is seen in this example. If the value is *yes*, then access to the share is read only.
- create mask: determines the permissions new files will have when created.

Now that Samba is configured, the directory needs to be created and the permissions changed. From a terminal enter:

\$ sudo mkdir -p /srv/samba/share \$ sudo chown nobody:nogroup /srv/samba/share/

Note

The -p switch tells mkdir to create the entire directory tree if it doesn't exist.

Finally, restart the samba services to enable the new configuration:

\$ sudo systemctl restart smbd.service nmbd.service

Warning

Once again, the above configuration gives all access to any client on the local network. For a more secure configuration see Share Access Control.

From a Windows client you should now be able to browse to the Ubuntu file server and see the shared directory. If your client doesn't show your share automatically, try to access your server by its IP address, e.g. \\192.168.1.1, in a Windows Explorer window. To check that everything is working try creating a directory from Windows.

To create additional shares simply create new [sharename] sections in /etc/samba/smb.conf, and restart Samba. Just make sure that the directory you want to share actually exists and the permissions are correct.

Note

The file share named "[share]" and the path /srv/samba/share are just examples. Adjust the share and path names to fit your environment. It is a good idea to name a share after a directory on the file system. Another example would be a share name of /qa with a path of /srv/samba/qa.

Resources

- For in depth Samba configurations see the Samba HOWTO Collection
- The guide is also available in printed format.
- O'Reilly's Using Samba is another good reference.
- The Ubuntu Wiki Samba page.

Print Server

Another common use of Samba is to configure it to share printers installed, either locally or over the network, on an Ubuntu server. Similar to File Server, this section will configure Samba to allow any client on the local network to use the installed printers without prompting for a username and password.

For a more secure configuration see Share Access Control.

Installation

Before installing and configuring Samba it is best to already have a working CUPS installation. See CUPS for details.

To install the samba package, from a terminal enter:

```
$ sudo apt install samba
```

Configuration

After installing samba edit /etc/samba/smb.conf. Change the *workgroup* attribute to what is appropriate for your network:

```
workgroup = EXAMPLE
```

In the *[printers]* section change the *guest ok* option to *yes*:

browsable = yes
guest ok = yes

After editing smb.conf restart Samba:

\$ sudo systemctl restart smbd.service nmbd.service

The default Samba configuration will automatically share any printers installed. Simply install the printer locally on your Windows clients.

Resources

- For in depth Samba configurations see the Samba HOWTO Collection
- The guide is also available in printed format.
- O'Reilly's Using Samba is another good reference.
- Also, see the CUPS Website for more information on configuring CUPS.
- The Ubuntu Wiki Samba page.

Share Access Control

There are several options available to control access for each individual shared directory. Using the *[share]* example, this section will cover some common options.

Groups

Groups define a collection of users which have a common level of access to particular network resources and offer a level of granularity in controlling access to such resources. For example, if a group *qa* is defined and contains the users *freda*, *danika*, and *rob* and a second group *support* is defined and consists of users *danika*, *jeremy*, and *vincent* then certain network resources configured to allow access by the *qa* group will subsequently enable access by freda, danika, and rob, but not jeremy or vincent. Since the user *danika* belongs to both the *qa* and *support* groups, she will be able to access resources configured for access by both groups, whereas all other users will have only access to resources explicitly allowing the group they are part of.

When mentioning groups in the Samba configuration file, /etc/samba/smb.conf, the recognized syntax is to preface the group name with an "@" symbol. For example, if you wished to use a group named *sysadmin* in a certain section of the /etc/samba.conf, you would do so by entering the group name as @sysadmin. if a group name has a space in it, use double quotes, like "@LTS Releases".

Read and Write Permissions

Read and write permissions define the explicit rights a computer or user has to a particular share. Such permissions may be defined by editing the /etc/samba/smb.conf file and specifying the explicit permissions inside a share.

For example, if you have defined a Samba share called *share* and wish to give *read-only* permissions to the group of users known as *qa*, but wanted to allow writing to the share by the group called *sysadmin* and the user named *vincent*, then you could edit the /etc/samba/smb.conf file, and add the following entries under the *[share]* entry:

```
read list = @qa
write list = @sysadmin, vincent
```

Another possible Samba permission is to declare *administrative* permissions to a particular shared resource. Users having administrative permissions may read, write, or modify any information contained in the resource the user has been given explicit administrative permissions to.

For example, if you wanted to give the user *melissa* administrative permissions to the *share* example, you would edit the /etc/samba/smb.conf file, and add the following line under the *[share]* entry:

admin users = melissa

After editing /etc/samba/smb.conf, reload Samba for the changes to take effect:

```
sudo smbcontrol smbd reload-config
```

Now that Samba has been configured to limit which groups have access to the shared directory, the filesystem permissions need to be checked.

Traditional Linux file permissions do not map well to Windows NT Access Control Lists (ACLs). Fortunately POSIX ACLs are available on Ubuntu servers providing more fine grained control. For example, to enable ACLs on /srv an EXT3 filesystem, edit /etc/fstab adding the *acl* option:

UUID=66bcdd2e-8861-4fb0-b7e4-e61c569fe17d /srv ext3 noatime,relatime,acl 0 1

Then remount the partition:

sudo mount -v -o remount /srv

Note

The above example assumes /srv on a separate partition. If /srv, or wherever you have configured your share path, is part of the / partition a reboot may be required.

To match the Samba configuration above the *sysadmin* group will be given read, write, and execute permissions to /srv/samba/share, the *qa* group will be given read and execute permissions, and the files will be owned by the username *melissa*. Enter the following in a terminal:

sudo chown -R melissa /srv/samba/share/ sudo chgrp -R sysadmin /srv/samba/share/ sudo setfacl -R -m g:qa:rx /srv/samba/share/

Note

The setfacl command above gives *execute* permissions to all files in the /srv/samba/share directory, which you may or may not want.

Now from a Windows client you should notice the new file permissions are implemented. See the acl and setfacl man pages for more information on POSIX ACLs.

Resources

- For in depth Samba configurations see the Samba HOWTO Collection
- The guide is also available in printed format.
- O'Reilly's Using Samba is also a good reference.
- Chapter 18 of the Samba HOWTO Collection is devoted to security.
- For more information on Samba and ACLs see the Samba ACLs page.
- The Ubuntu Wiki Samba page.

Samba AppArmor Profile

Ubuntu comes with the AppArmor security module, which provides mandatory access controls. The default AppArmor profile for Samba may need to be adapted to your configuration. More details on using AppArmor can be found here.

There are default AppArmor profiles for /usr/sbin/smbd and /usr/sbin/nmbd, the Samba daemon binaries, as part of the apparmor-profiles package. To install the package, from a terminal prompt enter:

```
$ sudo apt install apparmor-profiles apparmor-utils
```

Note

This package contains profiles for several other binaries.

By default the profiles for smbd and nmbd are in *complain* mode allowing Samba to work without modifying the profile, and only logging errors. There is no need to add exceptions for the shares, as the smbd service unit takes care of doing that automatically via a helper script.

This is what an ALLOWED message looks like. It means that, were the profile not in complain mode, this action would have been denied instead (formatted into multiple lines here for better visibility):

```
Jun 30 14:41:09 ubuntu kernel: [ 621.478989] audit:
type=1400 audit(1656600069.123:418):
apparmor="ALLOWED" operation="exec" profile="smbd"
name="/usr/lib/x86_64-linux-gnu/samba/samba-bgqd" pid=4122 comm="smbd"
requested_mask="x" denied_mask="x" fsuid=0 ouid=0
target="smbd//null-/usr/lib/x86_64-linux-gnu/samba/samba-bgqd"
```

To place the profile into *enforce* mode and reload it, run:

```
$ sudo aa-enforce /usr/sbin/smbd
```

```
$ sudo apparmor_parser -r -W -T /etc/apparmor.d/usr.sbin.smbd
```

It's advisable to monitor /var/log/syslog for audit entries that contain apparmor DENIED messages, or /var/log/audit/audit.log if you are running the auditd daemon. Actions blocked by apparmor may surface as odd or unrelated errors in the application.

Samba - OpenLDAP Backend

NOTE

This section is flagged as *legacy* because nowadays Samba 4 is best integrated with its own LDAP server in AD mode. Integrating Samba with LDAP as described here covers the NT4 mode, deprecated for many years.

This section covers the integration of Samba with LDAP. The Samba server's role will be that of a "standalone" server and the LDAP directory will provide the authentication layer in addition to containing the user, group, and machine account information that Samba requires in order to function (in any of its 3 possible roles). The pre-requisite is an OpenLDAP server configured with a directory that can accept authentication requests. See Service - LDAP with TLS for details on fulfilling this requirement. Once those steps are completed, you will need to decide what specifically you want Samba to do for you and then configure it accordingly.

This guide will assume that the LDAP and Samba services are running on the same server and therefore use SASL EXTERNAL authentication whenever changing something under cn=config. If that is not your scenario, you will have to run those ldap commands on the LDAP server.

Software Installation

There are two packages needed when integrating Samba with LDAP: samba and smbldap-tools.

Strictly speaking, the smbldap-tools package isn't needed, but unless you have some other way to manage the various Samba entities (users, groups, computers) in an LDAP context then you should install it.

Install these packages now:

```
sudo apt install samba smbldap-tools
```

LDAP Configuration

We will now configure the LDAP server so that it can accomodate Samba data. We will perform three tasks in this section:

- Import a schema
- Index some entries
- Add objects

Samba schema

In order for OpenLDAP to be used as a backend for Samba, the DIT will need to use attributes that can properly describe Samba data. Such attributes can be obtained by introducing a Samba LDAP schema. Let's do this now.

The schema is found in the now-installed samba package and is already in the ldif format. We can import it with one simple command:

```
sudo ldapadd -Q -Y EXTERNAL -H ldapi:/// -f /usr/share/doc/samba/examples/LDAP/samba.ldif
```

To query and view this new schema:

sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=schema.cn=config 'cn=*samba*'

Samba indices

Now that slapd knows about the Samba attributes, we can set up some indices based on them. Indexing entries is a way to improve performance when a client performs a filtered search on the DIT.

Create the file samba_indices.ldif with the following contents:

dn: olcDatabase={1}mdb,cn=config changetype: modify replace: olcDbIndex olcDbIndex: objectClass eq olcDbIndex: uidNumber,gidNumber eq olcDbIndex: loginShell eq olcDbIndex: uid,cn eq,sub olcDbIndex: memberUid eq,sub olcDbIndex: member,uniqueMember eq olcDbIndex: sambaSID eq olcDbIndex: sambaSID eq olcDbIndex: sambaGroupType eq olcDbIndex: sambaGroupType eq olcDbIndex: sambaSIDList eq olcDbIndex: sambaDomainName eq olcDbIndex: default sub,eq

Using the ldapmodify utility load the new indices:

sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// -f samba_indices.ldif

If all went well you should see the new indices using ldapsearch:

sudo ldapsearch -Q -LLL -Y EXTERNAL -H \ ldapi:/// -b cn=config olcDatabase={1}mdb olcDbIndex

Adding Samba LDAP objects

Next, configure the smbldap-tools package to match your environment. The package comes with a configuration helper script called *smbldap-config*. Before running it, though, you should decide on two important configuration settings in /etc/samba/smb.conf:

- *netbios name*: how this server will be known. The default value is derived from the server's hostname, but truncated at 15 characters.
- *workgroup*: the workgroup name for this server, or, if you later decide to make it a domain controller, this will be the domain.

It's important to make these choices now because *smbldap-config* will use them to generate the config that will be later stored in the LDAP directory. If you run *smbldap-config* now and later change these values in /etc/samba/smb.conf there will be an inconsistency.

Once you are happy with *netbios name* and *workgroup*, proceed to generate the *smbldap-tools* configuration by running the configuration script which will ask you some questions:

sudo smbldap-config

Some of the more important ones:

- workgroup name: has to match what you will configure in /etc/samba/smb.conf later on.
- *ldap suffix*: has to match the ldap suffix you chose when you configured the LDAP server.
- other ldap suffixes: they are all relative to *ldap suffix* above. For example, for *ldap user suffix* you should use *ou=People*, and for computer/machines, use *ou=Computers*.
- *ldap master bind dn* and *bind password*: use the rootDN credentials.

The *smbldap-populate* script will then add the LDAP objects required for Samba. It will ask you for a password for the "domain root" user, which is also the "root" user stored in LDAP:

sudo smbldap-populate -g 10000 -u 10000 -r 10000

The -g, -u and -r parameters tell smbldap-tools where to start the numeric uid and gid allocation for the LDAP users. You should pick a range start that does not overlap with your local /*etc/passwd* users.

You can create a LDIF file containing the new Samba objects by executing sudo smbldap-populate -e samba.ldif. This allows you to look over the changes making sure everything is correct. If it is, rerun the script without the '-e' switch. Alternatively, you can take the LDIF file and import its data per usual.

Your LDAP directory now has the necessary information to authenticate Samba users.

Samba Configuration

To configure Samba to use LDAP, edit its configuration file /etc/samba/smb.conf commenting out the default *passdb* backend parameter and adding some ldap-related ones. Make sure to use the same values you used when running *smbldap-populate*:

```
# passdb backend = tdbsam
workgroup = EXAMPLE
# LDAP Settings
passdb backend = ldapsam:ldap://ldap01.example.com
ldap suffix = dc=example,dc=com
ldap user suffix = ou=People
ldap group suffix = ou=Groups
ldap machine suffix = ou=Computers
ldap idmap suffix = ou=Idmap
ldap admin dn = cn=admin,dc=example,dc=com
ldap ssl = start tls
ldap passwd sync = yes
```

Change the values to match your environment.

Note

The smb.conf as shipped by the package is quite long and has many configuration examples. An easy way to visualize it without any comments is to run *testparm* -s.

Now inform Samba about the rootDN user's password (the one set during the installation of the slapd package):

```
sudo smbpasswd -W
```

As a final step to have your LDAP users be able to connect to samba and authenticate, we need these users to also show up in the system as "unix" users. Use SSSD for that as detailed in Service - SSSD.

Install sssd-ldap

sudo apt install sssd-ldap

Configure /etc/sssd/sssd.conf:

[sssd]
config_file_version = 2
domains = example.com

[domain/example.com] id_provider = ldap auth_provider = ldap ldap_uri = ldap://ldap01.example.com cache_credentials = True ldap_search_base = dc=example,dc=com

Adjust permissions and start the service:

sudo chmod 0600 /etc/sssd/sssd.conf sudo chown root:root /etc/sssd/sssd.conf sudo systemctl start sssd

Restart the Samba services:

sudo systemctl restart smbd.service nmbd.service

To quickly test the setup, see if getent can list the Samba groups:

\$ getent group Replicators
Replicators:*:552:

Note

The names are case sensitive!

If you have existing LDAP users that you want to include in your new LDAP-backed Samba they will, of course, also need to be given some of the extra Samba specific attributes. The *smbpasswd* utility can do this for you:

sudo smbpasswd -a username

You will prompted to enter a password. It will be considered as the new password for that user. Making it the same as before is reasonable. Note that this command cannot be used to create a new user from scratch in LDAP (unless you are using *ldapsam:trusted* and *ldapsam:editposix*, not covered in this guide).

To manage user, group, and machine accounts use the utilities provided by the *smbldap-tools* package. Here are some examples:

• To add a new user with a home directory:

```
sudo smbldap-useradd -a -P -m username
```

The -a option adds the Samba attributes, and the -P option calls the *smbldap-passwd* utility after the user is created allowing you to enter a password for the user. Finally, -m creates a local home directory. Test with the getent command:

getent passwd username

• To remove a user:

sudo smbldap-userdel username

In the above command, use the -r option to remove the user's home directory.

• To add a group:

sudo smbldap-groupadd -a groupname

As for *smbldap-useradd*, the *-a* adds the Samba attributes.

- To make an existing user a member of a group:
 - sudo smbldap-groupmod -m username groupname

The *-m* option can add more than one user at a time by listing them in comma-separated format.

• To remove a user from a group:

sudo smbldap-groupmod -x username groupname

• To add a Samba machine account:

sudo smbldap-useradd -t 0 -w username

Replace *username* with the name of the workstation. The $-t \ \theta$ option creates the machine account without a delay, while the -w option specifies the user as a machine account.

Resources

- Upstream documentation collection: https://www.samba.org/samba/docs/
- Upstream samba wiki: https://wiki.samba.org/index.php/Main_Page

Kerberos

Kerberos is a network authentication system based on the principal of a trusted third party. The other two parties being the user and the service the user wishes to authenticate to. Not all services and applications can use Kerberos, but for those that can, it brings the network environment one step closer to being Single Sign On (SSO).

This section covers installation and configuration of a Kerberos server, and some example client configurations.

Overview

If you are new to Kerberos there are a few terms that are good to understand before setting up a Kerberos server. Most of the terms will relate to things you may be familiar with in other environments:

- Principal: any users, computers, and services provided by servers need to be defined as Kerberos Principals.
- *Instances:* are a variation for service principals. For example, the principal for an NFS service will have an instance for the hostname of the server, like nfs/server.example.com@REALM. Similarly admin privileges on a principal use an instance of /admin, like john/admin@REALM, differentiating it from john@REALM. These variations fit nicely with ACLs.
- *Realms:* the unique realm of control provided by the Kerberos installation. Think of it as the domain or group your hosts and users belong to. Convention dictates the realm should be in uppercase. By default, Ubuntu will use the DNS domain converted to uppercase (EXAMPLE.COM) as the realm.
- *Key Distribution Center:* (KDC) consist of three parts: a database of all principals, the authentication server, and the ticket granting server. For each realm there must be at least one KDC.
- *Ticket Granting Ticket:* issued by the Authentication Server (AS), the Ticket Granting Ticket (TGT) is encrypted in the user's password which is known only to the user and the KDC. This is the starting point for a user to acquire additional tickets for the services being accessed.
- Ticket Granting Server: (TGS) issues service tickets to clients upon request.
- *Tickets:* confirm the identity of the two principals. One principal being a user and the other a service requested by the user. Tickets establish an encryption key used for secure communication during the authenticated session.
- Keytab Files: contain encryption keys for a service or host extracted from the KDC principal database.

To put the pieces together, a Realm has at least one KDC, preferably more for redundancy, which contains a database of Principals. When a user principal logs into a workstation that is configured for Kerberos authentication, the KDC issues a Ticket Granting Ticket (TGT). If the user supplied credentials match, the user is authenticated and can then request tickets for Kerberized services from the Ticket Granting Server (TGS). The service tickets allow the user to authenticate to the service without entering another username and password.

Resources

- For more information on MIT's version of Kerberos, see the MIT Kerberos site.
- Also, feel free to stop by the #ubuntu-server and #kerberos IRC channels on Libera.Chat if you have Kerberos questions.
- Another guide for installing Kerberos on Debian, includes PKINIT

Kerberos Server

Installation

For this discussion, we will create a MIT Kerberos domain with the following features (edit them to fit your needs):

- *Realm:* EXAMPLE.COM
- Primary KDC: kdc01.example.com
- Secondary KDC: kdc02.example.com
- User principal: ubuntu
- Admin principal: ubuntu/admin

Before installing the Kerberos server, a properly configured DNS server is needed for your domain. Since the Kerberos Realm by convention matches the domain name, this section uses the EXAMPLE.COM domain configured in the section Primary Server of the DNS documentation.

Also, Kerberos is a time sensitive protocol. If the local system time between a client machine and the server differs by more than five minutes (by default), the workstation will not be able to authenticate. To correct the problem all hosts should have their time synchronized using the same *Network Time Protocol (NTP)* server. Check out the NTP chapter for more details.

The first step in creating a Kerberos Realm is to install the krb5-kdc and krb5-admin-server packages. From a terminal enter:

sudo apt install krb5-kdc krb5-admin-server

You will be asked at the end of the install to supply the hostname for the Kerberos and Admin servers, which may or may not be the same server, for the realm. Since we are going to create the realm, and thus these servers, type in the full hostname of this server.

Note

By default the realm name will be domain name of the KDC server.

Next, create the new realm with the $\tt kdb5_newrealm$ utility:

sudo krb5_newrealm

It will ask you for a database master password, which is used to encrypt the local database. Chose a secure password: its strength is not verified for you.

Configuration

The questions asked during installation are used to configure the /etc/krb5.conf and /etc/krb5kdc/kdc.conf files. The former is used by the kerberos 5 libraries, and the latter configures the KDC. If you need to adjust the Key Distribution Center (KDC) settings simply edit the file and restart the krb5-kdc daemon. If you need to reconfigure Kerberos from scratch, perhaps to change the realm name, you can do so by typing

sudo dpkg-reconfigure krb5-kdc

Note

The manpage for krb5.conf is in the krb5-doc package.

Let's create our first principal. Since there is no principal create yet, we need to use kadmin.local, which uses a local unix socket to talk to the KDC, and requires root privileges:

```
$ sudo kadmin.local
Authenticating as principal root/admin@EXAMPLE.COM with password.
kadmin.local: addprinc ubuntu
WARNING: no policy specified for ubuntu@EXAMPLE.COM; defaulting to no policy
```

Enter password for principal "ubuntu@EXAMPLE.COM": Re-enter password for principal "ubuntu@EXAMPLE.COM": Principal "ubuntu@EXAMPLE.COM" created. kadmin.local: quit

To be able to use kadmin remotely, we should create an *admin principal*. Convention suggests it should be an *admin instance*, as that also makes creating generic ACLs easier. Let's create an *admin* instance for the *ubuntu* principal:

```
$ sudo kadmin.local
Authenticating as principal root/admin@EXAMPLE.COM with password.
kadmin.local: addprinc ubuntu/admin
WARNING: no policy specified for ubuntu/admin@EXAMPLE.COM; defaulting to no policy
Enter password for principal "ubuntu/admin@EXAMPLE.COM":
Re-enter password for principal "ubuntu/admin@EXAMPLE.COM":
Principal "ubuntu/admin@EXAMPLE.COM" created.
kadmin.local: guit
```

Next, the new admin principal needs to have the appropriate Access Control List (ACL) permissions. The permissions are configured in the /etc/krb5kdc/kadm5.acl file:

ubuntu/admin@EXAMPLE.COM

You can also use a more generic form for this ACL:

*/admin@EXAMPLE.COM

The above will grant all privileges to any *admin instance* of a principal. See the kadm5.acl manpage for details.

Now restart the krb5-admin-server for the new ACL to take affect:

sudo systemctl restart krb5-admin-server.service

The new user principal can be tested using the kinit utility:

*

\$ kinit ubuntu/admin
Password for ubuntu/admin@EXAMPLE.COM:

After entering the password, use the klist utility to view information about the Ticket Granting Ticket (TGT):

\$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: ubuntu/admin@EXAMPLE.COM

Valid starting Expires Service principal 04/03/20 19:16:57 04/04/20 05:16:57 krbtgt/EXAMPLE.COM@EXAMPLE.COM renew until 04/04/20 19:16:55

Where the cache filename krb5cc_1000 is composed of the prefix krb5cc_ and the user id (uid), which in this case is 1000.

kinit will inspect /etc/krb5.conf to find out which KDC to contact, and its address. The KDC can also be found via DNS lookups for special TXT and SRV records. You can add these records to your example.com DNS zone:

_kerberos._udp.EXAMPLE.COM.INSRV1088kdc01.example.com._kerberos._tcp.EXAMPLE.COM.INSRV1088kdc01.example.com._kerberos._udp.EXAMPLE.COM.INSRV10088kdc02.example.com._kerberos._tcp.EXAMPLE.COM.INSRV10088kdc02.example.com._kerberos-adm._tcp.EXAMPLE.COM.INSRV10749kdc01.example.com._kpasswd._udp.EXAMPLE.COM.INSRV10464kdc01.example.com.

See the DNS chapter for detailed instructions on setting up DNS.

A very quick and useful way to troubleshoot what kinit is doing is to set the environment variable KRB5_TRACE to a file, or stderr, and it will show extra information. The output is quite verbose:

```
$ KRB5_TRACE=/dev/stderr kinit ubuntu/admin
[2898] 1585941845.278578: Getting initial credentials for ubuntu/admin@EXAMPLE.COM
[2898] 1585941845.278580: Sending unauthenticated request
[2898] 1585941845.278581: Sending request (189 bytes) to EXAMPLE.COM
[2898] 1585941845.278582: Resolving hostname kdc01.example.com
(...)
```

Your new Kerberos Realm is now ready to authenticate clients.

Kerberos - Service Principals

The specific steps to enable Kerberos for a service can vary a bit, but in general the following is needed:

- a principal for the service: usually service/host@REALM
- a keytab accessible to the service wherever it's running: usually in /etc/krb5.keytab

For example, let's create a principal for an LDAP service running on the ldap-server.example.com host:

ubuntu@ldap-server:~\$ sudo kadmin -p ubuntu/admin Authenticating as principal ubuntu/admin with password. Password for ubuntu/admin@EXAMPLE.COM: kadmin: addprinc -randkey ldap/ldap-server.example.com No policy specified for ldap/ldap-server.example.com@EXAMPLE.COM; defaulting to no policy Principal "ldap/ldap-server.example.com@EXAMPLE.COM" created.

Let's dig a bit into what is happening here:

- the kadmin command is being run in the *ldap-server* machine, not on the KDC. We are using kadmin remotely
- it's being run with sudo, the reason will become clear later
- we are logged in on the server as *ubuntu*, but specifying a *ubuntu/admin* principal. Remember the *ubuntu* principal has no special privileges
- the name of the principal we are creating follows the pattern service/hostname
- in order to select a random secret, we pass the **-randkey** parameter. Otherwise we would be asked to type in a password.

With the principal created, we need to extract the key from the KDC and store it in the *ldap-server* host, so that the *ldap* service can use it to authenticate itself with the KDC. Still in the same *kadmin* session:

kadmin: ktadd ldap/ldap-server.example.com

Entry for principal ldap/ldap-server.example.com with kvno 2, encryption type aes256-cts-hmac-shal-96 added to keytab FILE:/etc/krb5.keytab.

Entry for principal ldap/ldap-server.example.com with kvno 2, encryption type aes128-cts-hmac-shal-96 added to keytab FILE:/etc/krb5.keytab.

This is why he needed to run kadmin with sudo: so that it can write to /etc/krb5.keytab. This is the system keytab file, which is the default file for all keys that might be needed for services on this host. And we can list them with klist. Back in the shell:

\$ sudo klist -k Keytab name: FILE:/etc/krb5.keytab KVNO Principal

- 2 ldap/ldap-server.example.com@EXAMPLE.COM
- 2 ldap/ldap-server.example.com@EXAMPLE.COM

If you don't have the kadmin utility on the target host, one alternative is to extract the keys on a different host and into a different file, and then transfer this file *securely* to the target server. For example:

kadmin: ktadd -k /home/ubuntu/ldap.keytab ldap/ldap-server.example.com

Entry for principal ldap/ldap-server.example.com with kvno 3, encryption type aes256-cts-hmac-shal-96 added to keytab WRFILE:/home/ubuntu/ldap.keytab.

Entry for principal ldap/ldap-server.example.com with kvno 3, encryption type aes128-cts-hmac-shal-96 added to keytab WRFILE:/home/ubuntu/ldap.keytab.

\mathbf{Note}

Notice how the kvno changed from 2 to 3 in the example above, when using ktadd a second time? This is the key version, and it basically invalidated the key with kvno 2 that was extracted before. Everytime a key is extracted with ktadd, its version is bumped and that invalidates the previous ones!

In this case, as long as the target location is writable, you don't even have to run kadmin with sudo.

Then use scp to transfer it to the target host:

\$ scp /home/ubuntu/ldap.keytab ldap-server.example.com:

And over there copy it to /etc/krb5.keytab, making sure it's mode 0600 and owned by root:root.
Kerberos - Secondary KDC

Once you have one Key Distribution Center (KDC) on your network, it is good practice to have a Secondary KDC in case the primary becomes unavailable. Also, if you have Kerberos clients that are in different networks (possibly separated by routers using NAT), it is wise to place a secondary KDC in each of those networks.

Note

The native replication mechanism explained here relies on a cronjob, and essentially dumps the DB on the primary and loads it back up on the secondary. You may want to take a look at using the *kldap* backend which can use the OpenLDAP replication mechanism. It is explained further below.

First, install the packages, and when asked for the Kerberos and Admin server names enter the name of the Primary KDC:

sudo apt install krb5-kdc krb5-admin-server

Once you have the packages installed, create the host principals for both KDCs. From a terminal prompt, enter:

```
$ kadmin -q "addprinc -randkey host/kdc01.example.com"
$ kadmin -q "addprinc -randkey host/kdc02.example.com"
```

Note

The kadmin command defaults to using a principal like *username/admin@EXAMPLE.COM*, where *username* is your current shell user. If you need to override that, use -p <principal-you-want>

Extract the key file for the kdc02 principal, which is this server we are on::

\$ sudo kadmin -p ubuntu/admin -q "ktadd host/kdc02.example.com"

Next, there needs to be a kpropd.acl file on each KDC that lists all KDCs for the Realm. For example, on both primary and secondary KDC, create /etc/krb5kdc/kpropd.acl:

host/kdc01.example.com@EXAMPLE.COM host/kdc02.example.com@EXAMPLE.COM

Note

It's customary to allow both KDCs because one may want to switch their roles if one goes bad. For such an eventuality, both are already listed here.

Create an empty database on the Secondary KDC:

```
$ sudo kdb5_util create -s
```

Now install kpropt daemon, which listens for connections from the kprop utility from the primary kdc:

\$ sudo apt install krb5-kpropd

The service will be running right after installation.

From a terminal on the *Primary KDC*, create a dump file of the principal database:

\$ sudo kdb5_util dump /var/lib/krb5kdc/dump

Still on the *Primary KDC*, extract its key:

\$ sudo kadmin.local -q "ktadd host/kdc01.example.com"

On the *Primary KDC*, run the kprop utility to push the database dump made before to the Secondary KDC:

\$ sudo kprop -r EXAMPLE.COM -f /var/lib/krb5kdc/dump kdc02.example.com
Database propagation to kdc02.example.com: SUCCEEDED

Note the *SUCCEEDED* message, which signals that the propagation worked. If there is an error message check /var/log/syslog on the secondary KDC for more information.

You may also want to create a cron job to periodically update the database on the Secondary KDC. For example, the following will push the database every hour:

m h dom mon dow command

0 * * * * root /usr/sbin/kdb5_util dump /var/lib/krb5kdc/dump && /usr/sbin/kprop -r EXAMPLE.COM f /var/lib/krb5kdc/dump kdc02.example.com

Finally, start the krb5-kdc daemon on the Secondary KDC:

\$ sudo systemctl start krb5-kdc.service

Note

The Secondary KDC does not run an admin server, since it's a read-only copy

From now on, you can specify both KDC servers in /etc/krb5.conf for the EXAMPLE.COM realm, in any host participating in this realm (including kdc01 and kdc02), but remember that there can only be one admin server and that's the one running on kdc01:

```
[realms]
EXAMPLE.COM = {
    kdc = kdc01.example.com
    kdc = kdc02.example.com
    admin_server = kdc01.example.com
}
```

The *Secondary KDC* should now be able to issue tickets for the Realm. You can test this by stopping the krb5-kdc daemon on the Primary KDC, then by using kinit to request a ticket. If all goes well you should receive a ticket from the Secondary KDC. Otherwise, check /var/log/syslog and /var/log/auth.log in the Secondary KDC.

Kerberos - Basic Workstation Authentication

This section covers configuring a Linux system as a Kerberos client. This will allow access to any kerberized services once a user has successfully logged into the system.

Note that Kerberos alone is not enough for a user to exist in a Linux system. Meaning, we cannot just point the system at a kerberos server and expect all the kerberos principals to be able to *login* on the linux system, simply because these users do not *exist* locally. Kerberos only provides authentication: it doesn't know about user groups, Linux uids and gids, home directories, etc. Normally another network source is used for this information, such as an LDAP or Windows server, and, in the old days, NIS was used for that as well.

Installation

If you have local users matching the principals in a Kerberos realm, and just want to switch the authentication from local to remote using Kerberos, you can follow this section. This is not a very usual scenario, but serves to highlight the separation between user authentication and user information (full name, uid, gid, home directory, groups, etc). If you just want to be able to grab tickets and use them, it's enough to install krb5-user and run kinit.

We are going to use **sssd** with a trick so that it will fetch the user information from the local system files, instead of a remote source which is the common case.

To install the packages enter the following in a terminal prompt:

```
$ sudo apt install krb5-user sssd-krb5
```

You will be prompted for the addresses of your KDCs and admin servers. If you have been following this chapter so far, the KDCs will be: kdc01.example.com kdc02.example.com (space separated)

And the admin server will be: kdc01.example.com. Remember that kdc02 is a read-only copy of the primary KDC, so it doesn't run an admin server.

Note

If you have added the appropriate SRV records to DNS, none of those prompts will need answering.

Configuration

If you missed the questions earlier, you can reconfigure the package to fill them in again: sudo dpkg-reconfigure krb5-config.

You can test the kerberos configuration by requesting a ticket using the kinit utility. For example:

\$ kinit ubuntu

Password for ubuntu@EXAMPLE.COM:

Note

kinit doesn't need for the principal to exist as a local user in the system. In fact, you can kinit any principal you want. If you don't specify one, then the tool will use the username of whoever is running kinit.

The only remaining configuration now is for sssd. Create the file /etc/sssd/sssd.conf with the following content:

```
[sssd]
config_file_version = 2
services = pam
domains = example.com
```

[pam]

```
[domain/example.com]
id_provider = proxy
proxy_lib_name = files
auth_provider = krb5
krb5_server = kdc01.example.com,kdc02.example.com
krb5_kpasswd = kdc01.example.com
krb5_realm = EXAMPLE.COM
```

The above configuration will use kerberos for *authentication* (auth_provider), but will use the local system users for user and group information (id_provider).

Adjust the permissions of the config file and start sssd:

```
$ sudo chown root:root /etc/sssd/sssd.conf
$ sudo chmod 0600 /etc/sssd/sssd.conf
$ sudo systemctl start sssd
```

Just by having installed sssd and its dependencies, PAM will already have been configured to use sssd, with a fallback to local user authentication. To try it out, if this is a workstation, simply switch users (in the GUI), or open a login terminal (CTRL-ALT-<number>), or spawn a login shell with sudo login, and try logging in using the name of a kerberos principal. Remember that this user must already exist on the local system:

```
$ sudo login
focal-krb5-client login: ubuntu
Password:
Welcome to Ubuntu Focal Fossa (development branch) (GNU/Linux 5.4.0-21-generic x86_64)
(...)
```

```
Last login: Thu Apr 9 21:23:50 UTC 2020 from 10.20.20.1 on pts/0
$ klist
Ticket cache: FILE:/tmp/krb5cc_1000_NlfnSX
Default principal: ubuntu@EXAMPLE.COM
```

```
Valid starting Expires Service principal
04/09/20 21:36:12 04/10/20 07:36:12 krbtgt/EXAMPLE.COM@EXAMPLE.COM
renew until 04/10/20 21:36:12
```

And you will have a Kerberos ticket already right after login.

Kerberos and LDAP

Kerberos supports a few database backends. The default one is what we have been using so far, called db2. The DB Types documentation shows all the options, one of which is LDAP.

There are several reasons why one would want to have the Kerberos principals stored in LDAP as opposed to a local on-disk database. There are also cases when it is not a good idea. Each site has to evaluate the pros and cons. Here are a few:

- the OpenLDAP replication is faster and more robust then the native Kerberos one, based on a cron job
- setting things up with the LDAP backend isn't exactly trivial and shouldn't be attempted by administrators without prior knowledge of OpenLDAP
- as highlighted in LDAP section of DB Types, since krb5kdc is single threaded there may be higher latency in servicing requests when using the OpenLDAP backend
- if you already have OpenLDAP setup for other things, like storing users and groups, adding the Kerberos attributes to the same mix might be beneficial and can provide a nice integrated story

This section covers configuring a primary and secondary kerberos server to use OpenLDAP for the principal database. Note that as of version 1.18, the KDC from MIT Kerberos does not support a primary KDC using a read-only consumer (secondary) LDAP server. What we have to consider here is that a Primary KDC is read-write, and it needs a read-write backend. The Secondaries can use both a read-write and read-only backend, because they are expected to be read-only. Therefore there are only some possible layouts we can use:

- 1. Simple case: Primary KDC connected to primary OpenLDAP, Secondary KDC connected to both Primary and Secondary OpenLDAP
- 2. Extended simple case: Multiple Primary KDCs connected to one Primary OpenLDAP, and multiple Secondary KDCs connected to Primary and Secondary OpenLDAP
- 3. OpenLDAP with multi-master replication: multiple primary KDCs connected to all primary OpenLDAP servers

We haven't covered OpenLDAP multi-master replication in this guide, so we will show the first case only. The second scenario is an extension: just add another primary KDC to the mix, talking to the same primary OpenLDAP server.

Configuring OpenLDAP

We are going to install the OpenLDAP server on the same host as the KDC, to simplify the communication between them. In such a setup, we can use the *ldapi:///* transport, which is via an unix socket, and don't need to setup SSL certificates to secure the communication between the Kerberos services and OpenLDAP. Note, however, that SSL is still needed for the OpenLDAP replication. See LDAP with TLS for details.

If you want to use an existing OpenLDAP server that you have somewhere else, that's of course also possible, but keep in mind that you should then use SSL for the communication between the KDC and this OpenLDAP server.

First, the necessary *schema* needs to be loaded on an OpenLDAP server that has network connectivity to the Primary and Secondary KDCs. The rest of this section assumes that you also have LDAP replication configured between at least two servers. For information on setting up OpenLDAP see OpenLDAP Server.

Note

cn=admin,dc=example,dc=com is a default admin user that is created during the installation of the slapd
package (the OpenLDAP server). The domain component will change for your server, so adjust accordingly.

• Install the necessary packages (it's assumed that OpenLDAP is already installed):

sudo apt install krb5-kdc-ldap krb5-admin-server

• Next, extract the kerberos.schema.gz file:

sudo cp /usr/share/doc/krb5-kdc-ldap/kerberos.schema.gz /etc/ldap/schema/ sudo gunzip /etc/ldap/schema/kerberos.schema.gz

• The *kerberos* schema needs to be added to the *cn=config* tree. This schema file needs to be converted to LDIF format before it can be added. For that we will use a helper tool, called schema2ldif, provided by the package of the same name which is available in the Universe archive:

sudo apt install schema2ldif

• To import the kerberos schema, run:

```
$ sudo ldap-schema-manager -i kerberos.schema
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
executing 'ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/ldap/schema/kerberos.ldif'
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
adding new entry "cn=kerberos,cn=schema,cn=config"
```

• With the new schema loaded, let's index an attribute often used in searches:

```
$ sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// <<EOF
dn: olcDatabase={1}mdb,cn=config
add: olcDbIndex
olcDbIndex: krbPrincipalName eq,pres,sub
EOF
```

modifying entry "olcDatabase={1}mdb,cn=config"

• Let's create LDAP entries for the Kerberos administrative entities that will contact the OpenLDAP server to perform operations. There are two:

- ldap_kdc_dn: needs to have read rights on the realm container, principal container and realm sub-trees.
 If disable_last_success and disable_lockout are not set, however, then ldap_kdc_dn needs write access to the kerberos container just like the admin dn below.
- ldap_kadmind_dn: needs to have read and write rights on the realm container, principal container and realm sub-trees

Here is the command to create these entities:

```
$ ldapadd -x -D cn=admin,dc=example,dc=com -W <<EOF
dn: uid=kdc-service,dc=example,dc=com
uid: kdc-service
objectClass: account
objectClass: simpleSecurityObject
userPassword: {CRYPT}x
description: Account used for the Kerberos KDC
dn: uid=kadmin-service,dc=example,dc=com
uid: kadmin-service
objectClass: account
objectClass: simpleSecurityObject
userPassword: {CRYPT}x
description: Account used for the Kerberos Admin server
EOF
```

Enter LDAP Password: adding new entry "uid=kdc-service,dc=example,dc=com"

adding new entry "uid=kadmin-service,dc=example,dc=com"

Now let's set a password for them. Note that first the tool asks for the password you want for the specified user dn, and then for the password of the cn=admin dn:

```
$ ldappasswd -x -D cn=admin,dc=example,dc=com -W -S uid=kdc-service,dc=example,dc=com
New password: <-- password you want for uid-kdc-service
Re-enter new password:
Enter LDAP Password: <-- password for the dn specified with the -D option</pre>
```

. . . .

Repeat for the ${\tt uid=kadmin-service}\ dn.$ These passwords will be needed later.

You can test these with ldapwhoami:

\$ ldapwhoami -x -D uid=kdc-service,dc=example,dc=com -W Enter LDAP Password: dn:uid=kdc-service,dc=example,dc=com

• Finally, update the Access Control Lists (ACL). These can be tricky, as it highly depends on what you have defined already. By default, the slapd package configures your database with the following ACLs:

```
olcAccess: {0}to attrs=userPassword by self write by anonymous auth by * none
olcAccess: {1}to attrs=shadowLastChange by self write by * read
olcAccess: {2}to * by * read
```

We need to insert new rules before the final to * by * read one, to control access to the Kerberos related entries and attributes:

```
$ sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// <<EOF
dn: olcDatabase={1}mdb,cn=config
add: olcAccess
olcAccess: {2}to attrs=krbPrincipalKey
by anonymous auth
by dn.exact="uid=kdc-service,dc=example,dc=com" read
by dn.exact="uid=kadmin-service,dc=example,dc=com" write
by self write
by * none
-
add: olcAccess
olcAccess: {3}to dn.subtree="cn=krbContainer,dc=example,dc=com"
by dn.exact="uid=kdc-service,dc=example,dc=com" read
```

by dn.exact="uid=kadmin-service,dc=example,dc=com" write

```
by * none
EOF
```

modifying entry "olcDatabase={1}mdb,cn=config"

This will make the existing {2} rule become {4}. Check with sudo slapcat -b cn=config (the output below was reformatted a bit for clarity):

```
olcAccess: {0}to attrs=userPassword
    by self write
    by anonymous auth
    by * none
olcAccess: {1}to attrs=shadowLastChange
    by self write
    by * read
olcAccess: {2}to attrs=krbPrincipalKey by anonymous auth
    by dn.exact="uid=kdc-service,dc=example,dc=com" read
    by dn.exact="uid=kadmin-service,dc=example,dc=com" write
    by self write
    by * none
olcAccess: {3}to dn.subtree="cn=krbContainer,dc=example,dc=com"
    by dn.exact="uid=kdc-service,dc=example,dc=com" read
    by dn.exact="uid=kadmin-service,dc=example,dc=com" write
    by * none
olcAccess: {4}to * by * read
```

That's it, your LDAP directory is now ready to serve as a Kerberos principal database.

Primary KDC Configuration (LDAP)

With OpenLDAP configured it is time to configure the KDC. In this example we are doing it in the same OpenLDAP server to take advantage of local unix socket communication.

• Reconfigure the krb5-config package if neededd to get a good starting point with /etc/krb5.conf:

```
sudo dpkg-reconfigure krb5-config
```

• Now edit /etc/krb5.conf adding the database_module option to the EXAMPLE.COM realm section:

```
[realms]
EXAMPLE.COM = {
    kdc = kdc01.example.com
    kdc = kdc02.example.com
    admin_server = kdc01.example.com
    default_domain = example.com
    database_module = openldap_ldapconf
}
```

Then also add these new sections:

```
[dbdefaults]
```

ldap_kerberos_container_dn = cn=krbContainer,dc=example,dc=com

```
[dbmodules]
```

```
openldap_ldapconf = {
    db_library = kldap

    # if either of these is false, then the ldap_kdc_dn needs to
    # have write access
    disable_last_success = true
    disable_lockout = true

    # this object needs to have read rights on
    # the realm container, principal container and realm sub-trees
    ldap_kdc_dn = "uid=kdc-service,dc=example,dc=com"

    # this object needs to have read and write rights on
    # the realm container, principal container and realm sub-trees
```

ldap_kadmind_dn = "uid=kadmin-service,dc=example,dc=com"

ldap_service_password_file = /etc/krb5kdc/service.keyfile ldap_servers = ldapi:/// ldap_conns_per_server = 5

```
}
```

• Next, use the kdb5_ldap_util utility to create the realm:

\$ sudo kdb5_ldap_util -D cn=admin,dc=example,dc=com create -subtrees dc=example,dc=com -r EXAMPLE.COM s -H ldapi:///
Password for "cn=admin,dc=example,dc=com":
Initializing database for realm 'EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:

• Create a stash of the password used to bind to the LDAP server. Run it once for each *ldap_kdc_dn* and *ldap_kadmin_dn*::

sudo kdb5_ldap_util -D cn=admin,dc=example,dc=com stashsrvpw -f /etc/krb5kdc/service.keyfile uid=kdcservice,dc=example,dc=com

sudo kdb5_ldap_util -D cn=admin,dc=example,dc=com stashsrvpw -f /etc/krb5kdc/service.keyfile uid=kadminservice,dc=example,dc=com

Note

The /etc/krb5kdc/service.keyfile file now contains clear text versions of the passwords used by the KDC to contact the LDAP server!

• Create a /etc/krb5kdc/kadm5.acl file for the admin server, if you haven't already:

*/admin@EXAMPLE.COM

• Start the Kerberos KDC and admin server:

sudo systemctl start krb5-kdc.service krb5-admin-server.service

You can now add Kerberos principals to the LDAP database, and they will be copied to any other LDAP servers configured for replication. To add a principal using the kadmin.local utility enter:

```
$ sudo kadmin.local
Authenticating as principal root/admin@EXAMPLE.COM with password.
kadmin.local: addprinc ubuntu
WARNING: no policy specified for ubuntu@EXAMPLE.COM; defaulting to no policy
Enter password for principal "ubuntu@EXAMPLE.COM":
Re-enter password for principal "ubuntu@EXAMPLE.COM":
Principal "ubuntu@EXAMPLE.COM" created.
kadmin.local:
```

The above will create an ubuntu *principal* with a *dn* of krbPrincipalName=ubuntu@EXAMPLE.COM, cn=EXAMPLE.COM, cn=krbContainer, do Let's say, however, that you already have an user in your directory, and it's in uid=testuser1,ou=People,dc=example,dc=com, how to add the kerberos attributes to it? You use the -x parameter to specify the location. For the *ldap_kadmin_dn* to be able to write to it, we first need to update the ACLs:

```
$ sudo ldapmodify -Q -Y EXTERNAL -H ldapi:/// <<EOF
dn: olcDatabase={1}mdb,cn=config
add: olcAccess
olcAccess: {4}to dn.subtree="ou=People,dc=example,dc=com"
by dn.exact="uid=kdc-service,dc=example,dc=com" read
by dn.exact="uid=kadmin-service,dc=example,dc=com" write
by * break
EOF
```

And now we can specify the new location:

```
$ sudo kadmin.local
Authenticating as principal root/admin@EXAMPLE.COM with password.
kadmin.local: addprinc -x dn=uid=testuser1,ou=People,dc=example,dc=com testuser1
WARNING: no policy specified for testuser1@EXAMPLE.COM; defaulting to no policy
```

Enter password for principal "testuser1@EXAMPLE.COM": Re-enter password for principal "testuser1@EXAMPLE.COM": Principal "testuser1@EXAMPLE.COM" created.

Since the specified dn already exists, kadmin.local will just add the required kerberos attributes to this existing entry. If it didn't exist, it would be created from scratch, with just the kerberos attributes, like what happened with the ubuntu example above, but in the specified location.

Note

The *ldap_kadmin_dn* DN (*uid=kadmin-service* in our example) does not have write access to the location specified by the -x parameter, you will get an Insufficient access error.

Both places are visible for kinit, since, when the realm was created with kdb5_ldap_util, the default value for the search scope and base were taken: subtree, and dc=example,dc=com.

Secondary KDC Configuration (LDAP)

The setup of the secondary KDC (and its OpenLDAP replica) is very similar. Once you have the OpenLDAP replication setup, repeat these steps on the secondary:

- install krb5-kdc-ldap, ldap-utils. Do not install krb5-admin-server.
- load the kerberos schema using schema2ldif
- add the index for *krbPrincipalName*
- add the ACLs
- configure *krb5.conf* in the same way, initially. If you want and if you configured SSL properly, you can add ldaps://kdc01.example.com to the ldap_servers list after ldapi:///, so that the Secondary KDC can have two LDAP backends at its disposal
- DO NOT run kdb5_ldap_util. There is no need to create the database since it's being replicated from the Primary
- copy over the following files from the Primary KDC and place them in the same location on the Secondary:
 - /etc/krb5kdc/stash
 - /etc/krb5kdc/service.keyfile
- start the KDC: sudo systemctl start krb5-kdc.service

Resources

- Configuring Kerberos with OpenLDAP back-end
- MIT Kerberos backend types

Network User Authentication with SSSD

SSSD stands for System Security Services Daemon and it's actually a collection of daemons that handle authentication, authorization, and user and group information from a variety of network sources. At its core it has support for:

- Active Directory
- LDAP
- Kerberos

SSSD provides PAM and NSS modules to integrate these remote sources into your system and allow remote users to login and be recognized as valid users, including group membership. To allow for disconnected operation, SSSD also can also cache this information, so that users can continue to login in the event of a network failure, or other problem of the same sort.

This guide will focus on the most common scenarios where SSSD is deployed.

References

• Upstream project: https://sssd.io/

SSSD and Active Directory

This section describes the use of sssd to authenticate user logins against an Active Directory via using sssd's "ad" provider. At the end, Active Directory users will be able to login on the host using their AD credentials. Group membership will also be maintained.

Prerequisites, Assumptions, and Requirements

- This guide does not explain Active Directory, how it works, how to set one up, or how to maintain it.
- This guide assumes that a working Active Directory domain is already configured and you have access to the credentials to join a machine to that domain.
- The domain controller is acting as an authoritative DNS server for the domain.
- The domain controller is the primary DNS resolver (check with systemd-resolve --status)
- System time is correct and in sync, maintained via a service like *chrony* or *ntp*
- The domain used in this example is adl.example.com .

Software Installation

Install the following packages:

sudo apt install sssd-ad sssd-tools realmd adcli

Join the domain

We will use the realm command, from the realmd package, to join the domain and create the sssd configuration.

Let's verify the domain is discoverable via DNS:

\$ sudo realm -v discover ad1.example.com * Resolving: ldap. tcp.ad1.example.com * Performing LDAP DSE lookup on: 10.51.0.5 * Successfully discovered: ad1.example.com ad1.example.com type: kerberos realm-name: AD1.EXAMPLE.COM domain-name: adl.example.com configured: no server-software: active-directory client-software: sssd required-package: sssd-tools required-package: sssd required-package: libnss-sss required-package: libpam-sss required-package: adcli required-package: samba-common-bin

This performs several checks and determines the best software stack to use with sssd. sssd can install the missing packages via *packagekit*, but we installed them already previously.

Now let's join the domain:

```
$ sudo realm join adl.example.com
Password for Administrator:
```

That was quite uneventful. If you want to see what it was doing, pass the -v option:

\$ sudo realm join -v ad1.example.com

- * Resolving: _ldap._tcp.ad1.example.com
- * Performing LDAP DSE lookup on: 10.51.0.5
- * Successfully discovered: ad1.example.com

Password for Administrator:

- * Unconditionally checking packages
- * Resolving required packages

* LANG=C /usr/sbin/adcli join --verbose --domain adl.example.com --domain-realm AD1.EXAMPLE.COM --domaincontroller 10.51.0.5 --login-type user --login-user Administrator --stdin-password

* Using domain name: adl.example.com

- * Calculated computer account name from fqdn: AD-CLIENT
- * Using domain realm: adl.example.com
- * Sending NetLogon ping to domain controller: 10.51.0.5
- * Received NetLogon info from: SERVER1.ad1.example.com

* Wrote out krb5.conf snippet to /var/cache/realmd/adcli-krb5-hUfTUg/krb5.d/adcli-krb5-conf-hv2kzi

```
* Authenticated as user: Administrator@AD1.EXAMPLE.COM
```

```
* Looked up short domain name: AD1
```

- * Looked up domain SID: S-1-5-21-2660147319-831819607-3409034899
- * Using fully qualified name: ad-client.ad1.example.com
- * Using domain name: adl.example.com
- * Using computer account name: AD-CLIENT
- * Using domain realm: adl.example.com
- * Calculated computer account name from fqdn: AD-CLIENT
- * Generated 120 character computer password
- * Using keytab: FILE:/etc/krb5.keytab
- * Found computer account for AD-CLIENT\$ at: CN=AD-CLIENT,CN=Computers,DC=ad1,DC=example,DC=com
- * Sending NetLogon ping to domain controller: 10.51.0.5
- * Received NetLogon info from: SERVER1.ad1.example.com
- * Set computer password
- * Retrieved kvno '3' for computer account in directory: CN=AD-CLIENT,CN=Computers,DC=ad1,DC=example,DC=com
- * Checking RestrictedKrbHost/ad-client.adl.example.com
- * Added RestrictedKrbHost/ad-client.ad1.example.com
- * Checking RestrictedKrbHost/AD-CLIENT
- * Added RestrictedKrbHost/AD-CLIENT
- * Checking host/ad-client.ad1.example.com
- * Added host/ad-client.ad1.example.com
- * Checking host/AD-CLIENT
- * Added host/AD-CLIENT
- * Discovered which keytab salt to use
- * Added the entries to the keytab: AD-CLIENT\$@AD1.EXAMPLE.COM: FILE:/etc/krb5.keytab
- * Added the entries to the keytab: host/AD-CLIENT@AD1.EXAMPLE.COM: FILE:/etc/krb5.keytab
- * Added the entries to the keytab: host/ad-client.ad1.example.com@AD1.EXAMPLE.COM: FILE:/etc/krb5.keytab
- * Added the entries to the keytab: RestrictedKrbHost/AD-CLIENT@AD1.EXAMPLE.COM: FILE:/etc/krb5.keytab
- * Added the entries to the keytab: RestrictedKrbHost/ad-client.ad1.example.com@AD1.EXAMPLE.COM: FILE:/etc/krb5.keytab
- * /usr/sbin/update-rc.d sssd enable
- * /usr/sbin/service sssd restart
- * Successfully enrolled machine in realm

By default, realm will use the Administrator account of the domain to request the join. If you need to use another account, pass it to the tool with the -U option.

Another popular way of joining a domain is using an *OTP*, or *One Time Password*, token. For that, use the --one-time-password option.

SSSD Configuration

The *realm* tool already took care of creating an sssd configuration, adding the pam and nss modules, and starting the necessary services.

Let's take a look at /etc/sssd/sssd.conf:

```
[sssd]
domains = adl.example.com
config file version = 2
services = nss, pam
[domain/ad1.example.com]
default_shell = /bin/bash
krb5_store_password_if_offline = True
cache_credentials = True
krb5_realm = AD1.EXAMPLE.COM
realmd_tags = manages-system joined-with-adcli
id_provider = ad
fallback homedir = /home/%u@%d
ad domain = adl.example.com
use_fully_qualified_names = True
ldap_id_mapping = True
access_provider = ad
     Note
```

Something very important to remember is that this file must have permissions 0600 and ownership *root:root*, or else sssd won't start!

Let's highlight a few things from this config:

- cache_credentials: this allows logins when the AD server is unreachable
- home directory: it's by default /home/<user>@<domain>. For example, the AD user john will have a home directory of /home/john@ad1.example.com
- use_fully_qualified_names: users will be of the form user@domain, not just user. This should only be changed if you are certain no other domains will ever join the AD forest, via one of the several possible trust relationships

Automatic home directory creation

What the realm tool didn't do for us is setup pam_mkhomedir, so that network users can get a home directory when they login. This remaining step can be done by running the following command:

```
sudo pam-auth-update --enable mkhomedir
```

Checks

You should now be able to fetch information about AD users. In this example, John Smith is an AD user:

```
$ getent passwd john@adl.example.com
john@adl.example.com:*:1725801106:1725800513:John Smith:/home/john@adl.example.com:/bin/bash
```

Let's see his groups:

```
$ groups john@adl.example.com
john@adl.example.com : domain users@adl.example.com engineering@adl.example.com
```

Note

If you just changed the group membership of a user, it may be a while before sssd notices due to caching.

Finally, how about we try a login:

```
$ sudo login
ad-client login: john@adl.example.com
Password:
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-24-generic x86_64)
...
Creating directory '/home/john@adl.example.com'.
john@adl.example.com@ad-client:~$
```

Notice how the home directory was automatically created.

You can also use ssh, but note that the command will look a bit funny because of the multiple @ signs:

```
$ ssh john@adl.example.com@10.51.0.11
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-24-generic x86_64)
(...)
Last login: Thu Apr 16 21:22:55 2020
john@adl.example.com@ad-client:~$
```

Note

In the ssh example, public key authentication was used, so no password was required. Remember that ssh password authentication is by default disabled in /etc/ssh/sshd_config.

Kerberos Tickets

If you install krb5-user, your AD users will also get a kerberos ticket upon logging in:

```
john@adl.example.com@ad-client:~$ klist
Ticket cache: FILE:/tmp/krb5cc_1725801106_9UxVIz
Default principal: john@AD1.EXAMPLE.COM
```

```
Valid starting Expires Service principal
04/16/20 21:32:12 04/17/20 07:32:12 krbtgt/AD1.EXAMPLE.COM@AD1.EXAMPLE.COM
renew until 04/17/20 21:32:12
```

Note

 realm also configured /etc/krb5.conf for you, so there should be no further configuration prompts when installing <code>krb5-user</code>

Let's test with *smbclient* using kerberos authentication to list he shares of the domain controller:

john@ad1.example.com@ad-client:~\$ smbclient -k -L server1.ad1.example.com

9	Sharename		Туре	Commer	nt	
					· -	
4	ADMIN\$		Disk	Remote	e Admin	
(C\$		Disk	Defaul	lt share	е
]	EPC\$		IPC	Remote	e IPC	
1	NETLOGON		Disk	Logon	server	share
9	SYSV0L		Disk	Logon	server	share
SMB1	disabled	no	o workgroup	availa	able	

Notice how we now have a ticket for the *cifs* service, which was used for the share list above:

```
john@adl.example.com@ad-client:~$ klist
Ticket cache: FILE:/tmp/krb5cc_1725801106_9UxVIz
Default principal: john@AD1.EXAMPLE.COM
```

```
Valid starting Expires Service principal
04/16/20 21:32:12 04/17/20 07:32:12 krbtgt/AD1.EXAMPLE.COM@AD1.EXAMPLE.COM
renew until 04/17/20 21:32:12
04/16/20 21:32:21 04/17/20 07:32:12 cifs/server1.ad1.example.com@AD1.EXAMPLE.COM
```

Desktop Ubuntu Authentication

The desktop login only shows local users in the list to pick from, and that's on purpose.

To login with an Active Directory user for the first time, follow these steps:

• click on the "Not listed?" option:



click-not-listed 1024 $\times768$ 14.5 KB

• type in the login name followed by the password:

Apr 17: 14:12	₽ - 4 0 -
C Johngiud'Lexample.com	
ubustu@	
uduntu	

type-in-username 1024×768 16.3 KB

• the next time you login, the AD user will be listed as if it was a local user:



next-time 1024 $\times768$ 16.1 KB

Known Issues

When logging in on a system joined with an Active Directory domain, sssd (the package responsible for this integration) will try to apply Group Policies by default. There are cases where if a specific policy is missing, the login will be denied.

This is being tracked in bug #1934997. Until the fix becomes available, please see comment #5 in that bug report for existing workarounds.

Resources

- GitHub SSSD Project
- Active Directory DNS Zone Entries

SSSD and LDAP

SSSD can also use LDAP for authentication, authorization, and user/group information. In this section we will configure a host to authenticate users from an OpenLDAP directory.

Prerequisites, Assumptions, and Requirements

For this setup, we need:

- an existing OpenLDAP server with SSL enabled and using the RFC2307 schema for users and groups
- a client host where we will install the necessary tools and login as a user from the LDAP server

Software Installation

Install the following packages:

sudo apt install sssd-ldap ldap-utils

SSSD Configuration

Create the /etc/sssd/sssd.conf configuration file, with permissions 0600 and ownership root:root, and this content:

```
[sssd]
config_file_version = 2
domains = example.com
```

```
[domain/example.com]
id_provider = ldap
auth_provider = ldap
ldap_uri = ldap://ldap01.example.com
cache_credentials = True
ldap_search_base = dc=example,dc=com
```

Make sure to start the *sssd* service:

sudo systemctl start sssd.service

Note

sssd will use *START_TLS* by default for authentication requests against the LDAP server (the *auth_provider*), but not for the *id_provider*. If you want to also enable *START_TLS* for the *id_provider*, specify ldap_id_use_start_tls = true.

Automatic home directory creation

To enable automatic home directory creation, run the following command:

sudo pam-auth-update --enable mkhomedir

Check SSL setup on the client

The client must be able to use $START_TLS$ when connecting to the LDAP server, with full certificate checking. This means:

- the client host knows and trusts the CA that signed the LDAP server certificate
- the server certificate was issued for the correct host (ldap01.example.com in this guide)
- the time is correct on all hosts performing the TLS connection
- and, of course, that neither certificate (CA or server's) expired

If using a custom CA, an easy way to have a host trust it is to place it in /usr/local/share/ca-certificates/ with a .crt extension and run sudo update-ca-certificates.

Alternatively, you can edit /etc/ldap/ldap.conf and point TLS_CACERT to the CA public key file.

Note

You may have to restart sssd after these changes: sudo systemctl restart sssd

Once that is all done, check that you can connect to the LDAP server using verified SSL connections:

\$ ldapwhoami -x -ZZ -H ldap://ldap01.example.com
anonymous

and for ldaps (if enabled in /etc/default/slapd):

\$ ldapwhoami -x -H ldaps://ldap01.example.com

The -ZZ parameter tells the tool to use *START_TLS*, and that it must not fail. If you have LDAP logging enabled on the server, it will show something like this:

```
slapd[779]: conn=1032 op=0 STARTTLS
slapd[779]: conn=1032 op=0 RESULT oid= err=0 text=
slapd[779]: conn=1032 fd=15 TLS established tls_ssf=256 ssf=256
slapd[779]: conn=1032 op=1 BIND dn="" method=128
slapd[779]: conn=1032 op=1 RESULT tag=97 err=0 text=
```

slapd[779]: conn=1032 op=2 EXT oid=1.3.6.1.4.1.4203.1.11.3
slapd[779]: conn=1032 op=2 WH0AMI
slapd[779]: conn=1032 op=2 RESULT oid= err=0 text=

 $START_TLS$ with err=0 and TLS established is what we want to see there, and, of course, the WHOAMI extended operation.

Final verification

In this example, the LDAP server has the following user and group entry we are going to use for testing:

dn: uid=john,ou=People,dc=example,dc=com uid: john objectClass: inetOrgPerson objectClass: posixAccount cn: John Smith sn: Smith givenName: John mail: john@example.com userPassword: johnsecret uidNumber: 10001 gidNumber: 10001 loginShell: /bin/bash homeDirectory: /home/john dn: cn=john,ou=Group,dc=example,dc=com cn: john objectClass: posixGroup gidNumber: 10001 memberUid: john dn: cn=Engineering,ou=Group,dc=example,dc=com cn: Engineering objectClass: posixGroup gidNumber: 10100 memberUid: john The user *john* should be known to the system: ubuntu@ldap-client:~\$ getent passwd john john:*:10001:10001:John Smith:/home/john:/bin/bash ubuntu@ldap-client:~\$ id john uid=10001(john) gid=10001(john) groups=10001(john),10100(Engineering) And we should be able to authenticate as *john*: ubuntu@ldap-client:~\$ sudo login ldap-client login: john Password: Welcome to Ubuntu Focal Fossa (development branch) (GNU/Linux 5.4.0-24-generic x86 64) (...) Creating directory '/home/john'. john@ldap-client:~\$

SSSD, LDAP and Kerberos

Finally, we can mix it all together in a setup that is very similar to Active Directory in terms of the technologies used: use LDAP for users and groups, and Kerberos for authentication.

Prerequisites, Assumptions, and Requirements

For this setup, we will need:

• an existing OpenLDAP server using the RFC2307 schema for users and groups. SSL support is recommended, but not strictly necessary because authentication in this setup is being done via Kerberos, and not LDAP.

- a Kerberos server. It doesn't have to be using the OpenLDAP backend
- a client host where we will install and configure SSSD

Software Installation

On the client host, install the following packages:

sudo apt install sssd-ldap sssd-krb5 ldap-utils krb5-user

You may be asked about the default Kerberos realm. For this guide, we are using EXAMPLE.COM.

At this point, you should already be able to obtain tickets from your Kerberos server, assuming DNS records point at it like explained elsewhere in this guide:

\$ kinit ubuntu
Password for ubuntu@EXAMPLE.COM:

ubuntu@ldap-krb-client:~\$ klist Ticket cache: FILE:/tmp/krb5cc_1000 Default principal: ubuntu@EXAMPLE.COM

Valid starting Expires Service principal 04/17/20 19:51:06 04/18/20 05:51:06 krbtgt/EXAMPLE.COM@EXAMPLE.COM renew until 04/18/20 19:51:05

But we want to be able to login as an LDAP user, authenticated via Kerberos. Let's continue with the configuration.

SSSD Configuration

Create the /etc/sssd/sssd.conf configuration file, with permissions 0600 and ownership root:root, and this content:

```
[sssd]
config_file_version = 2
domains = example.com
```

```
[domain/example.com]
id_provider = ldap
ldap_uri = ldap://ldap01.example.com
ldap_search_base = dc=example,dc=com
auth_provider = krb5
krb5_server = kdc01.example.com,kdc02.example.com
krb5_kpasswd = kdc01.example.com
krb5_realm = EXAMPLE.COM
cache_credentials = True
```

This example uses two KDCs, which made it necessary to also specify the $krb5_kpasswd$ server because the second KDC is a replica and is not running the admin server.

Start the *sssd* service:

sudo systemctl start sssd.service

Automatic home directory creation

To enable automatic home directory creation, run the following command:

sudo pam-auth-update --enable mkhomedir

Final verification

In this example, the LDAP server has the following user and group entry we are going to use for testing:

```
dn: uid=john,ou=People,dc=example,dc=com
uid: john
objectClass: inetOrgPerson
objectClass: posixAccount
cn: John Smith
sn: Smith
givenName: John
```

mail: john@example.com uidNumber: 10001 aidNumber: 10001 loginShell: /bin/bash homeDirectory: /home/john dn: cn=john,ou=Group,dc=example,dc=com cn: john objectClass: posixGroup gidNumber: 10001 memberUid: john dn: cn=Engineering,ou=Group,dc=example,dc=com cn: Engineering objectClass: posixGroup gidNumber: 10100 memberUid: john Note how the *john* user has no *userPassword* attribute. The user *john* should be known to the system: ubuntu@ldap-client:~\$ getent passwd john john:*:10001:10001:John Smith:/home/john:/bin/bash ubuntu@ldap-client:~\$ id john uid=10001(john) gid=10001(john) groups=10001(john),10100(Engineering) Let's try a login as this user: ubuntu@ldap-krb-client:~\$ sudo login ldap-krb-client login: john Password: Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-24-generic x86 64) (...)Creating directory '/home/john'. john@ldap-krb-client:~\$ klist Ticket cache: FILE:/tmp/krb5cc_10001_B0rxWr Default principal: john@EXAMPLE.COM Valid starting Expires Service principal 04/17/20 20:29:50 04/18/20 06:29:50 krbtgt/EXAMPLE.COM@EXAMPLE.COM renew until 04/18/20 20:29:50

john@ldap-krb-client:~\$

We logged in using the kerberos password, and user/group information from the LDAP server.

SSSD and KDC spoofing

When using SSSD to manage kerberos logins on a Linux host, there is an attack scenario you should be aware of: KDC spoofing.

The objective of the attacker is to login on a workstation that is using Kerberos authentication. Let's say he knows john is a valid user on that machine.

The attacker first deploys a rogue KDC server in the network, and creates the john principal there with a password of his choosing. What he has to do now is to have his rogue KDC respond to the login request from the workstation, before (or instead of) the real KDC. If the workstation isn't authenticating the KDC, it will accept the reply from the rogue server and let john in.

There is a configuration parameter that can be set to protect the workstation from this attack. It will have SSSD authenticate the KDC, and block the login if the KDC cannot be verified. This option is called krb5_validate, and it's false by default.

To enable it, edit ${\tt /etc/sssd/sssd.conf}$ and add this line to the domain section:

[sssd]
config_file_version = 2

[domain/example.com] id_provider = ldap ... krb5_validate = True

The second step is to create a host principal on the KDC for this workstation. This is how the KDC's authenticity is verified. It's like a "machine account", with a shared secret that the attacker cannot control and replicate in his rogue KDC...The host principal has the format host/<fqdn>@REALM.

After the host principal is created, its keytab needs to be stored on the workstation. This two step process can be easily done on the workstation itself via kadmin (not kadmin.local) to contact the KDC remotely:

```
$ sudo kadmin -p ubuntu/admin
```

```
kadmin: addprinc -randkey host/ldap-krb-client.example.com@EXAMPLE.COM
WARNING: no policy specified for host/ldap-krb-client.example.com@EXAMPLE.COM; defaulting to no policy
Principal "host/ldap-krb-client.example.com@EXAMPLE.COM" created.
```

kadmin: ktadd -k /etc/krb5.keytab host/ldap-krb-client.example.com Entry for principal host/ldap-krb-client.example.com with kvno 6, encryption type aes256-cts-hmac-shal-96 added to keytab WRFILE:/etc/krb5.keytab. Entry for principal host/ldap-krb-client.example.com with kvno 6, encryption type aes128-cts-hmac-shal-

Then exit the tool and make sure the permissions on the keytab file are tight:

```
sudo chmod 0600 /etc/krb5.keytab
sudo chown root:root /etc/krb5.keytab
```

96 added to keytab WRFILE:/etc/krb5.keytab.

You can also do it on the KDC itself using kadmin.local, but you will have to store the keytab temporarily in another file and securely copy it over to the workstation.

Once these steps are complete, you can restart sssd on the workstation and perform the login. If the rogue KDC picks the attempt up and replies, it will fail the host verification. With debugging we can see that happening on the workstation:

=> /var/log/sssd/krb5_child.log <==
(Mon Apr 20 19:43:58 2020) [[sssd[krb5_child[2102]]]] [validate_tgt] (0x0020): TGT failed verification using key for [host
krb-client.example.com@EXAMPLE.COM].
(Mon Apr 20 19:43:58 2020) [[sssd[krb5_child[2102]]]] [get_and_save_tgt] (0x0020): 1741: [-1765328377][Server host/ldap-</pre>

krb-client.example.com@EXAMPLE.COM not found in Kerberos database]

And the login is denied. If the real KDC picks it up, however, the host verification succeeds:

=> /var/log/sssd/krb5_child.log <==
 (Mon Apr 20 19:46:22 2020) [[sssd[krb5_child[2268]]]] [validate_tgt] (0x0400): TGT verified using key for [host/ldapkrb-client.example.com@EXAMPLE.COM].

And the login is accepted.

Debugging and troubleshooting

Here are some tips to help troubleshoot sssd.

debug_level

The debug level of sssd can be changed on-the-fly via sssctl, from the sssd-tools package:

```
sudo apt install sssd-tools
sssctl debug-level <new-level>
```

Or change add it to the config file and restart sssd:

```
[sssd]
config_file_version = 2
domains = example.com
```

```
[domain/example.com]
debug_level = 6
```

• • •

Either will yield more logs in /var/log/sssd/*.log and can help identify what is going on. The sssctl approach has the clear advantage of not having to restart the service.

Caching

Caching is useful to speed things up, but it can get in the way big time when troubleshooting. It's useful to be able to remove the cache while chasing down a problem. This can also be done with the <code>sssctl</code> tool from the <code>sssd-tools</code> package.

You can either remove the whole cache:

```
# sssctl cache-remove
Creating backup of local data...
SSSD backup of local data already exists, override? (yes/no) [no] yes
Removing cache files...
SSSD= needs to be running. Start SSSD now? (yes/no) [yes] yes
```

Or just one element:

sssctl cache-expire -u john

Or expire everything:

sssctl cache-expire -E

DNS

Kerberos is quite sensitive to DNS issues. If you suspect something related to DNS, here are two suggestions:

FQDN hostname

Make sure hostname -f returns a fully qualified domain name. Set it in /etc/hostname if necessary, and use sudo resolvectl set-hostname <fqdn> to set it at runtime.

Reverse name lookup

You can try disabling a default reverse name lookup that the krb5 libraries do, by editing (or creating) /etc/krb5.conf and setting rdns = false in the [libdefaults] section:

```
[libdefaults]
rdns = false
```

Introduction

WireGuard is a simple, fast and modern VPN implementation, widely deployed and cross-platform.

VPNs have traditionally been hard to understand, configure and deploy. WireGuard removed most of that complexity by focusing on its single task, and leaving out things like key distribution and pushed configurations. You get a network interface which encrypts and verifies the traffic, and the remaining tasks like setting up addresses, routing, etc, are left to the usual system tools like ip-route(8) and ip-address(8).

Setting up the cryptographic keys is very much similar to configuring ssh for key based authentication: each side of the connection has its own private and public key, and the peers' public key, and this is enough to start encrypting and verifying the exchanged traffic.

For more details on how WireGuard works, and information on its availability in other platforms, please see the references section.

WireGuard Concepts

It helps to think of WireGuard primarly as a network interface, like any other. It will have the usual attributes, like IP address, CIDR, and there will be some routing associated with it. But it also has WireGuard specific attributes, which handle the VPN part of things.

All of this can be configured via different tools. WireGuard itself ships its own tools in the userspace package wireguard-tools: wg(8) and wg-quick(8). But these are not strictly needed: any userspace with the right privileges and kernel calls can configure a WireGuard interface. For example, systemd-networkd and network-manager can do it on their own, without the WireGuad userspace utilities.

Important attributes of a WireGuard interface are:

- *private key*: together with the corresponding public key, they are used to authenticate and encrypt data. This is generated with the wg genkey command.
- *listen port*: the UDP port that WireGuard will be listening to for incoming traffic.
- List of *peers*, each one with:
 - *public key*: the public counterpart of the private key. Generated from the private key of that peer, using the wg pubkey command.
 - *endpoint*: where to send the encrypted traffic to. This is optional, but at least one of the corresponding peers must have it to bootstrap the connection.
 - allowed IPs: list of inner tunnel destination networks or addresses for this peer when sending traffic, or, when receiving traffic, which source networks or addresses are allowed to send traffic to us.

NOTE

Cryptography is not simple. When we say that, for example, a private key is used to decrypt or sign traffic, and a public key is used to encrypt or verify the authenticity of traffic, this is a simplification and is hiding a lot of important details. WireGuard has a detailed explanation of its protocols and cryptography handling in their website, at https://www.wireguard.com/protocol/

These parameters can be set with the low-level wg(8) tool, directly via the command line or with a configuration file. This tool, however, doesn't handle the non-WireGuard settings of the interface. It won't assign an IP address to it, for example, nor setup routing. For this reason, it's more common to use wg-quick(8).

wg-quick(8) will handle the lifecycle of the WireGuard interface. It can bring it up or down, setup routing, execute arbitrary commands before or after the interface is up, and more. It augments the configuration file that wg(8) can use, with its own extra settings, which is important to keep in mind when feeding that file to wg(8), as it will contain settings wg(8) knows nothing about.

The wg-quick(8) configuration file can have an arbitrary name, and can even be placed anywhere on the system, but the best practice is:

- Place the file in /etc/wireguard.
- Name it after the interface it controls.

For example, a file called /etc/wireguard/wg0.conf will have the needed configurations setting for a WireGuard network interface called wg0. By following this practice, you get the benefit of being able to call wg-quick with just the interface name:

\$ sudo wg-quick up wg0

And that will bring the wg0 interface up, give it an IP address, setup routing, and configure the WireGuard specific parameters for it to work. This interface is usually called wg0, but can have any valid network interface name, like office (it doesn't need an index number after the name), home1, etc. It can help to give it a meaningful name if you plan to connect to multiple peers.

Let's go over an example of such a configuration file:

```
[Interface]
PrivateKey = eJdSgoS7BZ/uWkuSREN+vhCJPPr3M3UlB3v1Su/amWk=
ListenPort = 51000
Address = 10.10.11.10/24
[Peer]
# office
PublicKey = xeWmdxiLjgebpcItF1ouRo0ntrgFekquRJZQ0+vsQVs=
Endpoint = wg.example.com:51000 # fake endpoint, just an example
AllowedIPs = 10.10.11.0/24, 10.10.10.0/24
```

In the [Interface] section:

- Address: this is the IP address, and CIDR, that the WireGuard interface will be setup with.
- ListenPort: the UDP port WireGuard will use for traffic (listening and sending).
- **PrivateKey**: the secret key used to decrypt traffic destined to this interface.

The *peers* list, each one in its own [Peer] section (example above has just one), comes next:

- PublicKey: the key that will be used to encrypt traffic to this peer.
- Endpoint: where to send encrypted traffic to.
- AllowedIPs: when sending traffic, this is the list of target addresses that identify this peer. When receiving traffic, it's the list of addresses that are allowed to be the source of the traffic.

To generate the keypairs for each peer, the wg(8) command is used:

\$ umask 077
\$ wg genkey > wg0.key
\$ wg pubkey < wg0.key > wg0.pub

And then the contents of wg0.key and wg0.pub can be used in the configuration file.

This is what it looks like when this interface is brought up by wg-quick(8):

\$ sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.10.11.10/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] ip -4 route add 10.10.10.0/24 dev wg0

This is what wg-quick(8) did for us:

- Created the WireGuard wg0 interface.
- Configured it with the data from the configuration file.
- Added the IP/CIDR from the Address field to the wg0 interface.
- Calculated a proper MTU (which can be overridden in the config if needed)
- Added a route for AllowedIPs.

Note that in this example AllowedIPs is a list of two CIDR network blocks, but wg-quick(8) only added a route for 10.10.0/24 and skipped 10.10.11.0/24. That's because the Address was already specified as a /24 one. Had we specified the address as 10.10.11.10/32 instead, then wg-quick(8) would have added a route for 10.10.11.0/24 explicitly.

To better understand how AllowedIPs work, let's go through a quick example.

Let's say this system wants to send traffic to 10.10.201/24. There is a route for it which says to use the wg0 interface for that:

Since wg0 is a WireGuard interface, it will consult its configuration to see if any peer has that target address in the AllowedIPs list. Turns out one peer has it, in which case the traffic will:

a) Be authenticated as us, and encrypted for that peer.

b) Sent away via the configured Endpoint.

Now let's picture the reverse. This system received traffic on the ListenPort UDP port. If it can be decrypted, and verified as having come from one of the listed peers using its respective public key, and if the source IP matches the corresponding AllowedIPs list, then the traffic is accepted.

What if there is no Endpoint? Well, to bootstrap the VPN, at least one of the peers must have an Endpoint, or else it won't know where to send the traffic to, and you will get an error saying "Destination address required" (see the troubleshooting section for details).

But once the peers know each other, the one that didn't have an Endpoint setting in the interface will remember where the traffic came from, and use that address as the current endpoint. This has a very nice side effect of automatically tracking the so called "road warrior" peer, which keeps changing its IP. That is very common with laptops that keep being suspended and awakened in a new network, and then try to establish the VPN again from that new address.

Peers

You will notice that the term "peers" is used preferably to "server" or "client". Other terms used in some VPN documentation are "left" and "right", which is already starting to convey that the difference between a "server" and a "client" is a bit blurry. It only matters, if at all, at the start of the traffic exchange: who sends the first packet of data. In that sense, "servers" expect to sit idle and wait for connections to be initiated to them, and "clients" are the initiators. For example, a laptop on a public cafe initiating a connection to the company VPN peer. The laptop needs to know the address of that peer, because it's initiating the exchange. But the "server" doesn't need to know the IP of the laptop beforehand.

On a site to site VPN, however, when two separate networks are connected through the tunnel, who is the server, and who is the client? Both, so it's best to call them "peers" instead.

Putting it all together

Key takeaways from this introduction:

- Each peer participating in the WireGuard VPN has a private key and a public key.
- AllowedIPs is used as a routing key when sending traffic, and as an ACL when receiving traffic.
- To establish a VPN with a remote peer, you need its public key. Likewise, the remote peer will need your public key.
- At least one of the peers needs an Endpoint configured in order to be able to initiate the VPN.

To help better understand these and other concepts, we will create some WireGuard VPNs in the next sections, illustrating some common setups.

NOTE

Throghout this guide, we will sometimes mention a VPN "connection". This is technically false, as Wire-Guard uses UDP and there is no persistent connection. The term is used just to facilitate understanding, and means that the peers in the examples know each other and have completed a handshake already.

References

- See the WireGuard website for more detailed information.
- The WireGuard Quickstart has a good introduction and demo.
- wg(8) and wg-quick(8) manual pages.
- Detailed explanation of the algorithms used by WireGuard.

Peer to site - Introduction

To help understand the WireGuard concepts, we will show some practical setups that hopefully match many scenarios out there.

This is probably the most common setup for a VPN: connecting a single system to a remote site, and getting access to the remote network "as if you were there".

Where to place the remote WireGuard endpoint in the network will vary a lot depending on the topology. It can be in a firewall box, the router itself, or some random system in the middle of the network.

Here we will cover a simpler case more resembling what a home network could be like:

public internet



This diagram represents a typical simple home network setup. You have a router/modem, usually provided by the ISP (Internet Service Provider), and some internal devices like a Raspberry PI perhaps, a NAS (Network Attached Storage), and some other device.

There are basically two approaches that can be taken here: install WireGuard on the router, or on another system in the home network. We will discuss both in the following sections.

Note that in this scenario the "fixed" side, the home network, normally won't have a WireGuard Endpoint configured, as the peer is typically "on the road" and will have a dynamic IP address.

WireGuard on the router

In this diagram, we are depicting a home network with some devices and a router where we can install WireGuard.



Of course, this setup is only possible if you can install software on the router. Most of the time, when it's provided by your ISP, you can't. But some ISPs allow their device to be put in a bridge mode, in which case you can use your own device (a computer, or a Raspberry PI, or something else) as the routing device.

Since the router is the default gateway of the network already, this means you can create a whole new network for your VPN users. You also won't have to create any (D)NAT rules since the router is directly reachable from the Internet.

Let's define some addresses, networks, and terms used in this guide:

- *laptop in coffee shop*: just your normal user at a coffee shop, using the provided WiFi access to connect to their home network. This will be one of our *peers* in the VPN setup.
- home0: this will be the WireGuard interface on the laptop. It's called home0 to convey the information that it is used to connect to the *home* network.
- *router*: the existing router at the home network. It has a public interface ppp0 that has a routable but dynamic IPv4 address (not CGNAT), and an internal interface at 10.10.10.1/24 which is the default gateway for the home network.
- *home network*: the existing home network, 10.10.0/24 in this example, with existing devices that the user wishes to access remotely over the WireGuard VPN.
- 10.10.11.0/24: the WireGuard VPN network. This is a whole new network that was created just for the VPN users.
- wg0 on the *router*: this is the WireGuard interface that we will bring up on the router, at the 10.10.11.1/24 address. It is the gateway for the 10.10.11.0/24 VPN network.

With this topology, if, say, the NAS wants to send traffic to 10.10.11.2/24, it will send it to the default gateway (since the NAS has no specific route to 10.10.11.0/24), and the gateway will know how to send it to 10.10.11.2/24 because it has the wg0 interface on that network.

Configuration

First, we need to create keys for the peers of this setup. We need one pair of keys for the laptop, and another for the home router:

```
$ umask 077
$ wg genkey > laptop-private.key
$ wg pubkey < laptop-private.key > laptop-public.key
$ wg genkey > router-private.key
$ wg pubkey < router-private.key > router-public.key
```

Let's create the router wg0 interface configuration file. The file will be /etc/wireguard/wg0.conf and have these contents:

```
[Interface]
PrivateKey = <contents-of-router-private.key>
ListenPort = 51000
Address = 10.10.11.1/24
```

```
[Peer]
PublicKey = <contents-of-laptop-public.key>
AllowedIPs = 10.10.11.2
```

There is no Endpoint configured for the laptop peer, because we don't know what IP address it will have beforehand, nor will that IP address be always the same. This laptop can be connecting from a coffee shop free wifi, or an airport lounge, or a friend's house.

Not having an endpoint here also means that the home network side will never be able to *initiate* the VPN connection. It will sit and wait, and can only *respond* to VPN handshake requests, at which time it will learn the endpoint from the peer and use that until it changes (the peer reconnects from a different site) or times out.

IMPORTANT

This configuration file contains a secret: **PrivateKey**. Make sure to adjust its permissions accordingly, like:

sudo chmod 0600 /etc/wireguard/wg0.conf sudo chown root: /etc/wireguard/wg0.conf

When activated, this will bring up a wg0 interface with the address 10.10.11.1/24, listening on port 51000/udp, and add a route for the 10.10.11.0/24 network using that interface.

The *Peer* section is identifying a peer via its public key, and listing who can connect from that peer. This AllowedIPs setting has two meanings:

- When sending packets, the AllowedIPs list serves as a routing table, indicating that this peer's public key should be used to encrypt the traffic.
- When receiving packets, AllowedIPs behaves like an access control list. After decryption, the traffic is only allowed if it matches the list.

Finally, the ListenPort parameter specifies the *UDP* port on which WireGuard will listen for traffic. This port will have to be allowed in the firewall rules of the router. There is no default nor a standard port for WireGuard, so you can pick any value you prefer.

Now let's create a similar configuration on the other peer, the laptop. Here the interface is called home0, so the configuration file is /etc/wireguard/home0.conf:

```
[Interface]
PrivateKey = <contents-of-laptop-private.key>
ListenPort = 51000
Address = 10.10.11.2/24
```

[Peer]
PublicKey = <contents-of-router-public.key>
Endpoint = <home-ppp0-IP-or-hostname>:51000
AllowedIPs = 10.10.11.0/24,10.10.10.0/24

IMPORTANT

Like before, this configuration file contains a secret: **PrivateKey**. Make sure to adjust its permissions accordingly, like:

sudo chmod 0600 /etc/wireguard/home0.conf sudo chown root: /etc/wireguard/home0.conf

We have given this laptop the 10.10.11.2/24 address. It could have been any valid address in the 10.10.11.0/24 network, as long as it doesn't collide with an existing one, and is allowed in the router's peer's *AllowedIPs* list.

NOTE

You may have noticed by now that address allocation is manual, and not via something like DHCP. Keep tabs on it!

In the [Peer] stanza for the laptop we have:

- The usual PublicKey item, which identifies the peer. Traffic to this peer will be encrypted using this public key.
- Endpoint: this tells WireGuard where to actually send the encrypted traffic to. Since in our scenario the laptop will be initiating connections, it has to know the public IP address of the home router. If your ISP gave you a fixed IP address, great, nothing else to do. If however you have a dynamic IP address, one that changes everytime you establish a new connection, then you will have to setup some sort of dynamic DNS service. There are many free such services available on the Internet, but setting one up is out of scope for this guide.
- In AllowedIPs we list our destinations. The VPN network 10.10.11.0/24 is listed so that we can ping wg0 on the home router as well as other devices on the same VPN, and the actual home network, which is 10.10.10.0/24.

If we had used 0.0.0.0/0 alone in AllowedIPs, then the VPN would become our default gateway, and all traffic would be sent to this peer. See Default Gateway for details on that type of setup.

Testing

With these configuration files in place, it's time to bring the WireGuard interfaces up.

On the home router, run:

\$ sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.10.11.1/24 dev wg0
[#] ip link set mtu 1378 up dev wg0

Verify you have a wg0 interface with an address of $10.10.11.1/24{:}$

\$ ip a show dev wg0
9: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1378 qdisc noqueue state UNKNOWN group default qlen 1000
 link/none
 inet 10.10.11.1/24 scope global wg0
 valid_lft forever preferred_lft forever

Verify you have a wg0 interface up with an address of 10.10.11.1/24:

\$ ip a show dev wg0
9: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1378 qdisc noqueue state UNKNOWN group default qlen 1000
 link/none
 inet 10.10.11.1/24 scope global wg0
 valid_lft forever preferred_lft forever

And a route to the 10.10.1.0/24 network via the wg0 interface:

\$ ip route | grep wg0
10.10.11.0/24 dev wg0 proto kernel scope link src 10.10.11.1

And wg show should show some status information, but no connected peer yet:

\$ sudo wg show interface: wg0 public key: <router public key> private key: (hidden) listening port: 51000

```
peer: <laptop public key>
  allowed ips: 10.10.11.2/32
```

In particular, verify that the public keys listed match what you created and expect.

Before we start the interface on the other peer, it helps to leave the above **show** command running continuously, so we can see when there are changes:

\$ sudo watch wg show

Now start the interface on the laptop:

\$ sudo wg-quick up home0
[#] ip link add home0 type wireguard
[#] wg setconf home0 /dev/fd/63
[#] ip -4 address add 10.10.11.2/24 dev home0
[#] ip link set mtu 1420 up dev home0
[#] ip -4 route add 10.10.0.0/24 dev home0

Similarly, verify the interface's IP and added routes:

\$ ip a show dev home0
24: home0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
 link/none
 inet 10.10.11.2/24 scope global home0
 valid_lft forever preferred_lft forever

\$ ip route | grep home0 10.10.10.0/24 dev home0 scope link 10.10.11.0/24 dev home0 proto kernel scope link src 10.10.11.2 Up to this point, the wg show output on the home router probably didn't change. That's because we haven't sent any traffic to the home network, which didn't trigger the VPN yet. By default, WireGuard is very "quiet" on the network.

If we trigger some traffic, however, the VPN will "wake up". Let's ping the internal address of the home router a few times:

\$ ping -c 3 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=603 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=300 ms
64 bytes from 10.10.10.1: icmp seq=3 ttl=64 time=304 ms

Note how the first ping was slower. That's because the VPN was "waking up" and being established. Afterwards, with the tunnel already established, the latency reduced.

At the same time, the wg show output on the home router will have changed to something like this:

```
$ sudo wg show
interface: wg0
public key: <router public key>
private key: (hidden)
listening port: 51000
peer: <laptop public key>
endpoint: <laptop public IP>:51000
allowed ips: 10.10.11.2/32
latest handshake: 1 minute, 8 seconds ago
```

transfer: 564 B received, 476 B sent

WireGuard on an internal system

Sometimes it's not possible to install WireGuard on the home router itself. Perhaps it's a closed system to which you do not have access, or there is no easy build for that architecture, or any other of the many possible reasons.

But you do have a spare system inside your network that you could use. Here we are going to show one way to make this work. There are others, but we believe this to be the one less involved as it only requires a couple of (very common) changes in the router itself: NAT port forwarding, and DHCP range editing.

To recap, our home network has the 10.10.0/24 address, and we want to connect to it from a remote location and be "inserted" into that network as if we were there:



10.10.10.2-9

Router changes

Since in this scenario we don't have a new network dedicated to our VPN users, we need to "carve out" a section of the home network and reserve it to the VPN.

The easiest way to reserve IPs for the VPN is to change the router configuration (assuming it's responsible for DHCP in this network) and tell its DHCP server to only hand out addresses from a specific range, leaving a "hole" for our VPN users.

For example, in the case of the 10.10.10.0/24 network, the DHCP server on the router might already be configured to hand out IP addresses from 10.10.10.2 through 10.10.10.254. We can carve out a "hole" for our VPN users by reducing the DHCP range, like below:

Network	10.10.0/24
Usable Addresses	10.10.10.2 - 10.10.10.254 (.1 is the router)
DHCP Range	10.10.10.50 - 10.10.10.254
VPN Range	10.10.10.10 - 10.10.10.59

Or any other layout that is better suited for your case. In that way, the router will never hand out a DHCP address that conflicts with one that we selected for a VPN user.

The second change we need to do in the router is to port forward the WireGuard traffic to the internal system that will be the endpoint. In the diagram above, we selected the 10.10.10.10 system to be the internal WireGuard endpoint, and we will run it on the 51000/udp port. Therefore, you need to configure the router to forward all 51000/udp traffic to 10.10.10.10.10 on the same 51000/udp port.

Finally, we also need to allow hosts on the internet to send traffic to the router on the 51000/udp port we selected for WireGuard. This is done in the firewall rules of the device. Sometimes just doing the port forwarding from before also configures the firewall to allow that traffic, but better check.

Now we are ready to configure the internal endpoint.

Configuring the internal WireGuard endpoint

Install the wireguard package:

\$ sudo apt install wireguard

Generate the keys for this host:

```
$ umask 077
$ wg genkey > internal-private.key
$ wg pubkey < internal-private.key > internal-public.key
```

And create the <code>/etc/wireguard/wg0.conf</code> file with these contents:

```
[Interface]
Address = 10.10.10.10/32
ListenPort = 51000
PrivateKey = <contents of internal-private.key>
```

```
[Peer]
# laptop
PublicKey = <contents of laptop-public.key>
AllowedIPs = 10.10.10.11/32 # any available IP in the VPN range
```

NOTE

Just like in the *Peer to Site* scenario with WireGuard on the router, there is no Endpoint configuration here for the laptop peer, because we don't know where it will be connecting from beforehand.

The final step is to configure this internal system as a router for the VPN users. For that, we need to enable a couple of settings:

- ip_forward: to enable forwarding (aka, routing) of traffic between interfaces.
- proxy_arp: to reply to *arp* requests on behalf of the VPN systems, as if they were locally present on the network segment.

To do that, and make it persist across reboots, create the file /etc/sysctl.d/70-wireguard-routing.conf file with this content:

```
net.ipv4.ip_forward = 1
net.ipv4.conf.all.proxy_arp = 1
```

Then run this command to apply those settings:

\$ sudo sysctl -p /etc/sysctl.d/70-wireguard-routing.conf -w

Now the WireGuard interface can be brought up:

\$ sudo wg-quick up wg0

Configuring the peer

The peer configuration will be very similar to what was done before. What changes will be the address, since now it won't be on an exclusive network for the VPN, but have an address carved out of the home network block.

Let's call this new configuration file /etc/wireguard/home_internal.conf:

```
[Interface]
ListenPort = 51000
Address = 10.10.10.11/24
PrivateKey = <contents of the private key for this system>
[Peer]
```

```
PublicKey = <contents of internal-public.key>
Endpoint = <home-ppp0-IP-or-hostname>:51000
AllowedIPs = 10.10.10.0/24
```

And bring up this WireGuard interface:

\$ sudo wg-quick up home_internal

NOTE

There is no need to add an index number to the end of the interface name. That is a convention, but not strictly a requirement.

Testing

With the WireGuard interfaces up on both peers, traffic should flow seamlessly in the 10.10.0/24 network between remote and local systems.

More specifically, it's best to test the non-trivial cases, that is, traffic between the remote peer and a host other than the one with the WireGuard interface on the home network.

Site to Site

Another usual VPN configuration where one could deploy WireGuard is to connect two distinct networks over the internet. Here is a simplified diagram:



The goal here is to seamlessly integrate network *alpha* with network *beta*, so that systems on the *alpha* site can transparently access systems on the *beta* site, and vice-versa.

Such a setup as a few particular details:

- Both peers are likely to be always up and running.
- We can't assume one side will always be the initiator, like the laptop in a coffee shop scenario.
- Because of the above, both peers should have a static endpoint, like a fixed IP address, or valid domain name.

- Since we are not assigning VPN IPs to all systems on each side, the VPN network here will be very small (a /31, which allows for two IPs) and only used for routing. The only systems with an IP in the VPN network are the gateways themselves.
- There will be no NAT applied to traffic going over the WireGuard network. Therefore, the networks of both sites *must* be different and not overlap.

This is what an *mtr* report from a system in the *beta* network to an *alpha* system will look like:

```
ubuntu@b1:~$ mtr -n -r 10.10.10.230
Start: 2022-09-02T18:56:51+0000
HOST: b1
                       Loss%
                                Snt
                                     Last
                                            Avg Best Wrst StDev
  1.|-- 10.10.11.1
                        0.0%
                                10
                                      0.1
                                             0.1
                                                  0.1
                                                         0.2
                                                               0.0
  2. | -- 10.10.9.0
                        0.0%
                                10 299.6 299.3 298.3 300.0
                                                               0.6
  3. |-- 10.10.10.230
                         0.0%
                                 10 299.1 299.1 298.0 300.2
                                                               0.6
```



Hopefully you will have a lower latency

NOTE

Technically, a /31 CIDR network has no usable IP addresses, since the first one is the network address, and the second (and last) one is the broadcast address. RFC 3021 however allows for it, but if you encounter routing or other networking issues, switch to a /30 CIDR and its two valid host ips.

Configuring WireGuard

On the system that is the gateway for each site, and has internet connectivity, we start by installing WireGuard and generating the keys. For the *alpha* site:

```
$ sudo apt install wireguard
$ wg genkey | sudo tee /etc/wireguard/wgA.key
$ sudo cat /etc/wireguard/wgA.key | wg pubkey | sudo tee /etc/wireguard/wgA.pub
```

And the configuration on *alpha* will be:

```
[Interface]
PostUp = wg set %i private-key /etc/wireguard/%i.key
Address = 10.10.9.0/31
ListenPort = 51000
```

[Peer]
beta site
PublicKey = <contents of /etc/wireguard/wgB.pub>
AllowedIPs = 10.10.11.0/24,10.10.9.0/31
Endpoint = <beta-gw-ip>:51000

On the gateway for the *beta* site we take similar steps:

```
$ sudo apt install wireguard
$ wg genkey | sudo tee /etc/wireguard/wgB.key
$ sudo cat /etc/wireguard/wgB.key | wg pubkey | sudo tee /etc/wireguard/wgB.pub
```

And create the corresponding configuration file for *beta*:

```
[Interface]
Address = 10.10.9.1/31
PostUp = wg set %i private-key /etc/wireguard/%i.key
ListenPort = 51000
[Peer]
# alpha site
PublicKey = <contents of /etc/wireguard/wgA.pub>
AllowedIPs = 10.10.10.0/24,10.10.9.0/31
Endpoint = <alpha-gw-ip>:51000
```

IMPORTANT

WireGuard is being setup on the gateways for these two networks. As such, there are no changes needed on individual hosts of each network, but keep in mind that the WireGuard tunneling and encryption is only happening between the *alpha* and *beta* gateways, and **NOT** between the hosts of each network.

Bringing the interfaces up

Since this VPN is permanent between static sites, it's best to use the *systemd* unit file for wg-quick to bring the interfaces up and control them in general. In particular, we want them to be brought up automatically on reboot events.

On alpha:

\$ sudo systemctl enable --now wg-quick@wgA

And similarly on *beta*:

\$ sudo systemctl enable --now wg-quick@wgB

This both enables the interface on reboot, and starts it right away.

Firewall and routing

Both gateways probably already have some routing and firewall rules. These might need changes depending on how they are setup.

The individual hosts on each network won't need any changes regarding the remote *alpha* or *beta* networks, because they will just send that traffic to the default gateway (as any other non-local traffic), which knows how to route it because of the routes that wg-quick added.

In the configuration we did so far, there have been no restrictions in place, so traffic between both sites flows without impediments.

In general, what needs to be done or checked is:

- Make sure both gateways can contact each other on the specified endpoint addresses and UDP port. In the case of this example, that's port 51000. For extra security, create a firewall rule only allowing each peer to contact this port, instead of the Internet at large.
- Do NOT masquerade or NAT the traffic coming from the internal network and going out via the WireGuard interface towards the other site. This is purely routed traffic.
- There shouldn't be any routing changes needed on the gateways, since wg-quick takes care of adding the route for the remote site, but do check the routing table to see if it makes sense (ip route and ip route | grep wg are a good start).
- You may have to create new firewall rules if you need to restrict traffic between the *alpha* and *beta* networks. For example, if you want to prevent *ssh* between the sites, you could add a firewall rule like this one to *alpha*:

\$ sudo iptables -A FORWARD -i wgA -p tcp --dport 22 -j REJECT

And similarly on *beta*:

\$ sudo iptables -A FORWARD -i wgB -p tcp --dport 22 -j REJECT

You can add these as PostUp actions in the WireGuard interface config. Just don't forget the remove them in the corresponding PreDown hook, or you will end up with multiple rules.

Using the VPN as the default gateway

WireGuard can also be setup to route all traffic through the VPN, and not just specific remote networks. There could be many reasons for this, but mostly they are related to privacy.

Here we will assume a scenario where the local network is considered untrusted, and we want to leak as little information as possible about our behavior on the Internet. This could be the case of an airport, or a coffee shop, a conference, a hotel, or any other public network.

	public untrusted	—	wg0	10.90.90.2/24
10.90.90.1/24	network/internet		VPN	network
wg0	xxxxxx	<u> </u>		
	xx xxxxx VPN	gw		



For this to work best, we need a system we can reach on the internet and that we control. Most commonly this can be a simple small VM in a public cloud, but a home network also works. Here we will assume it's a brand new system that will be configured from scratch for this very specific purpose.

WireGuard Configuration

Let's start the configuration by installing WireGuard and generating the keys.

On the client:

```
$ sudo apt install wireguard
$ umask 077
$ wg genkey > wg0.key
$ wg pubkey < wg0.key > wg0.pub
$ sudo mv wg0.key wg0.pub /etc/wireguard
And on the gateway server:
```

```
$ sudo apt install wireguard
$ umask 077
$ wg genkey > gateway0.key
$ wg pubkey < gateway0.key > gateway0.pub
$ sudo mv gateway0.key gateway0.pub /etc/wireguard
```

On the client, we will create /etc/wireguard/wg0.conf:

```
[Interface]
PostUp = wg set %i private-key /etc/wireguard/wg0.key
ListenPort = 51000
Address = 10.90.90.1/24
```

[Peer]
PublicKey = <contents of gateway0.pub>
Endpoint = <public IP of gateway server>
AllowedIPs = 0.0.0.0/0

Key points here:

- We selected the 10.90.90.1/24 IP address for the WireGuard interface. This can be any private IP address, as long as it doesn't conflict with the network you are on, so double check that. If it needs changing, don't forget to also change the IP for the WireGuard interface on the gateway server.
- The AllowedIPs value is 0.0.0.0/0, which means "all IPv4 addresses".
- We are using PostUp to load the private key instead of specifying it directly in the configuration file, so we don't have to set the permissions on the config file to 0600.

The counterpart configuration on the gateway server is /etc/wireguard/gateway0.conf with these contents:

```
[Interface]
PostUp = wg set %i private-key /etc/wireguard/%i.key
Address = 10.90.90.2/24
ListenPort = 51000
[Peer]
```

PublicKey = <contents of wg0.pub>
AllowedIPs = 10.90.90.1/32

Since we don't know from where this remote peer will be connecting, there is no Endpoint setting for it, and the expectation is that the peer will be the one initiating the VPN.

This finishes the WireGuard configuration on both ends, but there is one extra step we need to take on the gateway server.

Routing and masquerading

The WireGuard configuration that we did so far is enough to send the traffic from the client in the untrusted network, to the gateway server. But what about from there on? There are two extra configs we need to make on the gateway server:

- Masquerade (or apply source NAT rules) the traffic from 10.90.90.1/24.
- Enable IPv4 forwarding so our gateway server acts as a router.

To enable routing, create /etc/sysctl.d/70-wireguard-routing.conf with this content:

net.ipv4.ip_forward = 1

And run:

\$ sudo sysctl -p /etc/sysctl.d/70-wireguard-routing.conf -w

To masquerade the traffic from the VPN, one simple rule is needed:

\$ sudo iptables -t nat -A POSTROUTING -s 10.90.90.0/24 -o eth0 -j MASQUERADE

Replace eth0 with the name of the network interface on the gateway server, if it's different.

To have this rule persist across reboots, you can add it to /etc/rc.local (create the file if it doesn't exist and make it executable):

#!/bin/sh

iptables -t nat -A POSTROUTING -s 10.90.90.0/24 -o eth0 -j MASQUERADE

This completes the gateway server configuration.

Testing

Let's bring up the WireGuard interfaces on both peers.

On the gateway server:

```
$ sudo wg-quick up gateway0
[#] ip link add gateway0 type wireguard
[#] wg setconf gateway0 /dev/fd/63
[#] ip -4 address add 10.90.90.2/24 dev gateway0
[#] ip link set mtu 1378 up dev gateway0
[#] wg set gateway0 private-key /etc/wireguard/gateway0.key
```

And on the client:

\$ sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.90.90.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] wg set wg0 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] nft -f /dev/fd/63
[#] wg set wg0 private-key /etc/wireguard/wg0.key

From the client you should now be able to verify that your traffic reaching out to the internet is going through the gateway server via the WireGuard VPN. For example:

```
$ mtr -r 1.1.1.1
Start: 2022-09-01T12:42:59+0000
HOST: laptop.lan
                                Loss%
                                        Snt
                                              Last
                                                     Avg Best Wrst StDev
  1. | -- 10.90.90.2
                                  0.0%
                                          10 184.9 185.5 184.9 186.9
                                                                        0.6
  2. | -- 10.48.128.1
                                  0.0%
                                          10 185.6 185.8 185.2 188.3
                                                                        0.9
  (...)
                                  0.0%
                                          10 186.2 186.3 185.9 186.6
  7. |-- one.one.one.one
                                                                        0.2
```

Above, hop 1 is the gateway0 interface on the gateway server, then 10.48.128.1 is the default gateway for that server, then come some in between hops, and the final hit is the target.

If you just look at the output of ip route, however, it's not immediately obvious that the WireGuard VPN is the default gateway:

```
$ ip route
default via 192.168.122.1 dev enp1s0 proto dhcp src 192.168.122.160 metric 100
10.90.90.0/24 dev wg0 proto kernel scope link src 10.90.90.1
192.168.122.0/24 dev enp1s0 proto kernel scope link src 192.168.122.160 metric 100
192.168.122.1 dev enp1s0 proto dhcp scope link src 192.168.122.160 metric 100
```

That's because WireGuard is using *fwmarks* and policy routing. WireGuard cannot simply set the $wg\theta$ interface as the default gateway: that traffic needs to reach the specified endpoint on port 51000/UDP outside of the VPN tunnel.

If you want to dive deeper into how this works, check ip rule list, ip route list table 51820, and consult the documentation on "Linux Policy Routing".

DNS leaks

The traffic is now being routed through the VPN to the gateway server that you control, and from there on to the Internet at large. The local network you are in cannot see the contents of that traffic, because it's encrypted. But you are still leaking information about the sites you access via DNS.

When the laptop got its IP address in the local (untrusted) network it is sitting in, it likely also got a pair of IPs for DNS servers to use. These might be servers from that local network, or other DNS servers from the internet like 1.1.1.1 or 8.8.8.8. When you access an internet site, a DNS query will be sent to those servers to discover their IP addresses. Sure, that traffic goes over the VPN, but at some point it exits the VPN, and then reaches those servers, which will then know what you are trying to access.

There are DNS leak detectors out there, and if you want a quick check you can try out https://dnsleaktest.com. It will tell you which DNS servers your connection is using, and it's up to you if you trust them or not. You might be surprised that, even if you are in a conference network for example, using a default gateway VPN like the one described here, you are still using the DNS servers from the conference infrastructure. In a way, the DNS traffic is leaving your machine encrypted, and then coming back in clear text to the local DNS server.

There are basically two things you can do about this: select a specific DNS server to use for your VPN connection, or install your own DNS server.

Selecting a DNS server

If you can use a DNS server that you trust, or don't mind using, that is probably the easiest solution. Many people would start with the DNS server assigned to the gateway server used for the VPN. This address can be checked by running the following command in a shell on the gateway server:

```
$ resolvectl status
Global
      Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub
Link 2 (ens2)
      Current Scopes: DNS
         Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.48.0.5
         DNS Servers: 10.48.0.5
         DNS Domain: openstacklocal
Link 5 (gateway0)
Current Scopes: none
        Protocols: -DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
```

Look for Current DNS Server. In the example above, it's 10.48.0.5.

Let's change the WireGuard wg0 interface config to use that DNS server. Edit /etc/wireguard/wg0.conf and add a second PostUp line with the resolvectl command like below:

```
[Interface]
PostUp = wg set %i private-key /etc/wireguard/wg0.key
PostUp = resolvectl dns %i 10.48.0.5; resolvectl domain %i \~.
ListenPort = 51000
Address = 10.90.90.1/24
```

[Peer]
PublicKey = <contents of gateway0.pub>
Endpoint = <public IP of gateway server>
AllowedIPs = 0.0.0.0/0

You can run that resolvectl command by hand if you want to avoid having to restart the WireGuard VPN:

 $\$ sudo resolvectl dns wg0 10.48.0.5; sudo resolvectl domain wg0 $\~\$

Or just restart the WireGuard interface:

\$ sudo wg-quick down wg0; sudo wg-quick up wg0

And if you check again for DNS leaks, this time you'll that you are only using the DNS server you specified.

Installing your own DNS server

If you don't want to use even the DNS server from the hosting provider where you have your gateway server, another alternative is to install your own DNS server.

There are multiple choices out there for this: bind9 and unbound are quite popular, and easy to find quick tutorials and instructions on how to do it.

Here we will proceed with bind9, which is in the Ubuntu Main repository.

On the gateway server, install the bind9 package:

```
$ sudo apt install bind9
```

And that's it for the server part.

On the client, add a PostUp line specifying this IP (or change the line we added in the previous section):

```
[Interface]
PostUp = wg set %i private-key /etc/wireguard/wg0.key
PostUp = resolvectl dns %i 10.90.90.2; resolvectl domain %i \~.
ListenPort = 51000
Address = 10.90.90.1/24
```

```
[Peer]
PublicKey = <contents of gateway0.pub>
Endpoint = <public IP of gateway server>
AllowedIPs = 0.0.0.0/0
```

And restart the WireGuard interface. Now your VPN client will be using the gateway server as the DNS server.

Other Tasks

Here are some common tasks and other helpful tips that can help you in your WireGuard deployment.

Controlling the WireGuard interface with systemd

The wg-quick tool is a simple way to bring the WireGuard interface up and down. That control is also exposed via a systemd service, which means the standard systemctl tool can be used.

Probably the best benefit of this is to be able to configure the interface to be brought up automatically when the system is booted up. For example, to configure the wg0 interface to be brought up at boot:

$\$ sudo systemctl enable wg-quick@wg0

The name of the systemd service follows the WireGuard interface name, and multiple such services can be enabled/started at the same time. You can also use the systemctl status, start, stop, reload and restart commands to control the WireGuard interface and query its status:

$\$ sudo systemctl reload wg-quick@wg0

The reload action does what we expect: it reloads the configuration of the interface without disrupting existing WireGuard tunnels. To add or remove peers, reload is sufficient, but if wg-quick options, like PostUp, Address or others alike, are changed, then a restart is needed.

DNS resolving

Let's say when you are inside the home network, literally at home, you can connect to your other systems via DNS names, because your router at 10.10.10.1 can act as an internal DNS server. It would be nice to have this capability also when connected via the WireGuard VPN.

To do that, we can add a PostUp command to the WireGuard configuration to run a command for us right after the VPN is established. This command can be anything you would run in a shell, as root. We can use that to adjust the DNS resolver configuration of the laptop that is remotely connected to the home network.

For example, if we have a WireGuard setup as follows:

- home0 WIreGuard interface.
- .home DNS domain for the remote network.
- 10.10.10.1/24 is the DNS server for the .home domain, reachable after the VPN is established.

We can add this PostUp command to the home0.conf configuration file to have our systemd-based resolver use 10.10.10.1 as the DNS server for any queries for the .home domain:

[Interface]

. . .

PostUp = resolvectl dns %i 10.10.10.1; resolvectl domain %i \~home

For PostUp (and PostDown, see the wg-quick(8) manpage for details), the i text is replaced with the WireGuard interface name. In this case, that would be home0.

These two resolvectl commands tell the local *systemd-resolved* resolver to a) associate the DNS server at 10.10.10.1 to the home0 interface; and b) associate the home domain to the home0 interface.

When you bring the homeo WireGuard interface up again, it will run the resolvectl commands:

\$ sudo wg-quick up home0
[#] ip link add home0 type wireguard
[#] wg setconf home0 /dev/fd/63
[#] ip -4 address add 10.10.11.2/24 dev home0
[#] ip link set mtu 1420 up dev home0
[#] ip -4 route add 10.10.0/24 dev home0
[#] resolvectl dns home0 10.10.10.1; resolvectl domain home0 \~home

You can verify that it worked by pinging some hostname in your home network, or checking the DNS resolution status for the home0 interface:

```
$ resolvectl status home0
Link 26 (home0)
Current Scopes: DNS
Protocols: -DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.10.10.1
DNS Servers: 10.10.10.1
DNS Domain: ~home
```

If you are using *systemctl* to control the WireGuard interface, this is the type of change (adding or changing PostUp) where the reload action won't be enough, and you actually need to issue a restart.

NOTE

The wg-quick(8) manpage documents the DNS setting of the WireGuard interface which has the same purpose, but only works if you have resolveconf installed. Ubuntu systems by default don't, and rely on systemd-resolved instead.

Adding another peer

To add another peer to an existing WireGuard setup, we have to:

- 1. generate a new keypair for the new peer
- 2. create a new [Peer] section on the "other side" of the WireGuard setup
- 3. pick a new IP for the new peer

Let's call the new system ontheroad, and generate the keys for it:

```
$ umask 077
```

```
$ wg genkey > ontheroad-private.key
```

```
$ wg pubkey < ontheroad-private.key > ontheroad-public.key
```

\$ ls -la ontheroad.*
-rw----- 1 ubuntu ubuntu 45 Aug 22 20:12 ontheroad-private.key
-rw----- 1 ubuntu ubuntu 45 Aug 22 20:13 ontheroad-public.key

As for its IP address, let's pick 10.10.11.3/24 for it, which is the next one in sequence of one of the previous examples in this guide:

```
[Interface]
PrivateKey = <contents-of-ontheroad-private.key>
ListenPort = 51000
Address = 10.10.11.3/24
```

[Peer]
PublicKey = <contents-of-router-public.key>
Endpoint = <home-ppp0-IP-or-hostname>:51000
AllowedIPs = 10.10.11.0/24,10.10.10.0/24

The only difference between this config and one for an existing system in this same WireGuard setup will be PrivateKey and Address.

On the "other side", we add the new [Peer] section to the existing config:

```
[Interface]
PrivateKey = <contents-of-router-private.key>
ListenPort = 51000
Address = 10.10.11.1/24
```

```
[Peer]
# laptop
PublicKey = <contents-of-laptop-public.key>
AllowedIPs = 10.10.11.2
```

```
[Peer]
# ontheroad
PublicKey = <contents-of-ontheroad-public.key>
AllowedIPs = 10.10.11.3
```

To update the interface with the new peer without disrupting existing connections, we use the **reload** action of the systemd unit:

\$ systemctl reload wg-quick@wg0

NOTE

For this case of a "server" or "vpn gateway", where we are just adding another peer to an existing config, the systemctl reload action will work well enough to insert the new peer in the wireguard configuration. But it won't create new routes, or do any of the other steps that wg-quick does. Depending on the setup, you might need a full restart so that wg-quick can fully do its job.

Adding a smartphone peer

WireGuard can be installed on many different platforms, and smartphones are included. The usptream installation page has links for Android and iOS apps.

Such a mobile client can be configured more easily with the use of QR codes.

We start by creating the new peer's config normally, as if it were any other system (generate keys, pick an IP address, etc). Then, to convert that configuration file to a QR code, install the **qrencode** package:

\$ sudo apt install grencode

And run the following command (assuming the config was written to phone.conf):

\$ cat phone.conf | qrencode -t ansiutf8

That will generate a QR code in the terminal, ready for scanning with the smartphone app. Note that there is no need for a graphical environment, and this command can be run remotely over SSH for example.

Note that you need to put the private key contents directly into that configuration file, and not use PostUp to load it from a separate file.
IMPORTANT

Treat this QR code as a secret, as it contains the private key for the wireguard interface!

Security tips

Here are some security tips for your WireGuard deployment.

Traffic goes both ways

Remember that the VPN traffic goes both ways. Once you are connected to the remote network, it means any device on that network can connect back to you! That is, unless you create specific firewall rules for this VPN network.

Since WireGuard is "just" an interface, you can create normal firewall rules for its traffic, and control the access to the network resources as usual. This is done more easily if you have a dedicated network for the VPN clients.

Using PreSharedKey

You may add another layer of cryptographic protection to your VPN with the PreSharedKey option. Its usage is optional, and adds a layer of symmetric-key cryptography to the traffic between specific peers.

Such a key can be generated with the ${\tt genpsk}$ command:

```
$ wg genpsk
vxlX6eMMin8uhxbKEhe/i0xi8ru+q1qWzCdjESXoFZY=
```

And then used in a [Peer] section, like this:

```
[Peer]
PublicKey = ....
Endpoint = ....
AllowedIPs = ....
PresharedKey = vxlX6eMMin8uhxbKEhe/i0xi8ru+qlqWzCdjESXoFZY=
```

Both sides need to have the same PresharedKey in their respective [Peer] sections.

Preventing accidental leakage of the private keys

When troubleshooting WireGuard, it's common to post the contents of the interface configuration file somewhere for others to help, like in a mailing list, or internet forum. Since the private key is listed in that file, one has to remember to strip or obfuscate it before sharing, or else the secret is leaked.

To avoid such mistakes, we can remove the private key from the configuration and leave it in its own file. This can be done via a PostUp hook. For example, let's update the home0.conf file to use such a hook:

```
[Interface]
ListenPort = 51000
Address = 10.10.11.3/24
PostUp = wg set %i private-key /etc/wireguard/%i.key
```

```
[Peer]
PublicKey = <contents-of-router-public.key>
Endpoint = 10.48.132.39:51000
AllowedIPs = 10.10.11.0/24,10.10.10.0/24
```

The %i macro is replaced by the WireGuard interface name (home0 in this case). When the interface comes up, the PostUp shell commands will be executed with that substitution in place, and the private key for this interface will be set with the contents of the /etc/wireguard/home0.key file.

There are some other advantages to this method, and perhaps one disadvantage.

Pros:

- The configuration file can now safely be stored in version control, like a git repository, without fear of leaking the private key (unless you also use the PreSharedKey option, which is also a secret).
- Since the key is now stored in a file, you can give that file a meaningful name, which helps to avoid mixups with keys and peers when setting up WireGuard.

Cons:

• You cannot directly use the **qrcode** tool to convert this image to a qrcode and use it to configure the mobile version of WireGuard, because that tool won't go after the private key in that separate file.

Troubleshooting WireGuard

General troubleshooting checklist:

- Verify public and private keys. When dealing with multiple peers, it's easy to mix these up, specially because the contents of these keys is just random data. There is nothing identifying them, and public and private keys are basically the same format-wise.
- Verify AllowedIPs list on all peers.
- Check with ip route and ip addr show dev <wg-interface> if the routes and IPs are set as you expect.
- Double check that you have /proc/sys/net/ipv4/ip_forward set to 1 where needed.
- When injecting the VPN users into an existing network, without routing, make sure /proc/sys/net/ipv4/conf/all/proxy_ar is set to 1.
- Make sure the above /proc entries are in /etc/sysctl.conf or a file in /etc/sysctl.d so that they persist reboots.

It can be helpful to leave a terminal open with the watch wg command. Here is a sample output showing a system with two peers configured, where only one has established the VPN so far:

Every 2.0s: wg j-wg: Fri Aug 26 17:44:37 2022

```
interface: wg0
  public key: +T3T3HTMeyrEDvim8FBxbYjbz+/P0e0tG3Rlvl9kJmM=
  private key: (hidden)
  listening port: 51000
```

```
peer: 2cJdFcNzXv4YUGyDTahtOfrbsrFsCByatPnNzKTs0Qo=
endpoint: 10.172.196.106:51000
allowed ips: 10.10.11.2/32
latest handshake: 3 hours, 27 minutes, 35 seconds ago
transfer: 3.06 KiB received, 2.80 KiB sent
```

```
peer: ZliZlhlarZqvfxPMyME2ECtXDk611NB7uzLAD4McpgI=
    allowed ips: 10.10.11.3/32
```

Kernel debug messages

WireGuard is also silent when it comes to logging. Being a kernel module essentially, we need to explicitly enable verbose logging of its module. This is done with the following command:

\$ echo "module wireguard +p" | sudo tee /sys/kernel/debug/dynamic_debug/control

This will write WireGuard logging messages to the kernel log, which can be watched live with:

\$ sudo dmesg -wT

To disable logging, run this:

\$ echo "module wireguard -p" | sudo tee /sys/kernel/debug/dynamic_debug/control

Destination address required

If you ping an IP and get back an error like this:

```
$ ping 10.10.11.2
PING 10.10.11.2 (10.10.11.2) 56(84) bytes of data.
From 10.10.11.1 icmp_seq=1 Destination Host Unreachable
ping: sendmsg: Destination address required
```

This is happening because the WireGuard interface selected for this destination doesn't know the endpoint for it. In other words, it doesn't know where to send the encrypted traffic.

One common scenario for this is on a peer where there is no Endpoint configuration, which is perfectly valid, and the host is trying to send traffic to that peer. Let's take the coffee shop scenario we described earlier as an example.

The laptop is connected to the VPN and exchanging traffic as usual. Then it stops for a bit (the person went to get one more cup). Traffic ceases (WireGuard is silent, remember). If the WireGuard on the home router is now restarted, when it comes back up, it won't know how to reach the laptop, because it was never contacted by it before. This means that at this time, if the home router tries to send traffic to the laptop in the coffee shop, it will get the above error.

Now the laptop user comes back, and generates some traffic to the home network (remember: the laptop has the home network's Endpoint value). The VPN "wakes up", data is exchanged, handshakes completed, and now the home router knows the Endpoint associated with the laptop, and can again initiate new traffic to it without issues.

Another possibility is that one of the peers is behind a NAT, and there wasn't enough traffic for the stateful firewall to consider the "connection" alive, and it dropped the NAT mapping it had. In this case, the peer might benefit from the PersistentKeepalive configuration, which makes WireGuard send a *keepalive* probe every so many seconds.

Required key not available

This error:

```
$ ping 10.10.11.1
PING 10.10.11.1 (10.10.11.1) 56(84) bytes of data.
From 10.10.11.2 icmp_seq=1 Destination Host Unreachable
ping: sendmsg: Required key not available
```

Can happen when you have a route directing traffic to the WireGuard interface, but that interface does not have the target address listed in its AllowedIPs configuration.

If you have enabled kernel debugging for WireGuard, you will also see a message like this one in the dmesg output:

wireguard: home0: No peer has allowed IPs matching 10.10.11.1

CUPS - Print Server

The primary mechanism for Ubuntu printing and print services is the **Common UNIX Printing System** (CUPS). This printing system is a freely available, portable printing layer which has become the new standard for printing in most Linux distributions.

CUPS manages print jobs and queues and provides network printing using the standard Internet Printing Protocol (IPP), while offering support for a very large range of printers, from dot-matrix to laser and many in between. CUPS also supports PostScript Printer Description (PPD) and auto-detection of network printers, and features a simple web-based configuration and administration tool.

Installation

To install CUPS on your Ubuntu computer, simply use sudo with the apt command and give the packages to install as the first parameter. A complete CUPS install has many package dependencies, but they may all be specified on the same command line. Enter the following at a terminal prompt to install CUPS:

sudo apt install cups

Upon authenticating with your user password, the packages should be downloaded and installed without error. Upon the conclusion of installation, the CUPS server will be started automatically.

For troubleshooting purposes, you can access CUPS server errors via the error log file at: /var/log/cups/error_log. If the error log does not show enough information to troubleshoot any problems you encounter, the verbosity of the CUPS log can be increased by changing the **LogLevel** directive in the configuration file (discussed below) to "debug" or even "debug2", which logs everything, from the default of "info". If you make this change, remember to change it back once you've solved your problem, to prevent the log file from becoming overly large.

Configuration

The Common UNIX Printing System server's behavior is configured through the directives contained in the file /etc/cups/cupsd.conf. The CUPS configuration file follows the same syntax as the primary configuration file for the Apache HTTP server, so users familiar with editing Apache's configuration file should feel at ease when editing the CUPS configuration file. Some examples of settings you may wish to change initially will be presented here.

Tip

Prior to editing the configuration file, you should make a copy of the original file and protect it from writing, so you will have the original settings as a reference, and to reuse as necessary.

Copy the /etc/cups/cupsd.conf file and protect it from writing with the following commands, issued at a terminal prompt:

sudo cp /etc/cups/cupsd.conf /etc/cups/cupsd.conf.original

sudo chmod a-w /etc/cups/cupsd.conf.original

• ServerAdmin: To configure the email address of the designated administrator of the CUPS server, simply edit the /etc/cups/cupsd.conf configuration file with your preferred text editor, and add or modify the *ServerAdmin* line accordingly. For example, if you are the Administrator for the CUPS server, and your e-mail address is 'bjoy@somebigco.com', then you would modify the ServerAdmin line to appear as such:

ServerAdmin bjoy@somebigco.com

• Listen: By default on Ubuntu, the CUPS server installation listens only on the loopback interface at IP address 127.0.0.1. In order to instruct the CUPS server to listen on an actual network adapter's IP address, you must specify either a hostname, the IP address, or optionally, an IP address/port pairing via the addition of a Listen directive. For example, if your CUPS server resides on a local network at the IP address 192.168.10.250 and you'd like to make it accessible to the other systems on this subnetwork, you would edit the /etc/cups/cupsd.conf and add a Listen directive, as such:

Listen 127.0.0.1:631 # existing loopback Listen Listen /var/run/cups/cups.sock # existing socket Listen Listen 192.168.10.250:631 # Listen on the LAN interface, Port 631 (IPP)

In the example above, you may comment out or remove the reference to the Loopback address (127.0.0.1) if you do not wish cupsd to listen on that interface, but would rather have it only listen on the Ethernet interfaces of the Local Area Network (LAN). To enable listening for all network interfaces for which a certain hostname is bound, including the Loopback, you could create a Listen entry for the hostname *socrates* as such:

Listen socrates:631 # Listen on all interfaces for the hostname 'socrates'

or by omitting the Listen directive and using *Port* instead, as in:

Port 631 # Listen on port 631 on all interfaces

For more examples of configuration directives in the CUPS server configuration file, view the associated system manual page by entering the following command at a terminal prompt:

man cupsd.conf

Note

Whenever you make changes to the /etc/cups/cupsd.conf configuration file, you'll need to restart the CUPS server by typing the following command at a terminal prompt:

sudo systemctl restart cups.service

Web Interface

Tip

CUPS can be configured and monitored using a web interface, which by default is available at http://localhost:631/admin. The web interface can be used to perform all printer management tasks.

In order to perform administrative tasks via the web interface, you must either have the root account enabled on your server, or authenticate as a user in the *lpadmin* group. For security reasons, CUPS won't authenticate a user that doesn't have a password.

To add a user to the *lpadmin* group, run at the terminal prompt:

sudo usermod -aG lpadmin username

Further documentation is available in the Documentation/Help tab of the web interface.

References

CUPS Website

Debian Open-iSCSI page

Debuginfod

This section describes what debuginfod is and how users can benefit from Ubuntu's debuginfod service.

What is debuginfod?

From the project's official page:

debuginfod is a client/server [...] that automatically distributes ELF/DWARF/source-code from servers to clients such as debuggers across HTTP.

This means that users will be able to debug programs shipped with Ubuntu without the need to manually install the distribution's debuginfo packages.

Ubuntu maintains its own debuginfod service, which regularly indexes the debug symbols present in ddebs and other packages and serve this information over HTTPS.

Currently, the service only provides DWARF information. There are plans to make it also index and serve source-code in the future.

Using the service

From Kinetic onwards, when you install GDB you will automatically be asked whether you would like to use Ubuntu's debuginfod service or not. You can also choose to change your answer by reconfiguring the libdebuginfod-common package:

dpkg-reconfigure libdebuginfod-common

That will install a system-wide configuration that either enables or disables the use of debuginfod. If you would like to make sure that you will always use the service no matter what the system-wide setting says, you can export the DEBUGINFOD_URLS environment variable inside your shell's configuration file (if you are using Bash, this will likely be ~/.bashrc). The following snippet will do:

export DEBUGINFOD_URLS="https://debuginfod.ubuntu.com"

When you run GDB, and if you have the DEBUGINFOD_URLS variable in your environment, you will be asked again whether you would like to use the service. If you want to make sure that GDB always uses debuginfod, you can put the following snippet inside your ~/.gdbinit file:

set debuginfod enabled on

The debug symbol files will be downloaded on-the-fly during the debugging session, and will be saved locally inside the ~/.cache/debuginfod_client/ directory. You can safely remove this directory or any files inside it; they will be downloaded again only if and when needed.

Example session with GDB

Assuming that you have enabled the use of debuginfod in your system, here is what happens when you invoke GDB to debug a binary from an Ubuntu package:

\$ gdb -q program Reading symbols from program...

This GDB supports auto-downloading debuginfo from the following URLs: https://debuginfod.ubuntu.com Enable debuginfod for this session? (y or [n])

When you answer y to the question above, GDB will download the debug symbols for program:

```
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading 0.20 MB separate debug info for /home/ubuntu/program
Reading symbols from /home/ubuntu/.cache/debuginfod_client/c0fbda15a807f880e9d0b2dcc635eeeb1f0f728e/debuginfo...
(gdb)
```

Opting out of the service

If, for some reason, you prefer not to use the service, you can opt-out of it by unsetting the DEBUGINFOD_URLS environment variable. This can be done by putting the following snippet inside your shell's configuration file:

unset DEBUGINFOD_URLS

You can also disable GDB's willingness to use debuginfod by putting the following snippet inside your ~/.gdbinit:

set debuginfod enabled off

FAQ

If you have more questions about the service, please refer to the FAQ page.

Debuginfod - FAQ

What releases of Ubuntu are supported by the service?

debuginfod is currently indexing ddebs packages from all supported Ubuntu releases. Once a release goes unsupported, we stop indexing ddebs from it and eventually stop serving debug symbols for its packages.

How does debuginfod know how to find the debug symbols for the binary I am debugging?

debuginfod relies on a unique hash that identifies binaries and shared libraries called **Build-ID**. This 160-bit SHA-1 hash is generated by the compiler, and can be consulted using tools like readelf:

```
$ readelf -n /usr/bin/bash
```

```
Displaying notes found in: .note.gnu.property
  0wner
                       Data size
                                        Description
  GNU
                                        NT_GNU_PROPERTY_TYPE_0
                       0x00000020
      Properties: x86 feature: IBT, SHSTK
        x86 ISA needed: x86-64-baseline
Displaying notes found in: .note.gnu.build-id
  0wner
                       Data size
                                        Description
  GNU
                       0x00000014
                                        NT GNU BUILD ID (unique build ID bitstring)
    Build ID: 3e770d2cd0302c6ff2a184e8d2bf4ec98cfcded4
Displaying notes found in: .note.ABI-tag
  0wner
                       Data size
                                         Description
  GNU
                       0x00000010
                                        NT_GNU_ABI_TAG (ABI version tag)
    OS: Linux, ABI: 3.2.0
```

When you are debugging a program, GDB will send the program's Build-ID to the debuginfod server, who will check if it has the corresponding debug information for that binary/library. If it does, then it will send the debug symbols over HTTPS back to GDB.

Where does debuginfod store the debug information it downloads?

The debug symbols downloaded from the service are saved inside \$XDG_CACHE_HOME/.debuginfod_client/. If \$XDG_CACHE_HOME is empty, then ~/.cache/ is used instead.

Can I remove files from the cache directory?

Yes, you can. debuginfod will simply download them again if and when needed.

Can ddebs packages co-exist with debuginfod?

Yes. GDB will try to use local debug information if available. That means that if you have a ddeb package installed that provides the necessary debug symbols for the program being debugged (or if you have already downloaded that information from the debuginfod service earlier), then GDB will use it in favour of performing the download.

Can I use debuginfod with my own binary that links against system libraries?

Yes! debuginfod will obviously not be able to provide any debug symbols for your own program, but it will happily serve debug information for any system libraries that your program links against.

Domain Name Service (DNS)

Domain Name Service (DNS) is an Internet service that maps IP addresses and fully qualified domain names (FQDN) to one another. In this way, DNS alleviates the need to remember IP addresses. Computers that run DNS are called *name servers*. Ubuntu ships with BIND (Berkley Internet Naming Daemon), the most common program used for maintaining a name server on Linux.

Installation

At a terminal prompt, enter the following command to install dns:

sudo apt install bind9

A very useful package for testing and troubleshooting DNS issues is the dnsutils package. Very often these tools will be installed already, but to check and/or install dnsutils enter the following:

sudo apt install dnsutils

Configuration

There are many ways to configure BIND9. Some of the most common configurations are a caching nameserver, primary server, and secondary server.

- When configured as a caching nameserver BIND9 will find the answer to name queries and remember the answer when the domain is queried again.
- As a primary server, BIND9 reads the data for a zone from a file on its host and is authoritative for that zone.
- As a secondary server, BIND9 gets the zone data from another nameserver that is authoritative for the zone.

Overview

The DNS configuration files are stored in the /etc/bind directory. The primary configuration file is /etc/bind/named.conf, which in the layout provided by the package just includes these files.

- /etc/bind/named.conf.options: global DNS options
- /etc/bind/named.conf.local: for your zones
- /etc/bind/named.conf.default-zones: default zones such as localhost, its reverse, and the root hints

The root nameservers used to be described in the file /etc/bind/db.root. This is now provided instead by the /usr/share/dns/root.hints file shipped with the dns-root-data package, and is referenced in the named.conf.default-zones configuration file above.

It is possible to configure the same server to be a caching name server, primary, and secondary: it all depends on the zones it is serving. A server can be the Start of Authority (SOA) for one zone, while providing secondary service for another zone. All the while providing caching services for hosts on the local LAN.

Caching Nameserver

The default configuration acts as a caching server. Simply uncomment and edit /etc/bind/named.conf.options to set the IP addresses of your ISP's DNS servers:

```
forwarders {
1.2.3.4;
5.6.7.8;
```

};

Note

Replace $1.2.3.4 \ {\rm and} \ 5.6.7.8$ with the IP Addresses of actual names ervers.

To enable the new configuration, restart the DNS server. From a terminal prompt:

```
sudo systemctl restart bind9.service
```

See dig for information on testing a caching DNS server.

Primary Server

In this section BIND9 will be configured as the Primary server for the domain example.com. Simply replace example.com with your FQDN (Fully Qualified Domain Name).

Forward Zone File

To add a DNS zone to BIND9, turning BIND9 into a Primary server, first edit /etc/bind/named.conf.local:

```
zone "example.com" {
    type master;
    file "/etc/bind/db.example.com";
}
```

};

Note

If bind will be receiving automatic updates to the file as with DDNS, then use /var/lib/bind/db.example.com rather than /etc/bind/db.example.com both here and in the copy command below.

Now use an existing zone file as a template to create the /etc/bind/db.example.com file:

sudo cp /etc/bind/db.local /etc/bind/db.example.com

Edit the new zone file /etc/bind/db.example.com and change localhost. to the FQDN of your server, leaving the additional . at the end. Change 127.0.0.1 to the nameserver's IP Address and root.localhost to a valid email address, but with a . instead of the usual @ symbol, again leaving the . at the end. Change the comment to indicate the domain that this file is for.

Create an *A record* for the base domain, example.com. Also, create an *A record* for ns.example.com, the name server in this example:

```
BIND data file for example.com
;
:
        604800
$TTL
        IN
                         example.com. root.example.com. (
6
                 SOA
                                2
                                           ; Serial
                           604800
                                           ; Refresh
                            86400
                                           ; Retry
                         2419200
                                           ; Expire
                           604800)
                                           ; Negative Cache TTL
(d
        ΙN
                 NS
                         ns.example.com.
        ΤN
                         192.168.1.10
ß
                 Δ
@
        IN
                 AAAA
                          ::1
                         192.168.1.10
ns
        ΙN
                 А
```

You must increment the *Serial Number* every time you make changes to the zone file. If you make multiple changes before restarting BIND9, simply increment the Serial once.

Now, you can add DNS records to the bottom of the zone file. See Common Record Types for details.

Note

Many admins like to use the last date edited as the serial of a zone, such as 2020012100 which is yyyymmddss (where ss is the Serial Number)

Once you have made changes to the zone file BIND9 needs to be restarted for the changes to take effect:

sudo systemctl restart bind9.service

Reverse Zone File

Now that the zone is setup and resolving names to IP Addresses, a *Reverse zone* needs to be added to allows DNS to resolve an address to a name.

Edit /etc/bind/named.conf.local and add the following:

```
zone "1.168.192.in-addr.arpa" {
   type master;
   file "/etc/bind/db.192";
```

};

Note

Replace 1.168.192 with the first three octets of whatever network you are using. Also, name the zone file /etc/bind/db.192 appropriately. It should match the first octet of your network.

Now create the /etc/bind/db.192 file:

sudo cp /etc/bind/db.127 /etc/bind/db.192

Next edit /etc/bind/db.192 changing the same options as /etc/bind/db.example.com:

```
; BIND reverse data file for local 192.168.1.XXX net
```

;

:

\$TTL	604800			
0	IN	SOA	ns.example.com.	root.example.com. (
			2	; Serial
			604800	; Refresh
			86400	; Retry
			2419200	; Expire
			604800)	; Negative Cache TTL
;				
0	IN	NS	ns.	
10	IN	PTR	ns.example.com.	

The *Serial Number* in the Reverse zone needs to be incremented on each change as well. For each *A record* you configure in /etc/bind/db.example.com, that is for a different address, you need to create a *PTR record* in /etc/bind/db.192.

After creating the reverse zone file restart BIND9:

sudo systemctl restart bind9.service

Secondary Server

Once a *Primary Server* has been configured a *Secondary Server* is highly recommended in order to maintain the availability of the domain should the Primary become unavailable.

First, on the Primary server, the zone transfer needs to be allowed. Add the allow-transfer option to the example Forward and Reverse zone definitions in /etc/bind/named.conf.local:

```
zone "example.com" {
   type master;
   file "/etc/bind/db.example.com";
   allow-transfer { 192.168.1.11; };
};
zone "1.168.192.in-addr.arpa" {
   type master;
   file "/etc/bind/db.192";
   allow-transfer { 192.168.1.11; };
};
```

Note

Replace 192.168.1.11 with the IP Address of your Secondary nameserver.

Restart BIND9 on the Primary server:

```
sudo systemctl restart bind9.service
```

Next, on the Secondary server, install the bind9 package the same way as on the Primary. Then edit the /etc/bind/named.conf.local and add the following declarations for the Forward and Reverse zones:

```
zone "example.com" {
   type secondary;
   file "db.example.com";
   masters { 192.168.1.10; };
};
zone "1.168.192.in-addr.arpa" {
   type secondary;
   file "db.192";
   masters { 192.168.1.10; };
};
```

Note

Replace 192.168.1.10 with the IP Address of your Primary nameserver.

Restart BIND9 on the Secondary server:

```
sudo systemctl restart bind9.service
```

In /var/log/syslog you should see something similar to the following (some lines have been split to fit the format of this document):

client 192.168.1.10#39448: received notify for zone '1.168.192.in-addr.arpa' zone 1.168.192.in-addr.arpa/IN: Transfer started. transfer of '100.18.172.in-addr.arpa/IN' from 192.168.1.10#53: connected using 192.168.1.11#37531 zone 1.168.192.in-addr.arpa/IN: transferred serial 5 transfer of '100.18.172.in-addr.arpa/IN' from 192.168.1.10#53: Transfer completed: 1 messages, 6 records, 212 bytes, 0.002 secs (106000 bytes/sec) zone 1.168.192.in-addr.arpa/IN: sending notifies (serial 5) client 192.168.1.10#20329: received notify for zone 'example.com' zone avample com/IN: Transfer eterted

zone example.com/IN: Transfer started. transfer of 'example.com/IN' from 192.168.1.10#53: connected using 192.168.1.11#38577 zone example.com/IN: transferred serial 5 transfer of 'example.com/IN' from 192.168.1.10#53: Transfer completed: 1 messages, 8 records, 225 bytes, 0.002 secs (112500 bytes/sec)

Note

Note: A zone is only transferred if the *Serial Number* on the Primary is larger than the one on the Secondary. If you want to have your Primary DNS notifying other Secondary DNS Servers of zone changes, you can add also-notify { ipaddress; }; to /etc/bind/named.conf.local as shown in the example below:

```
zone "example.com" {
   type master;
   file "/etc/bind/db.example.com";
   allow-transfer { 192.168.1.11; };
   also-notify { 192.168.1.11; };
};
zone "1.168.192.in-addr.arpa" {
   type master;
   file "/etc/bind/db.192";
   allow-transfer { 192.168.1.11; };
   also-notify { 192.168.1.11; };
```

};

Note

The default directory for non-authoritative zone files is /var/cache/bind/. This directory is also configured in AppArmor to allow the named daemon to write to it. For more information on AppArmor see Security - AppArmor.

Troubleshooting

This section covers diagnosing problems with DNS and BIND9 configurations.

Testing

resolv.conf

The first step in testing BIND9 is to add the nameserver's IP Address to a hosts resolver. The Primary nameserver should be configured as well as another host to double check things. Refer to DNS client configuration for details on adding nameserver addresses to your network clients. In the end your nameserver line in /etc/resolv.conf should be pointing at 127.0.0.53 and you should have a search parameter for your domain. Something like this:

nameserver 127.0.0.53
search example.com

To check which DNS server your local resolver is using, run:

systemd-resolve --status

Note

You should also add the IP Address of the Secondary nameserver to your client configuration in case the Primary becomes unavailable.

\mathbf{dig}

If you installed the dnsutils package you can test your setup using the DNS lookup utility dig:

• After installing BIND9 use dig against the loopback interface to make sure it is listening on port 53. From a terminal prompt:

```
dig -x 127.0.0.1
```

You should see lines similar to the following in the command output:

```
;; Query time: 1 msec
;; SERVER: 192.168.1.10#53(192.168.1.10)
```

• If you have configured BIND9 as a *Caching* nameserver "dig" an outside domain to check the query time:

dig ubuntu.com

Note the query time toward the end of the command output:

;; Query time: 49 msec

After a second dig there should be improvement:

;; Query time: 1 msec

ping

Now to demonstrate how applications make use of DNS to resolve a host name use the ping utility to send an ICMP echo request:

ping example.com

This tests if the nameserver can resolve the name ns.example.com to an IP Address. The command output should resemble:

```
PING ns.example.com (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=0.800 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=64 time=0.813 ms
```

named-checkzone

A great way to test your zone files is by using the named-checkzone utility installed with the bind9 package. This utility allows you to make sure the configuration is correct before restarting BIND9 and making the changes live.

• To test our example Forward zone file enter the following from a command prompt:

named-checkzone example.com /etc/bind/db.example.com

If everything is configured correctly you should see output similar to:

zone example.com/IN: loaded serial 6
OK

• Similarly, to test the Reverse zone file enter the following:

named-checkzone 1.168.192.in-addr.arpa /etc/bind/db.192

The output should be similar to:

```
zone 1.168.192.in-addr.arpa/IN: loaded serial 3
OK
```

Note

The Serial Number of your zone file will probably be different.

Quick temporary query logging

With the rndc tool, you can quickly turn query logging on and off, without restarting the service or changing the configuration file.

To turn query logging *on*, run:

sudo rndc querylog on

Likewise, to turn it off, run:

sudo rndc querylog off

The logs will be sent to syslog and will show up in /var/log/syslog by default:

```
Jan 20 19:40:50 new-n1 named[816]: received control channel command 'querylog on'
Jan 20 19:40:50 new-n1 named[816]: query logging is now on
Jan 20 19:40:57 new-n1 named[816]: client @0x7f48ec101480 192.168.1.10#36139 (ubuntu.com): query: ubuntu.com IN A +E(0)K (
```

Note

The amount of logs generated by enabling querylog could be huge!

Logging

BIND9 has a wide variety of logging configuration options available, but the two main ones are *channel* and *category*, which configure where logs go, and what information gets logged, respectively.

If no logging options are configured the default configuration is:

logging {

```
category default { default_syslog; default_debug; };
category unmatched { null; };
```

};

Let's instead configure BIND9 to send *debug* messages related to DNS queries to a separate file.

We need to configure a *channel* to specify which file to send the messages to, and a *category*. In this example, the *category* will log all queries. Edit /etc/bind/named.conf.local and add the following:

logging {

```
channel query.log {
    file "/var/log/named/query.log";
    severity debug 3;
};
category queries { query.log; };
```

};

Note

The *debug* option can be set from 1 to 3. If a level isn't specified, level 1 is the default.

• Since the *named daemon* runs as the *bind* user the /var/log/named directory must be created and the ownership changed:

sudo mkdir /var/log/named
sudo chown bind:bind /var/log/named

• Now restart BIND9 for the changes to take effect:

```
sudo systemctl restart bind9.service
```

You should see the file /var/log/named/query.log fill with query information. This is a simple example of the BIND9 logging options. For coverage of advanced options see More Information.

References

Common Record Types

This section covers some of the most common DNS record types.

• A record: This record maps an IP Address to a hostname.

www IN A 192.168.1.12

• CNAME record: Used to create an alias to an existing A record. You cannot create a CNAME record pointing to another CNAME record.

web IN CNAME www

• MX record: Used to define where email should be sent to. Must point to an A record, not a CNAME.

@ IN MX 1 mail.example.com. mail IN A 192.168.1.13 • NS record: Used to define which servers serve copies of a zone. It must point to an A record, not a CNAME. This is where Primary and Secondary servers are defined.

0	IN	NS	ns.example.com.
0	IN	NS	ns2.example.com
ns	IN	А	192.168.1.10
ns2	IN	Α	192.168.1.11

More Information

- Upstream BIND9 Documentation
- DNS and BIND is a popular book now in it's fifth edition. There is now also a DNS and BIND on IPv6 book.
- A great place to ask for BIND9 assistance, and get involved with the Ubuntu Server community, is the #ubuntu-server IRC channel on freenode.

FTP Server

File Transfer Protocol (FTP) is a TCP protocol for downloading files between computers. In the past, it has also been used for uploading but, as that method does not use encryption, user credentials as well as data transferred in the clear and are easily intercepted. So if you are here looking for a way to upload and download files securely, see the OpenSSH documentation instead.

FTP works on a client/server model. The server component is called an *FTP daemon*. It continuously listens for FTP requests from remote clients. When a request is received, it manages the login and sets up the connection. For the duration of the session it executes any of commands sent by the FTP client.

Access to an FTP server can be managed in two ways:

- Anonymous
- Authenticated

In the Anonymous mode, remote clients can access the FTP server by using the default user account called "anonymous" or "ftp" and sending an email address as the password. In the Authenticated mode a user must have an account and a password. This latter choice is very insecure and should not be used except in special circumstances. If you are looking to transfer files securely see SFTP in the section on OpenSSH-Server. User access to the FTP server directories and files is dependent on the permissions defined for the account used at login. As a general rule, the FTP daemon will hide the root directory of the FTP server and change it to the FTP Home directory. This hides the rest of the file system from remote sessions.

vsftpd - FTP Server Installation

vsftpd is an FTP daemon available in Ubuntu. It is easy to install, set up, and maintain. To install vsftpd you can run the following command:

sudo apt install vsftpd

Anonymous FTP Configuration

By default vsftpd is *not* configured to allow a nonymous download. If you wish to enable a nonymous download edit /etc/vsftpd.conf by changing:

anonymous_enable=YES

During installation a ftp user is created with a home directory of /srv/ftp. This is the default FTP directory.

If you wish to change this location, to /srv/files/ftp for example, simply create a directory in another location and change the *ftp* user's home directory:

sudo mkdir -p /srv/files/ftp
sudo usermod -d /srv/files/ftp ftp

After making the change restart vsftpd:

sudo systemctl restart vsftpd.service

Finally, copy any files and directories you would like to make available through anonymous FTP to /srv/files/ftp, or /srv/ftp if you wish to use the default.

User Authenticated FTP Configuration

By default vsftpd is configured to authenticate system users and allow them to download files. If you want users to be able to upload files, edit /etc/vsftpd.conf:

write_enable=YES

Now restart vsftpd:

sudo systemctl restart vsftpd.service

Now when system users login to FTP they will start in their *home* directories where they can download, upload, create directories, etc.

Similarly, by default, anonymous users are not allowed to upload files to FTP server. To change this setting, you should uncomment the following line, and restart vsftpd:

anon_upload_enable=YES

Warning

Enabling anonymous FTP upload can be an extreme security risk. It is best to not enable anonymous upload on servers accessed directly from the Internet.

The configuration file consists of many configuration parameters. The information about each parameter is available in the configuration file. Alternatively, you can refer to the man page, man 5 vsftpd.conf for details of each parameter.

Securing FTP

There are options in /etc/vsftpd.conf to help make vsftpd more secure. For example users can be limited to their home directories by uncommenting:

chroot_local_user=YES

You can also limit a specific list of users to just their home directories:

```
chroot_list_enable=YES
chroot_list_file=/etc/vsftpd.chroot_list
```

After uncommenting the above options, create a /etc/vsftpd.chroot_list containing a list of users one per line. Then restart vsftpd:

sudo systemctl restart vsftpd.service

Also, the /etc/ftpusers file is a list of users that are *disallowed* FTP access. The default list includes root, daemon, nobody, etc. To disable FTP access for additional users simply add them to the list.

FTP can also be encrypted using *FTPS*. Different from *SFTP*, *FTPS* is FTP over Secure Socket Layer (SSL). *SFTP* is a FTP like session over an encrypted *SSH* connection. A major difference is that users of SFTP need to have a *shell* account on the system, instead of a *nologin* shell. Providing all users with a shell may not be ideal for some environments, such as a shared web host. However, it is possible to restrict such accounts to only SFTP and disable shell interaction.

To configure *FTPS*, edit /etc/vsftpd.conf and at the bottom add:

ssl_enable=YES

Also, notice the certificate and key related options:

```
rsa_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
rsa_private_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
```

By default these options are set to the certificate and key provided by the ssl-cert package. In a production environment these should be replaced with a certificate and key generated for the specific host. For more information on certificates see Security - Certificates.

Now restart vsftpd, and non-anonymous users will be forced to use FTPS:

sudo systemctl restart vsftpd.service

To allow users with a shell of /usr/sbin/nologin access to FTP, but have no shell access, edit /etc/shells adding the *nologin* shell:

/etc/shells: valid login shells
/bin/csh
/bin/sh

/usr/bin/es /usr/bin/ksh /bin/ksh /usr/bin/rc /usr/bin/tcsh /bin/tcsh /bin/dash /bin/dash /bin/bash /bin/rbash /usr/bin/screen /usr/sbin/nologin

This is necessary because, by default vsftpd uses PAM for authentication, and the /etc/pam.d/vsftpd configuration file contains:

auth required pam_shells.so

The shells PAM module restricts access to shells listed in the /etc/shells file.

Most popular FTP clients can be configured to connect using FTPS. The lftp command line FTP client has the ability to use FTPS as well.

References

• See the vsftpd website for more information.

iSCSI Initiator (or Client)

Wikipedia iSCSI Definition:

iSCSI an acronym for **Internet Small Computer Systems Interface**, an Internet Protocol (IP)-based storage networking standard for linking data storage facilities. It provides block-level access to storage devices by carrying SCSI commands over a TCP/IP network.

iSCSI is used to facilitate data transfers over intranets and to manage storage over long distances. It can be used to transmit data over local area networks (LANs), wide area networks (WANs), or the Internet and can enable location-independent data storage and retrieval.

The protocol allows clients (called *initiators*) to send SCSI commands (CDBs) to storage devices (*targets*) on remote servers. It is a storage area network (SAN) protocol, allowing organizations to consolidate storage into storage arrays while providing clients (such as database and web servers) with the illusion of locally attached SCSI disks.

It mainly competes with Fibre Channel, but unlike traditional Fibre Channel, which usually requires dedicated cabling, iSCSI can be run over long distances using existing network infrastructure.

Ubuntu Server can be configured as both: **iSCSI initiator** and **iSCSI target**. This guide provides commands and configuration options to setup an **iSCSI initiator** (or Client).

Note: It is assumed that **you already have an iSCSI target on your local network** and have the appropriate rights to connect to it. The instructions for setting up a target vary greatly between hardware providers, so consult your vendor documentation to configure your specific iSCSI target.

Network Interfaces Configuration

ethernets:

Before start configuring iSCSI, make sure to have the network interfaces correctly set and configured in order to have open-iscsi package to behave appropriately, specially during boot time. In Ubuntu 20.04 LTS, the default network configuration tool is netplan.io.

For all the iSCSI examples bellow please consider the following netplan configuration for my iSCSI initiator:

/etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
{config: disabled}
/etc/netplan/50-cloud-init.yaml
network:

```
enp5s0:
        match:
            macaddress: 00:16:3e:af:c4:d6
        set-name: eth0
        dhcp4: true
        dhcp-identifier: mac
    enp6s0:
        match:
            macaddress: 00:16:3e:50:11:9c
        set-name: iscsi01
        dhcp4: true
        dhcp-identifier: mac
        dhcp4-overrides:
          route-metric: 300
    enp7s0:
        match:
            macaddress: 00:16:3e:b3:cc:50
        set-name: iscsi02
        dhcp4: true
        dhcp-identifier: mac
        dhcp4-overrides:
          route-metric: 300
version: 2
renderer: networkd
```

With this configuration, the interfaces names change by matching their mac addresses. This makes it easier to manage them in a server containing multiple interfaces.

From this point and beyond, 2 interfaces are going to be mentioned: **iscsi01** and **iscsi02**. This helps to demonstrate how to configure iSCSI in a multipath environment as well (check the Device Mapper Multipath session in this same Server Guide).

If you have only a single interface for the iSCSI network, make sure to follow the same instructions, but only consider the **iscsi01** interface command line examples.

iSCSI Initiator Install

To configure Ubuntu Server as an iSCSI initiator install the open-iscsi package. In a terminal enter:

\$ sudo apt install open-iscsi

Once the package is installed you will find the following files:

```
• /etc/iscsi/iscsid.conf
```

/etc/iscsi/initiatorname.iscsi

iSCSI Initiator Configuration

Configure the main configuration file like the example bellow:

```
/etc/iscsi/iscsid.conf
### startup settings
## will be controlled by systemd, leave as is
iscsid.startup = /usr/sbin/iscsidnode.startup = manual
### chap settings
# node.session.auth.authmethod = CHAP
## authentication of initiator by target (session)
# node.session.auth.username = username
# node.session.auth.password = password
# discovery.sendtargets.auth.authmethod = CHAP
```

```
## authentication of initiator by target (discovery)
# discovery.sendtargets.auth.username = username
# discovery.sendtargets.auth.password = password
### timeouts
## control how much time iscsi takes to propagate an error to the
## upper layer. if using multipath, having 0 here is desirable
## so multipath can handle path errors as quickly as possible
## (and decide to queue or not if missing all paths)
node.session.timeo.replacement_timeout = 0
node.conn[0].timeo.login timeout = 15
node.conn[0].timeo.logout timeout = 15
## interval for a NOP-Out request (a ping to the target)
node.conn[0].timeo.noop_out_interval = 5
## and how much time to wait before declaring a timeout
node.conn[0].timeo.noop_out_timeout = 5
## default timeouts for error recovery logics (lu & tgt resets)
node.session.err_timeo.abort_timeout = 15
node.session.err_timeo.lu_reset_timeout = 30
node.session.err timeo.tgt reset timeout = 30
### retry
node.session.initial login retry max = 8
### session and device queue depth
node.session.cmds max = 128
node.session.queue depth = 32
```

```
### performance
```

node.session.xmit_thread_priority = -20

and re-start the iSCSI daemon:

```
$ systemctl restart iscsid.service
```

This will set basic things up for the rest of configuration.

The other file mentioned:

/etc/iscsi/initiatorname.iscsi

InitiatorName=iqn.2004-10.com.ubuntu:01:60f3517884c3

contains this node's initiator name and is generated during open-iscsi package installation. If you modify this setting, make sure that you don't have duplicates in the same iSCSI SAN (Storage Area Network).

iSCSI Network Configuration

Before configuring the Logical Units that are going to be accessed by the initiator, it is important to inform the iSCSI service what are the interfaces acting as paths.

A straightforward way to do that is by:

• configuring the following environment variables

```
$ iscsi01_ip=$(ip -4 -o addr show iscsi01 | sed -r 's:.* (([0-9]{1,3}\.){3}[0-9]{1,3})/.*:\1:')
$ iscsi02_ip=$(ip -4 -o addr show iscsi02 | sed -r 's:.* (([0-9]{1,3}\.){3}[0-9]{1,3})/.*:\1:')
$ iscsi01_mac=$(ip -o link show iscsi01 | sed -r 's:.*\s+link/ether (([0-f]{2}(\:|)){6}).*:\1:g')
$ iscsi02_mac=$(ip -o link show iscsi02 | sed -r 's:.*\s+link/ether (([0-f]{2}(\:|)){6}).*:\1:g')
```

- configuring ${\bf iscsi01}$ interface

```
$ sudo iscsiadm -m iface -I iscsi01 --op=new
New interface iscsi01 added
$ sudo iscsiadm -m iface -I iscsi01 --op=update -n iface.hwaddress -v $iscsi01_mac
iscsi01 updated.
$ sudo iscsiadm -m iface -I iscsi01 --op=update -n iface.ipaddress -v $iscsi01_ip
iscsi01 updated.
```

• configuring **iscsi02** interface

```
$ sudo iscsiadm -m iface -I iscsi02 --op=new
New interface iscsi02 added
$ sudo iscsiadm -m iface -I iscsi02 --op=update -n iface.hwaddress -v $iscsi02_mac
iscsi02 updated.
$ sudo iscsiadm -m iface -I iscsi02 --op=update -n iface.ipaddress -v $iscsi02_ip
iscsi02 updated.
```

• discovering the **targets**

\$ sudo iscsiadm -m discovery -I iscsi01 --op=new --op=del --type sendtargets --portal storage.iscsi01
10.250.94.99:3260,1 iqn.2003-01.org.linux-iscsi.storage.x8664:sn.2c084c8320ca

\$ sudo iscsiadm -m discovery -I iscsi02 --op=new --op=del --type sendtargets --portal storage.iscsi02 10.250.93.99:3260,1 iqn.2003-01.org.linux-iscsi.storage.x8664:sn.2c084c8320ca

• configuring **automatic login**

\$ sudo iscsiadm -m node --op=update -n node.conn[0].startup -v automatic \$ sudo iscsiadm -m node --op=update -n node.startup -v automatic

• make sure needed **services** are enabled during OS initialization:

\$ systemctl enable open-iscsi

```
Synchronizing state of open-iscsi.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable open-iscsi
Created symlink /etc/systemd/system/iscsi.service → /lib/systemd/system/open-iscsi.service.
Created symlink /etc/systemd/system/sysinit.target.wants/open-iscsi.service → /lib/systemd/system/open-
iscsi.service.
```

\$ systemctl enable iscsid Synchronizing state of iscsid.service with SysV service script with /lib/systemd/systemd-sysv-install. Executing: /lib/systemd/systemd-sysv-install enable iscsid Created symlink /etc/systemd/system/sysinit.target.wants/iscsid.service → /lib/systemd/system/iscsid.service.

• restarting **iscsid** service

\$ systemctl restart iscsid.service

• and, finally, login in discovered logical units

\$ sudo iscsiadm -m node --loginall=automatic

Logging in to [iface: iscsi02, target: iqn.2003-01.org.linux-iscsi.storage.x8664:sn.2c084c8320ca, portal: 10.250.93.99,3 Logging in to [iface: iscsi01, target: iqn.2003-01.org.linux-iscsi.storage.x8664:sn.2c084c8320ca, portal: 10.250.94.99,3 Login to [iface: iscsi02, target: iqn.2003-01.org.linux-iscsi.storage.x8664:sn.2c084c8320ca, portal: 10.250.93.99,3260] Login to [iface: iscsi01, target: iqn.2003-01.org.linux-iscsi.storage.x8664:sn.2c084c8320ca, portal: 10.250.94.99,3260]

Accessing the Logical Units (or LUNs)

Check dmesg to make sure that the new disks have been detected:

dmesg

```
LIO-ORG TCMU device >
                                                                         0002 PQ: 0 ANSI: 5
[ 166.840694] scsi 7:0:0:4: Direct-Access
  166.840892] scsi 8:0:0:4: Direct-Access
                                                                         0002 PQ: 0 ANSI: 5
[
                                             LIO-ORG TCMU device >
[ 166.841741] sd 7:0:0:4: Attached scsi generic sg2 type 0
[ 166.841808] sd 8:0:0:4: Attached scsi generic sg3 type 0
[ 166.842278] scsi 7:0:0:3: Direct-Access
                                           LIO-ORG TCMU device >
                                                                         0002 PQ: 0 ANSI: 5
[ 166.842571] scsi 8:0:0:3: Direct-Access
                                             LIO-ORG TCMU device >
                                                                         0002 PQ: 0 ANSI: 5
[ 166.843482] sd 8:0:0:3: Attached scsi generic sg4 type 0
[ 166.843681] sd 7:0:0:3: Attached scsi generic sg5 type 0
```

```
[ 166.843706] sd 8:0:0:4: [sdd] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
  166.843884] scsi 8:0:0:2: Direct-Access
                                            LIO-ORG TCMU device > 0002 PQ: 0 ANSI: 5
ſ
  166.843971] sd 8:0:0:4: [sdd] Write Protect is off
ſ
[ 166.843972] sd 8:0:0:4: [sdd] Mode Sense: 2f 00 00 00
[ 166.844127] scsi 7:0:0:2: Direct-Access LIO-ORG TCMU device > 0002 PQ: 0 ANSI: 5
[ 166.844232] sd 7:0:0:4: [sdc] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
[ 166.844421] sd 8:0:0:4: [sdd] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
  166.844566] sd 7:0:0:4: [sdc] Write Protect is off
[
[
  166.844568] sd 7:0:0:4: [sdc] Mode Sense: 2f 00 00 00
  166.844846] sd 8:0:0:2: Attached scsi generic sg6 type 0
[
[ 166.845147] sd 7:0:0:4: [sdc] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
[ 166.845188] sd 8:0:0:4: [sdd] Optimal transfer size 65536 bytes
[ 166.845527] sd 7:0:0:2: Attached scsi generic sg7 type 0
[ 166.845678] sd 8:0:0:3: [sde] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
[ 166.845785] scsi 8:0:0:1: Direct-Access
                                             LIO-ORG TCMU device > 0002 PQ: 0 ANSI: 5
  166.845799] sd 7:0:0:4: [sdc] Optimal transfer size 65536 bytes
[
  166.845931] sd 8:0:0:3: [sde] Write Protect is off
ſ
  166.845933] sd 8:0:0:3: [sde] Mode Sense: 2f 00 00 00
Γ
  166.846424] scsi 7:0:0:1: Direct-Access
                                             LIO-ORG TCMU device >
                                                                          0002 PQ: 0 ANSI: 5
1
  166.846552] sd 8:0:0:3: [sde] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
[
  166.846708] sd 7:0:0:3: [sdf] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
[
[ 166.847024] sd 8:0:0:1: Attached scsi generic sg8 type 0
  166.847029] sd 7:0:0:3: [sdf] Write Protect is off
[
  166.847031] sd 7:0:0:3: [sdf] Mode Sense: 2f 00 00 00
ſ
ſ
  166.847043] sd 8:0:0:3: [sde] Optimal transfer size 65536 bytes
  166.847133] sd 8:0:0:2: [sdg] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
ſ
[ 166.849212] sd 8:0:0:2: [sdg] Write Protect is off
[ 166.849214] sd 8:0:0:2: [sdg] Mode Sense: 2f 00 00 00
[ 166.849711] sd 7:0:0:3: [sdf] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
  166.849718] sd 7:0:0:1: Attached scsi generic sg9 type 0
[
  166.849721] sd 7:0:0:2: [sdh] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
[
  166.853296] sd 8:0:0:2: [sdg] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
[
  166.853721] sd 8:0:0:2: [sdg] Optimal transfer size 65536 bytes
[
  166.853810] sd 7:0:0:2: [sdh] Write Protect is off
[
ſ
  166.853812] sd 7:0:0:2: [sdh] Mode Sense: 2f 00 00 00
  166.854026] sd 7:0:0:3: [sdf] Optimal transfer size 65536 bytes
Γ
[ 166.854431] sd 7:0:0:2: [sdh] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
  166.854625] sd 8:0:0:1: [sdi] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
[
  166.854898] sd 8:0:0:1: [sdi] Write Protect is off
[
ſ
  166.854900] sd 8:0:0:1: [sdi] Mode Sense: 2f 00 00 00
  166.855022] sd 7:0:0:2: [sdh] Optimal transfer size 65536 bytes
[
[ 166.855465] sd 8:0:0:1: [sdi] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
[ 166.855578] sd 7:0:0:1: [sdj] 2097152 512-byte logical blocks: > (1.07 GB/1.00 GiB)
[ 166.855845] sd 7:0:0:1: [sdj] Write Protect is off
[ 166.855847] sd 7:0:0:1: [sdj] Mode Sense: 2f 00 00 00
  166.855978] sd 8:0:0:1: [sdi] Optimal transfer size 65536 bytes
[
  166.856305] sd 7:0:0:1: [sdj] Write cache: enabled, read cache: > enabled, doesn't support DPO or FUA
Γ
  166.856701] sd 7:0:0:1: [sdj] Optimal transfer size 65536 bytes
[
  166.859624] sd 8:0:0:4: [sdd] Attached SCSI disk
ſ
Γ
  166.861304] sd 7:0:0:4: [sdc] Attached SCSI disk
  166.864409] sd 8:0:0:3: [sde] Attached SCSI disk
[
[ 166.864833] sd 7:0:0:3: [sdf] Attached SCSI disk
[ 166.867906] sd 8:0:0:2: [sdg] Attached SCSI disk
[ 166.868446] sd 8:0:0:1: [sdi] Attached SCSI disk
  166.871588] sd 7:0:0:1: [sdj] Attached SCSI disk
[
  166.871773] sd 7:0:0:2: [sdh] Attached SCSI disk
[
```

In the output above you will find $8 \ge SCSI$ disks recognized. The storage server is mapping $4 \ge LUNs$ to this node, AND the node has $2 \ge PATHs$ to each LUN. The OS recognizes each path to each device as 1 SCSI device.

You will find different output depending on the storage server your node is mapping the LUNs from, and the amount of LUNs being mapped as well.

Although not the objective of this session, let's find the 4 mapped LUNs using multipath-tools.

You will find further details about multipath in "Device Mapper Multipathing" session of this same guide.

```
$ apt-get install multipath-tools
```

```
$ sudo multipath -r
```

```
$ sudo multipath -ll
mpathd (360014051a042fb7c41c4249af9f2cfbc) dm-3 LIO-ORG,TCMU device
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 7:0:0:4 sde 8:64 active ready running
 -+- policy='service-time 0' prio=1 status=enabled
  `- 8:0:0:4 sdc 8:32 active ready running
mpathc (360014050d6871110232471d8bcd155a3) dm-2 LI0-0RG,TCMU device
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 7:0:0:3 sdf 8:80 active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 8:0:0:3 sdd 8:48 active ready running
mpathb (360014051f65c6cb11b74541b703ce1d4) dm-1 LIO-ORG,TCMU device
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 7:0:0:2 sdh 8:112 active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 8:0:0:2 sdg 8:96 active ready running
mpatha (36001405b816e24fcab64fb88332a3fc9) dm-0 LIO-ORG,TCMU device
size=1.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=1 status=active
| `- 7:0:0:1 sdj 8:144 active ready running
`-+- policy='service-time 0' prio=1 status=enabled
  `- 8:0:0:1 sdi 8:128 active ready running
```

Now it is much easier to understand each recognized SCSI device and common paths to same LUNs in the storage server. With the output above one can easily see that:

- mpatha device (/dev/mapper/mpatha) is a multipath device for: /dev/sdj
 - -/dev/dsi
- mpathb device (/dev/mapper/mpathb) is a multipath device for:
 /dev/sdh
 - -/dev/dsg
- mpathc device (/dev/mapper/mpathc) is a multipath device for:
 - /dev/sdf
 - -/dev/sdd
- mpathd device (/dev/mapper/mpathd) is a multipath device for:
 - -/dev/sde
 - -/dev/sdc

Do not use this in production without checking appropriate multipath configuration options in the **Device Mapper Multipathing** session. The *default multipath configuration* is less than optimal for regular usage.

Finally, to access the LUN (or remote iSCSI disk) you will:

- If accessing through a single network interface:
 - access it through /dev/sdX where X is a letter given by the OS
- If accessing through multiple network interfaces:
 - configure multipath and access the device through /dev/mapper/X

For everything else, the created devices are block devices and all commands used with local disks should work the same way:

• Creating a partition:

\$ sudo fdisk /dev/mapper/mpatha

```
Welcome to fdisk (util-linux 2.34).
```

Changes will remain in memory only, until you decide to write them. Be careful before using the write command. Device does not contain a recognized partition table. Created a new DOS disklabel with disk identifier 0x92c0322a. Command (m for help): p Disk /dev/mapper/mpatha: 1 GiB, 1073741824 bytes, 2097152 sectors Units: sectors of 1 * 512 = 512 bytes Sector size (logical/physical): 512 bytes / 512 bytes I/O size (minimum/optimal): 512 bytes / 65536 bytes Disklabel type: dos Disk identifier: 0x92c0322a Command (m for help): n Partition type p primary (0 primary, 0 extended, 4 free) е extended (container for logical partitions) Select (default p): p Partition number (1-4, default 1): First sector (2048-2097151, default 2048): Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-2097151, default 2097151): Created a new partition 1 of type 'Linux' and of size 1023 MiB. Command (m for help): w The partition table has been altered. • Creating a filesystem: \$ sudo mkfs.ext4 /dev/mapper/mpatha-part1 mke2fs 1.45.5 (07-Jan-2020) Creating filesystem with 261888 4k blocks and 65536 inodes Filesystem UUID: cdb70b1e-c47c-47fd-9c4a-03db6f038988 Superblock backups stored on blocks: 32768, 98304, 163840, 229376 Allocating group tables: done Writing inode tables: done Creating journal (4096 blocks): done Writing superblocks and filesystem accounting information: done • Mounting the block device: \$ sudo mount /dev/mapper/mpatha-part1 /mnt • Accessing the data: \$ ls /mnt lost+found Make sure to read other important sessions in Ubuntu Server Guide to follow up with concepts explored in this one.

References

- 1. iscsid
- 2. iscsi.conf
- 3. iscsid.conf
- 4. iscsi.service
- 5. iscsid.service
 6. Open-iSCSI
- 7. Debian Open-iSCSI

Network File System (NFS)

NFS allows a system to share directories and files with others over a network. By using NFS, users and programs can access files on remote systems almost as if they were local files.

Some of the most notable benefits that NFS can provide are:

- Local workstations use less disk space because commonly used data can be stored on a single machine and still remain accessible to others over the network.
- There is no need for users to have separate home directories on every network machine. Home directories could be set up on the NFS server and made available throughout the network.
- Storage devices such as floppy disks, CDROM drives, and USB Thumb drives can be used by other machines on the network. This may reduce the number of removable media drives throughout the network.

Installation

At a terminal prompt enter the following command to install the NFS Server:

sudo apt install nfs-kernel-server

To start the NFS server, you can run the following command at a terminal prompt:

sudo systemctl start nfs-kernel-server.service

Configuration

You can configure the directories to be exported by adding them to the /etc/exports file. For example:

```
/srv *(ro,sync,subtree_check)
/home *.hostname.com(rw,sync,no_subtree_check)
/scratch *(rw,async,no_subtree_check,no_root_squash)
```

Make sure any custom mount points you're adding have been created (/srv and /home will already exist):

sudo mkdir /scratch

Apply the new config via:

sudo exportfs -a

You can replace * with one of the hostname formats. Make the hostname declaration as specific as possible so unwanted systems cannot access the NFS mount. Be aware that *.hostname.com will match foo.hostname.com but not foo.bar.my-domain.com.

The *sync/async* options control whether changes are gauranteed to be committed to stable storage before replying to requests. *async* thus gives a performance benefit but risks data loss or corruption. Even though *sync* is the default, it's worth setting since exports will issue a warning if it's left unspecified.

subtree_check and *no_subtree_check* enables or disables a security verification that subdirectories a client attempts to mount for an exported filesystem are ones they're permitted to do so. This verification step has some performance implications for some use cases, such as home directories with frequent file renames. Read-only filesystems are more suitable to enable *subtree_check* on. Like with sync, exports will warn if it's left unspecified.

There are a number of optional settings for NFS mounts for tuning performance, tightening security, or providing conveniences. These settings each have their own trade-offs so it is important to use them with care, only as needed for the particular use case. *no_root_squash*, for example, adds a convenience to allow root-owned files to be modified by any client system's root user; in a multi-user environment where executables are allowed on a shared mount point, this could lead to security problems.

NFS Client Configuration

To enable NFS support on a client system, enter the following command at the terminal prompt:

sudo apt install nfs-common

Use the mount command to mount a shared NFS directory from another machine, by typing a command line similar to the following at a terminal prompt:

```
sudo mkdir /opt/example
sudo mount example.hostname.com:/srv /opt/example
```

Warning

The mount point directory /opt/example must exist. There should be no files or subdirectories in the /opt/example directory, else they will become inaccessible until the nfs filesystem is unmounted.

An alternate way to mount an NFS share from another machine is to add a line to the /etc/fstab file. The line must state the hostname of the NFS server, the directory on the server being exported, and the directory on the local machine where the NFS share is to be mounted.

The general syntax for the line in /etc/fstab file is as follows:

example.hostname.com:/srv /opt/example nfs rsize=8192,wsize=8192,timeo=14,intr

Advanced Configuration

NFS is comprised of several services, both on the server and the client. Each one of these services can have its own default configuration, and depending on the Ubuntu Server release you have installed, this configuration is done in different files, and with a different syntax.

Ubuntu Server 22.04 LTS ("jammy")

All NFS related services read a single configuration file: /etc/nfs.conf. This is a INI-style config file, see the nfs.conf(5) manpage for details. Furthermore, there is a /etc/nfs.conf.d directory which can hold *.conf snippets that can override settings from previous snippets or from the nfs.conf main config file itself.

There is a new command-line tool called nfsconf(8) which can be used to query or even set configuration parameters in nfs.conf. In particular, it has a --dump parameter which will show the effective configuration including all changes done by /etc/nfs.conf.d/*.conf snippets.

For Ubuntu Server 20.04 LTS ("focal") and earlier

Earlier Ubuntu releases use the traditional configuration mechanism for the NFS services via /etc/defaults/ configuration files. These are /etc/default/nfs-common and /etc/default/nfs/kernel-server, and are used basically to adjust the command-line options given to each daemon.

Each file has a small explanation about the available settings.

Warning

The NEED_* parameters have no effect on systemd-based installations, like Ubuntu 20.04 LTS ("focal") and Ubuntu 18.04 LTS ("bionic").

In those systems, to control whether a service should be running or not, use systemctl enable or systemctl disable, respectively.

Upgrading to Ubuntu 22.04 LTS ("jammy")

The main change to the NFS packages in Ubuntu 22.04 LTS ("jammy") is the configuration file. Instead of multiple files sourced by startup scripts from /etc/default/nfs-*, now there is one main configuration file in /etc/nfs.conf, with an INI-style syntax.

When upgrading to Ubuntu 22.04 LTS ("jammy") from a release that still uses the /etc/defaults/nfs-* configuration files, the following will happen:

- a default /etc/nfs.conf configuration file will be installed
- if the /etc/default/nfs-* files have been modified, a conversion script will be run and it will create /etc/nfs.conf.d/local.conf with the local modifications.

If this conversion script fails, then the package installation will fail. This can happen if the /etc/default/nfs-* files have an option that the conversion script wasn't prepared to handle, or a syntax error for example. In such cases, please file a bug using this link: https://bugs.launchpad.net/ubuntu/+source/nfs-utils/+filebug

You can run the conversion tool manually to gather more information about the error: it's in /usr/share/nfs-common/nfsconvert.py and must be run as root.

If all goes well, as it should in most cases, the system will have /etc/nfs.conf with the defaults, and /etc/nfs.conf.d/local.conf with the changes. You can merge these two together manually, and then delete local.conf, or leave it as is. Just keep in mind that /etc/nfs.conf is not the whole story: always inspect /etc/nfs.conf.d as well, as it may contain files overriding the defaults.

You can always run nfsconf --dump to check the final settings, as it merges together all configuration files and shows the resulting non-default settings.

Restarting NFS services

Since NFS is comprised of several individual services, it can be difficult to determine what to restart after a certain configuration change.

The tables below summarize all available services, which "meta" service they are linked to, and which configuration file each service uses.

Service	nfs-utils.service	nfs-server.service	config file (22.04)	config file (< 22.04) /etc/default/nfs-*
nfs-blkmap	PartOf		nfs.conf	
nfs-mountd		BindsTo	nfs.conf	nfs-kernel-server
nfsdcld				
nfs-idmapd		BindsTo	nfs.conf, idmapd.conf	idmapd.conf
rpc-gssd	PartOf		nfs.conf	
rpc-statd	PartOf		nfs.conf	nfs-common
$\operatorname{rpc-svcgssd}$	PartOf	BindsTo	nfs.conf	nfs-kernel-server

For example, systemctl restart nfs-server.service will restart nfs-mountd, nfs-idmapd and rpc-svcgssd (if running). On the other hand, restarting nfs-utils.service will restart nfs-blkmap, rpc-gssd, rpc-statd and rpc-svcgssd.

Of course, each service can still be individually restarted with the usual systemctl restart <service>.

The nfs.systemd(7) manpage has more details on the several systemd units available with the NFS packages.

NFSv4 with Kerberos

Kerberos with NFS adds an extra layer of security on top of NFS. It can be just a stronger authentication mechanism, or it can also be used to sign and encrypt the NFS traffic.

This section will assume you already have setup a Kerberos server, with a running KDC and admin services. Setting that up is explained elsewhere in the Ubuntu Server Guide.

NFS server (using kerberos)

The NFS server will have the usual nfs-kernel-server package and its dependencies, but we will also have to install kerberos packages. The kerberos packages are not strictly necessary, as the necessary keys can be copied over from the KDC, but it makes things much easier.

For this example, we will use:

- .vms DNS domain
- VMS Kerberos realm
- j-nfs-server.vms for the NFS server
- j-nfs-client.vms for the NFS client
- ubuntu/admin principal has admin privileges on the KDC

Adjust these names according to your setup.

First, install the krb5-user package:

sudo apt install krb5-user

Then, with an admin principal, let's create a key for the NFS server:

\$ sudo kadmin -p ubuntu/admin -q "addprinc -randkey nfs/j-nfs-server.vms"

And extract the key into the local keytab:

\$ sudo kadmin -p ubuntu/admin -q "ktadd nfs/j-nfs-server.vms" Authenticating as principal ubuntu/admin with password. Password for ubuntu/admin@VMS: Entry for principal nfs/j-nfs-server.vms with kvno 2, encryption type aes256-cts-hmac-shal-96 added to keytab FILE:/etc/k Entry for principal nfs/j-nfs-server.vms with kvno 2, encryption type aes128-cts-hmac-shal-96 added to keytab FILE:/etc/k

Confirm the key is available:

\$ sudo klist -k Keytab name: FILE:/etc/krb5.keytab KVNO Principal 2 nfs/j-nfs-server.vms@VMS

2 nfs/j-nfs-server.vms@VMS

Now install the NFS server:

\$ sudo apt install nfs-kernel-server

This will already automatically start the kerberos-related nfs services, because of the presence of /etc/krb5.keytab.

Now populate /etc/exports, restricting the exports to krb5 authentication. For example, exporting /storage using krb5p:

/storage *(rw,sync,no_subtree_check,sec=krb5p)

Refresh the exports:

\$ sudo exportfs -rav
exporting *:/storage

The security options are explained in the exports(5) manpage, but generally they are:

- krb5: use kerberos for authentication only (non-auth traffic is in clear text)
- krb5i: use kerberos for authentication and integrity checks (non-auth traffic is in clear text)
- krb5p: use kerberos for authentication, integrity and privacy protection (non-auth traffic is encrypted)

NFS client (using kerberos)

The NFS client has a similar set of steps. First we will prepare the client's keytab, so that when we install the NFS client package it will start the extra kerberos services automatically just by detecting the presence of the keytab:

sudo apt install krb5-user

To allow the **root** user to mount NFS shares via kerberos without a password, we have to create a host key for the NFS client:

sudo kadmin -p ubuntu/admin -q "addprinc -randkey host/j-nfs-client.vms"

And extract it:

\$ sudo kadmin -p ubuntu/admin -q "ktadd host/j-nfs-client.vms"

Now install the NFS client package:

```
$ sudo apt install nfs-common
```

And you should be able to do your first NFS kerberos mount:

```
$ sudo mount j-nfs-server:/storage /mnt
```

If you are using a machine credential, then the above mount will work without having a kerberos ticket, i.e., klist will show no tickets:

```
# mount j-nfs-server:/storage /mnt
# ls -l /mnt/*
-rw-r--r-- 1 root root 0 Apr 5 14:50 /mnt/hello-from-nfs-server.txt
# klist
klist: No credentials cache found (filename: /tmp/krb5cc_0)
```

Notice the above was done with root. Let's try accessing that existing mount with the ubuntu user, without acquiring a kerberos ticket:

```
# sudo -u ubuntu -i
$ ls -l /mnt/*
ls: cannot access '/mnt/*': Permission denied
```

The ubuntu user will only be able to access that mount if they have a kerberos ticket:

```
$ kinit
Password for ubuntu@VMS:
$ ls -l /mnt/*
-rw-r--r-- 1 root root 0 Apr 5 14:50 /mnt/hello-from-nfs-server.txt
```

And now we have not only the TGT, but also a ticket for the NFS service:

\$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: ubuntu@VMS

Valid starting Expires Service principal 04/05/22 17:48:50 04/06/22 03:48:50 krbtgt/VMS@VMS renew until 04/06/22 17:48:48 04/05/22 17:48:52 04/06/22 03:48:50 nfs/j-nfs-server.vms@ renew until 04/06/22 17:48:48 Ticket server: nfs/j-nfs-server.vms@VMS

One drawback of using a machine credential for mounts done by the root user is that you need a persistent secret (the /etc/krb5.keytab file) in the filesystem. Some sites may not allow such a persistent secret to be stored in the filesystem. An alternative is to use rpc.gssds -n option. From rpc.gssd(8):

• -n: when specified, UID 0 is forced to obtain user credentials which are used instead of the local system's machine credentials.

When this option is enabled and rpc.gssd restarted, then even the root user will need to obtain a kerberos ticket to perform an NFS kerberos mount.

Warning

Note that this prevents automatic NFS mounts via /etc/fstab, unless a kerberos ticket is obtained before.

In Ubuntu 22.04 LTS ("jammy"), this option is controlled in /etc/nfs.conf in the [gssd] section:

[gssd]

use-machine-creds=0

In older Ubuntu releases, the command line options for the rpc.gssd daemon are not exposed in /etc/default/nfs-common, therefore a systemd override file needs to be created. You can either run:

\$ sudo systemctl edit rpc-gssd.service

And paste the following into the editor that will open:

[Service] ExecStart= ExecStart=/usr/sbin/rpc.gssd \$GSSDARGS -n

Or manually create the file /etc/system/rpc-gssd.service.d/override.conf and any needed directories up to it, with the contents above.

After you restart the service with systemctl restart rpc-gssd.service, the root user won't be able to mount the NFS kerberos share without obtaining a ticket first.

References

Linux NFS wiki Linux NFS faq

Ubuntu Wiki NFS Howto Ubuntu Wiki NFSv4 Howto

OpenSSH Server

Introduction

OpenSSH is a powerful collection of tools for the remote control of, and transfer of data between, networked computers. You will also learn about some of the configuration settings possible with the OpenSSH server application and how to change them on your Ubuntu system.

OpenSSH is a freely available version of the Secure Shell (SSH) protocol family of tools for remotely controlling, or transferring files between, computers. Traditional tools used to accomplish these functions, such as telnet or rcp, are insecure and transmit the user's password in cleartext when used. OpenSSH provides a server daemon and client tools to facilitate secure, encrypted remote control and file transfer operations, effectively replacing the legacy tools.

The OpenSSH server component, sshd, listens continuously for client connections from any of the client tools. When a connection request occurs, sshd sets up the correct connection depending on the type of client tool connecting. For example, if the remote computer is connecting with the ssh client application, the OpenSSH server sets up a remote control session after authentication. If a remote user connects to an OpenSSH server with scp, the OpenSSH server daemon initiates a secure copy of files between the server and client after authentication. OpenSSH can use many authentication methods, including plain password, public key, and Kerberos tickets.

Installation

Installation of the OpenSSH client and server applications is simple. To install the OpenSSH client applications on your Ubuntu system, use this command at a terminal prompt:

sudo apt install openssh-client

To install the OpenSSH server application, and related support files, use this command at a terminal prompt:

sudo apt install openssh-server

Configuration

You may configure the default behavior of the OpenSSH server application, sshd, by editing the file /etc/ssh/sshd_config. For information about the configuration directives used in this file, you may view the appropriate manual page with the following command, issued at a terminal prompt:

man sshd_config

There are many directives in the sshd configuration file controlling such things as communication settings, and authentication modes. The following are examples of configuration directives that can be changed by editing the /etc/ssh/sshd_config file.

Tip

Prior to editing the configuration file, you should make a copy of the original file and protect it from writing so you will have the original settings as a reference and to reuse as necessary.

Copy the <code>/etc/ssh/sshd_config</code> file and protect it from writing with the following commands, issued at a terminal prompt:

sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.original sudo chmod a-w /etc/ssh/sshd_config.original

Furthermore since losing an ssh server might mean losing your way to reach a server, check the configuration after changing it and before restarting the server:

sudo sshd -t -f /etc/ssh/sshd_config

The following is an *example* of a configuration directive you may change:

• To make your OpenSSH server display the contents of the /etc/issue.net file as a pre-login banner, simply add or modify this line in the /etc/ssh/sshd_config file:

Banner /etc/issue.net

After making changes to the /etc/ssh/sshd_config file, save the file, and restart the sshd server application to effect the changes using the following command at a terminal prompt:

sudo systemctl restart sshd.service

Warning

Many other configuration directives for sshd are available to change the server application's behavior to fit your needs. Be advised, however, if your only method of access to a server is ssh, and you make a mistake in configuring sshd via the /etc/ssh/sshd_config file, you may find you are locked out of the server upon restarting it. Additionally, if an incorrect configuration directive is supplied, the sshd server may refuse to start, so be extra careful when editing this file on a remote server.

SSH Keys

SSH allow authentication between two hosts without the need of a password. SSH key authentication uses a *private* key and a *public key*.

To generate the keys, from a terminal prompt enter:

ssh-keygen -t rsa

This will generate the keys using the RSA Algorithm. At the time of this writing, the generated keys will have 3072 bits. You can modify the number of bits by using the -b option. For example, to generate keys with 4096 bits, you can do:

```
ssh-keygen -t rsa -b 4096
```

During the process you will be prompted for a password. Simply hit *Enter* when prompted to create the key.

By default the *public* key is saved in the file ~/.ssh/id_rsa.pub, while ~/.ssh/id_rsa is the *private* key. Now copy the id_rsa.pub file to the remote host and append it to ~/.ssh/authorized_keys by entering:

ssh-copy-id username@remotehost

Finally, double check the permissions on the authorized_keys file, only the authenticated user should have read and write permissions. If the permissions are not correct change them by:

```
chmod 600 .ssh/authorized_keys
```

You should now be able to SSH to the host without being prompted for a password.

Import keys from public keyservers

These days many users have already ssh keys registered with services like launchpad or github. Those can be easily imported with:

ssh-import-id <username-on-remote-service>

The prefix lp: is implied and means fetching from launchpad, the alternative gh: will make the tool fetch from github instead.

Two factor authentication with U2F/FIDO

OpenSSH 8.2 added support for U2F/FIDO hardware authentication devices. These devices are used to provide an extra layer of security on top of the existing key-based authentication, as the hardware token needs to be present to finish the authentication.

It's very simple to use and setup. The only extra step is generate a new keypair that can be used with the hardware device. For that, there are two key types that can be used: ecdsa-sk and ed25519-sk. The former has broader hardware support, while the latter might need a more recent device.

Once the keypair is generated, it can be used as you would normally use any other type of key in openssh. The only requirement is that in order to use the private key, the U2F device has to be present on the host.

For example, plug the U2F device in and generate a keypair to use with it:

```
$ ssh-keygen -t ecdsa-sk
Generating public/private ecdsa-sk key pair.
You may need to touch your authenticator to authorize key generation. <-- touch device
Enter file in which to save the key (/home/ubuntu/.ssh/id_ecdsa_sk):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_ecdsa_sk
Your public key has been saved in /home/ubuntu/.ssh/id_ecdsa_sk.pub
The key fingerprint is:
SHA256:V9PQ1MqaU8F0DXdHqDiH9Mxb8XK3o5aVYDQLV19IFRo ubuntu@focal
```

Now just transfer the public part to the server to $\sim/.ssh/authorized_keys$ and you are ready to go:

\$ ssh -i .ssh/id_ecdsa_sk ubuntu@focal.server Confirm user presence for key ECDSA-SK SHA256:V9PQ1MqaU8F0DXdHqDiH9Mxb8XK3o5aVYDQLVl9IFRo <-- touch device Welcome to Ubuntu Focal Fossa (GNU/Linux 5.4.0-21-generic x86_64) (...) ubuntu@focal.server:~\$

FIDO2 resident keys

FIDO2 private keys consist of two parts: a "key handle" part stored in the private key file on disk, and a per-device key that is unique to each FIDO2 token and that cannot be exported from the token hardware. These are combined by the hardware at authentication time to derive the real key that is used to

sign authentication challenges.

For tokens that are required to move between computers, it can be cumbersome to have to move the private key file first. To avoid this, tokens implementing the newer FIDO2 standard support *resident keys*, where it is possible to retrieve the key handle part of the key from the hardware.

Using resident keys increases the likelihood of an attacker being able to use a stolen token device. For this reason, tokens normally enforce PIN authentication before allowing download of keys, and users should set a PIN on their tokens before creating any resident keys. This is done via the hardware token management software.

OpenSSH allows resident keys to be generated using the ssh-keygen-O resident flag at key generation time. This will produce a public/private key pair as usual, but it will be possible to retrieve the private key part (the key handle) from the token later. This is done by running

\$ ssh-keygen -K

which will download the resident key from the token and write public/private key files for them. It is also possible to download and add resident keys directly to ssh-agent by running

\$ ssh-add -K

In this case no file is written, and the public key can be printed by running ssh-add -L.

References

- Ubuntu Wiki SSH page.
- OpenSSH Website
- OpenSSH 8.2 release notes
- Advanced OpenSSH Wiki Page

\mathbf{VPN}

OpenVPN is a Virtual Private Networking (VPN) solution provided in the Ubuntu Repositories. It is flexible, reliable and secure. It belongs to the family of SSL/TLS VPN stacks (different from IPSec VPNs). This chapter will cover installing and configuring OpenVPN to create a VPN.

If you want more than just pre-shared keys OpenVPN makes it easy to set up a Public Key Infrastructure (PKI) to use SSL/TLS certificates for authentication and key exchange between the VPN server and clients. OpenVPN can be used in a routed or bridged VPN mode and can be configured to use either UDP or TCP. The port number can be configured as well, but port 1194 is the official one; this single port is used for all communication. VPN client implementations are available for almost anything including all Linux distributions, OS X, Windows and OpenWRT based WLAN routers.

Server Installation

To install open ppn in a terminal enter:

sudo apt install openvpn easy-rsa

Public Key Infrastructure Setup

The first step in building an OpenVPN configuration is to establish a PKI (public key infrastructure). The PKI consists of:

- a separate certificate (also known as a public key) and private key for the server and each client.
- a master Certificate Authority (CA) certificate and key, used to sign the server and client certificates.

OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established.

Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server).

Certificate Authority Setup

To setup your own Certificate Authority (CA) and generate certificates and keys for an OpenVPN server and multiple clients first copy the easy-rsa directory to /etc/openvpn. This will ensure that any changes to the scripts will not be lost when the package is updated. From a terminal, run:

sudo make-cadir /etc/openvpn/easy-rsa

Note: If desired, you can alternatively edit /etc/openvpn/easy-rsa/vars directly, adjusting it to your needs.

As root user change to the newly created directory /etc/openvpn/easy-rsa and run:

./easyrsa init-pki
./easyrsa build-ca

Server Keys and Certificates

Next, we will generate a key pair for the server:

./easyrsa gen-req myservername nopass

Diffie Hellman parameters must be generated for the OpenVPN server. The following will place them in pki/dh.pem.

./easyrsa gen-dh

And finally a certificate for the server:

./easyrsa gen-req myservername nopass
./easyrsa sign-req server myservername

All certificates and keys have been generated in subdirectories. Common practice is to copy them to /etc/openvpn/:

cp pki/dh.pem pki/ca.crt pki/issued/myservername.crt pki/private/myservername.key /etc/openvpn/

Client Certificates

The VPN client will also need a certificate to authenticate itself to the server. Usually you create a different certificate for each client.

This can either be done on the server (as the keys and certificates above) and then securely distributed to the client. Or vice versa: the client can generate and submit a request that is sent and signed by the server.

To create the certificate, enter the following in a terminal while being user root:

```
./easyrsa gen-req myclient1 nopass
./easyrsa sign-req client myclient1
```

If the first command above was done on a remote system, then copy the .req file to the CA server. There you can then import it via easyrsa import-req /incoming/myclientl.req myclient1. Then you can go on with the second sign-eq command.

In both cases, afterwards copy the following files to the client using a secure method:

- pki/ca.crt
- pki/issued/myclient1.crt

As the client certificates and keys are only required on the client machine, you can remove them from the server.

Simple Server Configuration

Along with your OpenVPN installation you got these sample config files (and many more if you check):

root@server:/# ls -l /usr/share/doc/openvpn/examples/sample-config-files/ total 68 -rw-r--r-- 1 root root 3427 2011-07-04 15:09 client.conf -rw-r--r-- 1 root root 4141 2011-07-04 15:09 server.conf.gz

Start with copying and unpacking server.conf.gz to /etc/openvpn/server.conf.

sudo cp /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz /etc/openvpn/myserver.conf.gz sudo gzip -d /etc/openvpn/myserver.conf.gz

Edit /etc/openvpn/myserver.conf to make sure the following lines are pointing to the certificates and keys you created in the section above.

ca ca.crt
cert myservername.crt
key myservername.key
dh dh2048.pem

Complete this set with a ta key in etc/openvpn for tls-auth like:

sudo openvpn --genkey --secret ta.key

Edit /etc/sysctl.conf and uncomment the following line to enable IP forwarding.

#net.ipv4.ip_forward=1

Then reload sysctl.

sudo sysctl -p /etc/sysctl.conf

That is the minimum you have to configure to get a working OpenVPN server. You can use all the default settings in the sample server.conf file. Now start the server.

Be aware that the "systemctl start openvpn" is **not** starting your openvpn you just defined. Openvpn uses templatized systemd jobs, openvpn@CONFIGFILENAME. So if for example your configuration file is **myserver.conf** your service is called openvpn@myserver. You can run all kinds of service and systemctl commands like start/stop/enable/disable/preset against a templatized service like openvpn@server.

\$ sudo systemctl start openvpn@myserver

You will find logging and error messages in the journal. For example, if you started a templatized service openvpn@server you can filter for this particular message source with:

sudo journalctl -u openvpn@myserver -xe

The same templatized approach works for all of systemctl:

```
$ sudo systemctl status openvpn@myserver
openvpn@myserver.service - OpenVPN connection to myserver
   Loaded: loaded (/lib/systemd/system/openvpn@.service; disabled; vendor preset: enabled)
   Active: active (running) since Thu 2019-10-24 10:59:25 UTC; 10s ago
     Docs: man:openvpn(8)
           https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
           https://community.openvpn.net/openvpn/wiki/HOWTO
Main PID: 4138 (openvpn)
   Status: "Initialization Sequence Completed"
   Tasks: 1 (limit: 533)
   Memory: 1.0M
   CGroup: /system.slice/system-openvpn.slice/openvpn@myserver.service
               └─4138 /usr/sbin/openvpn --daemon ovpn-myserver --status /run/openvpn/myserver.status 10 --
cd /etc/openvpn --script-security 2 --config /etc/openvpn/myserver.conf --writepid /run/
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: /sbin/ip addr add dev tun0 local 10.8.0.1 peer 10.8.0.2
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: /sbin/ip route add 10.8.0.0/24 via 10.8.0.2
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: Could not determine IPv4/IPv6 protocol. Using AF INET
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: Socket Buffers: R=[212992->212992] S=[212992->212992]
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: UDPv4 link local (bound): [AF_INET][undef]:1194
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: UDPv4 link remote: [AF_UNSPEC]
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: MULTI: multi_init called, r=256 v=256
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: IFCONFIG POOL LIST
Oct 24 10:59:26 eoan-vpn-server ovpn-myserver[4138]: Initialization Sequence Completed
```

You can enable/disable various openvpn services on one system, but you could also let Ubuntu do it for you. There is config for AUTOSTART in /etc/default/openvpn. Allowed values are "all", "none" or space separated list of names of the VPNs. If empty, "all" is assumed. The VPN name refers to the VPN configuration file name. i.e. home would be /etc/openvpn/home.conf If you're running systemd, changing this variable will require running systemctl daemon-reload followed by a *restart* of the openvpn service (if you removed entries you may have to stop those manually).

After "systemctl daemon-reload" a restart of the "generic" openvpn will restart all dependent services that the generator in /lib/systemd/system-generators/openvpn-generator created for your conf files when you called daemon-reload.

Now check if OpenVPN created a tun0 interface:

root@server:/etc/openvpn# ip addr show dev tun0

5: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 100 link/none

inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
 valid_lft forever preferred_lft forever
inet6 fe80::b5ac:7829:f31e:32c5/64 scope link stable-privacy

Simple Client Configuration

There are various different OpenVPN client implementations with and without GUIs. You can read more about clients in a later section on VPN Clients. For now we use commandline/service based OpenVPN client for Ubuntu which is part of the very same package as the server. So you have to install the openvpn package again on the client machine:

sudo apt install openvpn

This time copy the client.conf sample config file to /etc/openvpn/:

sudo cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf /etc/openvpn/

Copy the following client keys and certificate files you created in the section above to e.g. /etc/openvpn/ and edit /etc/openvpn/client.conf to make sure the following lines are pointing to those files. If you have the files in /etc/openvpn/ you can omit the path.

ca ca.crt
cert myclient1.crt
key myclient1.key
tls-auth ta.key 1

And you have to specify the OpenVPN server name or address. Make sure the keyword client is in the config. That's what enables client mode.

client

```
remote vpnserver.example.com 1194
```

Now start the OpenVPN client with the same templatized mechanism:

```
$ sudo systemctl start openvpn@client
```

You can check status as you did on the server:

```
$ sudo systemctl status openvpn@client
openvpn@client.service - OpenVPN connection to client
  Loaded: loaded (/lib/systemd/system/openvpn@.service; disabled; vendor preset: enabled)
  Active: active (running) since Thu 2019-10-24 11:42:35 UTC; 6s ago
    Docs: man:openvpn(8)
          https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
          https://community.openvpn.net/openvpn/wiki/HOWTO
Main PID: 3616 (openvpn)
  Status: "Initialization Sequence Completed"
   Tasks: 1 (limit: 533)
  Memory: 1.3M
   CGroup: /system.slice/system-openvpn.slice/openvpn@client.service
                 cd /etc/openvpn --script-security 2 --config /etc/openvpn/client.conf --writepid /run/openvp
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256 bit ke
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256 bit ke
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: ROUTE_GATEWAY 192.168.122.1/255.255.255.0 IFACE=ens3 HWADDR=52:54:00:
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: TUN/TAP device tun0 opened
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: TUN/TAP TX queue length set to 100
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: /sbin/ip link set dev tun0 up mtu 1500
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: /sbin/ip addr add dev tun0 local 10.8.0.6 peer 10.8.0.5
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: /sbin/ip route add 10.8.0.1/32 via 10.8.0.5
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: WARNING: this configuration may cache passwords in memory -
- use the auth-nocache option to prevent this
Oct 24 11:42:36 eoan-vpn-client ovpn-client[3616]: Initialization Sequence Completed
On the server log an incoming connection looks like the following.
You can see client name and source address as well as success/failure messages.
ovpn-myserver[4818]: 192.168.122.114:55738 TLS: Initial packet from [AF_INET]192.168.122.114:55738, sid=5e943ab8 40ab9fe
ovpn-myserver[4818]: 192.168.122.114:55738 VERIFY OK: depth=1, CN=Easy-RSA CA
ovpn-myserver[4818]: 192.168.122.114:55738 VERIFY OK: depth=0, CN=myclient1
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV VER=2.4.7
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_PLAT=linux
```

```
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_PROTO=2
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV NCP=2
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_LZ4=1
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_LZ4v2=1
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_LZO=1
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV COMP STUB=1
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_COMP_STUBv2=1
ovpn-myserver[4818]: 192.168.122.114:55738 peer info: IV_TCPNL=1
ovpn-myserver[4818]: 192.168.122.114:55738 Control Channel: TLSv1.3, cipher TLSv1.3 TLS_AES_256_GCM_SHA384, 2048 bit RSA
ovpn-myserver[4818]: 192.168.122.114:55738 [myclient1] Peer Connection Initiated with [AF_INET]192.168.122.114:55738
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 MULTI_sva: pool returned IPv4=10.8.0.6, IPv6=(Not enabled)
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 MULTI: Learn: 10.8.0.6 -> myclient1/192.168.122.114:55738
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 MULTI: primary virtual IP for myclient1/192.168.122.114:55738: 10.8
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 PUSH: Received control message: 'PUSH REQUEST'
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 SENT CONTROL [myclient1]: 'PUSH_REPLY, route 10.8.0.1, topology net3
restart 120, if config 10.8.0.6 10.8.0.5, peer-id 0, cipher AES-256-GCM' (status=1)
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 Data Channel: using negotiated cipher 'AES-256-GCM'
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256 bi
ovpn-myserver[4818]: myclient1/192.168.122.114:55738 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256 bi
```

And you can check on the client if it created a tun0 interface:

```
$ ip addr show dev tun0
```

```
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 100
    link/none
    inet 10.8.0.6 peer 10.8.0.5/32 scope global tun0
        valid_lft forever preferred_lft forever
```

inet6 fe80::5a94:ae12:8901:5a75/64 scope link stable-privacy
 valid_lft forever preferred_lft forever

Check if you can ping the OpenVPN server:

```
root@client:/etc/openvpn# ping 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_req=1 ttl=64 time=0.920 ms
```

Note

The OpenVPN server always uses the first usable IP address in the client network and only that IP is pingable. E.g. if you configured a /24 for the client network mask, the .1 address will be used. The P-t-P address you see in the ip addr output above is usually not answering ping requests.

Check out your routes:

\$ ip route default via 192.168.122.1 dev ens3 proto dhcp src 192.168.122.114 metric 100 10.8.0.1 via 10.8.0.5 dev tun0 10.8.0.5 dev tun0 proto kernel scope link src 10.8.0.6 192.168.122.0/24 dev ens3 proto kernel scope link src 192.168.122.114 192.168.122.1 dev ens3 proto dhcp scope link src 192.168.122.114 metric 100

First trouble shooting

If the above didn't work for you, check this:

- Check your journal -xe
- Check that you have specified the keyfile names correctly in client and server conf files
- Can the client connect to the server machine? Maybe a firewall is blocking access? Check journal on server.
- Client and server must use same protocol and port, e.g. UDP port 1194, see port and proto config option
- Client and server must use same config regarding compression, see comp-lzo config option
- Client and server must use same config regarding bridged vs routed mode, see server vs server-bridge config option

Advanced configuration

Advanced routed VPN configuration on server

The above is a very simple working VPN. The client can access services on the VPN server machine through an encrypted tunnel. If you want to reach more servers or anything in other networks, push some routes to the clients. E.g. if your company's network can be summarized to the network 192.168.0.0/16, you could push this route to the clients. But you will also have to change the routing for the way back - your servers need to know a route to the VPN client-network.

The example config files that we have been using in this guide are full of all these advanced options in the form of a comment and a disabled configuration line as an example.

Note

Please read the OpenVPN hardening security guide for further security advice.

Advanced bridged VPN configuration on server

OpenVPN can be setup for either a routed or a bridged VPN mode. Sometimes this is also referred to as OSI layer-2 versus layer-3 VPN. In a bridged VPN all layer-2 frames - e.g. all ethernet frames - are sent to the VPN partners and in a routed VPN only layer-3 packets are sent to VPN partners. In bridged mode all traffic including traffic which was traditionally LAN-local like local network broadcasts, DHCP requests, ARP requests etc. are sent to VPN partners whereas in routed mode this would be filtered.

Prepare interface config for bridging on server

First, use netplan to configure a bridge device using the desired ethernet device.

```
$ cat /etc/netplan/01-netcfg.yaml
network:
    version: 2
    renderer: networkd
    ethernets:
        enp0s31f6:
            dhcp4: no
    bridges:
        br0:
            interfaces: [enp0s31f6]
            dhcp4: no
            addresses: [10.0.1.100/24]
            gateway4: 10.0.1.1
            nameservers:
            addresses: [10.0.1.1]
```

Static IP addressing is highly suggested. DHCP addressing can also work, but you will still have to encode a static address in the OpenVPN configuration file.

The next step on the server is to configure the ethernet device for promiscuous mode on boot. To do this, ensure the networkd-dispatcher package is installed and create the following configuration script.

```
sudo apt update
sudo apt install networkd-dispatcher
sudo touch /usr/lib/networkd-dispatcher/dormant.d/promisc_bridge
sudo chmod +x /usr/lib/networkd-dispatcher/dormant.d/promisc_bridge
Then add the following contents
```

Then add the following contents.

```
#!/bin/sh
set -e
if [ "$IFACE" = br0 ]; then
    # no networkd-dispatcher event for 'carrier' on the physical interface
    ip link set enp0s31f6 up promisc on
fi
```

Prepare server config for bridging

Edit /etc/openvpn/server.conf to use tap rather than tun and set the server to use the server-bridge directive:

;dev tun dev tap ;server 10.8.0.0 255.255.255.0 server-bridge 10.0.0.4 255.255.255.0 10.0.0.128 10.0.0.254

After configuring the server, restart openvpn by entering:

sudo systemctl restart openvpn@myserver

Prepare client config for bridging

The only difference on the client side for bridged mode to what was outlined above is that you need to edit /etc/openvpn/client.conf and set tap mode:

dev tap ;dev tun

Finally, restart openvpn:

sudo systemctl restart openvpn@client

You should now be able to connect to the full remote LAN through the VPN.

References

- EasyRSA
- OpenVPN quick start guide
- Snap'ed version of openvpn easy-openvpn
- Debians OpenVPN Guide

Gitolite

Installing a gitolite server

Gitolite provides a traditional source control management server for git, with multiple users and access rights management. gitolite can be installed with the following command:

sudo apt install gitolite3

Gitolite configuration

Configuration of the gitolite server is a little different that most other servers on Unix-like systems, in that gitolite stores its configuration in a git repository rather than in files in /etc/. The first step to configuring a new installation is therefore to allow access to the configuration repository.

First of all, let's create a user for gitolite to use for the service:

```
sudo adduser --system --shell /bin/bash --group --disabled-password --home /home/git git
```

Now we want to let gitolite know about the repository administrator's public SSH key. This assumes that the current user is the repository administrator. If you have not yet configured an SSH key, refer to openssh-keys in this manual.

cp ~/.ssh/id_rsa.pub /tmp/\$(whoami).pub

Let's switch to the git user and import the administrator's key into gitolite.

sudo su - git
gl-setup /tmp/*.pub

Gitolite will allow you to make initial changes to its configuration file during the setup process. You can now clone and modify the gitolite configuration repository from your administrator user (the user whose public SSH key you imported). Switch back to that user, then clone the configuration repository:

exit
git clone git@\$IP_ADDRESS:gitolite-admin.git
cd gitolite-admin

The gitolite-admin contains two subdirectories, "conf" and "keydir". The configuration files are in the conf dir, and the keydir directory contains the list of user's public SSH keys.

Managing gitolite users and repositories

Adding a new user to gitolite is simple: just obtain their public SSH key and add it to the keydir directory as \$DESIRED_USER_NAME.pub. Note that the gitolite usernames don't have to match the system usernames - they are only used in the gitolite configuration file to manage access control. Similarly, users are deleted by deleting their public key files. After each change, do not forget to commit the changes to git, and push the changes back to the server with

git commit -a git push origin master

Repositories are managed by editing the conf/gitolite.conf file. The syntax is space separated, and simply specifies the list of repositories followed by some access rules. The following is a default example

```
gitolite-admin
repo
        RW+
                 =
                     admin
        R
                 =
                      alice
repo
        project1
        RW+
                      alice
                 =
        RW
                      bob
                 =
        R
                      denise
                 =
```

Using your server

Once a user's public key has been imported by the gitolite admin and authorization granted to the user to one or more repositories, the user can access repositories with the following command:

```
git clone git@$SERVER_IP:$PR0JECT_NAME.git
```

To add the server as a new remote for an existing git repository:

```
git remote add gitolite git@$SERVER_IP:$PR0JECT_NAME.git
```

References

- Gitolite's code repository provides access to source code.
- Gitolite's documentation includes a "fool proof setup" guide and a cookbook with recipes for common tasks.
- Gitolite's maintainer has written a book, *Gitolite Essentials*, for more in-depth information about the software.
- General information about git itself can be found at the Git homepage.

VPN Client software implementations

Linux Network-Manager GUI for OpenVPN

Many Linux distributions including Ubuntu desktop variants come with Network Manager, a nice GUI to configure your network settings. It also can manage your VPN connections. It is the default, but if in doubt make sure you have package network-manager-openvpn installed.

Open the Network Manager GUI, select the VPN tab and then the 'Add' button. Select OpenVPN as the VPN type in the opening requester and press 'Create'. In the next window add the OpenVPN's server name as the 'Gateway', set 'Type' to 'Certificates (TLS)', point 'User Certificate' to your user certificate, 'CA Certificate' to your CA certificate and 'Private Key' to your private key file. Use the advanced button to enable compression (e.g. comp-lzo), dev tap, or other special settings you set on the server. Now try to establish your VPN.

OpenVPN with GUI for Mac OS X

• Tunnelblick is an excellent free, open source implementation of a GUI for OpenVPN for OS X. Download the latest OS X installer from there and install it. It also is recommended by upstream which would have a alternative on their own

Then put your client.ovpn config file together with the certificates and keys in /Users/username/Library/Application Support/Tunnelblick/Configurations/ and lauch Tunnelblick from your Application folder.

Instead of downloading manually, if you have brew set up on MacOS this is as easy as:

brew cask install tunnelblick
OpenVPN with GUI for Win

First download and install the latest OpenVPN Windows Installer. As of this writing, the management GUI is included with the Windows binary installer.

You need to start the OpenVPN service. Goto Start > Computer > Manage > Services and Applications > Services. Find the OpenVPN service and start it. Set it's startup type to automatic.

When you start the OpenVPN MI GUI the first time you need to run it as an administrator. You have to right click on it and you will see that option.

There is an updated guide by the upstream project for the client on Windows.

References

- See the OpenVPN website for additional information.
- OpenVPN hardening security guide
- Also, Pakt's OpenVPN: Building and Integrating Virtual Private Networks is a good resource.

Byobu

One of the most useful applications for any system administrator is an xtern multiplexor such as screen or tmux. It allows for the execution of multiple shells in one terminal. To make some of the advanced multiplexor features more user-friendly and provide some useful information about the system, the byobu package was created. It acts as a wrapper to these programs. By default Byobu is installed in Ubuntu server and it uses tmux (if installed) but this can be changed by the user.

Invoke it simply with:

byobu

Now bring up the configuration menu. By default this is done by pressing the F9 key. This will allow you to:

- Help Quick Start Guide
- Toggle status notifications
- Change the escape sequence
- Byobu currently does not launch at login (toggle on)

by obu provides a menu which displays the Ubuntu release, processor information, memory information, and the time and date. The effect is similar to a desktop menu.

Using the "Byobu currently does not launch at login (toggle on)" option will cause byobu to be executed any time a terminal is opened. Changes made to byobu are on a per user basis, and will not affect other users on the system.

One difference when using byobu is the *scrollback* mode. Press the F7 key to enter scrollback mode. Scrollback mode allows you to navigate past output using vi like commands. Here is a quick list of movement commands:

- h Move the cursor left by one character
- j Move the cursor down by one line
- k Move the cursor up by one line
- l Move the cursor right by one character
- θ Move to the beginning of the current line
- $\ensuremath{\mathcal{S}}$ Move to the end of the current line
- ${\cal G}$ Moves to the specified line (defaults to the end of the buffer)
- / Search forward
- ? Search backward
- n Moves to the next match, either forward or backward

Resources

- For more information on screen see the screen web site.
- And the Ubuntu Wiki screen page.
- Also, see the byobu project page for more information.

etckeeper

etckeeper allows the contents of /etc to be stored in a Version Control System (VCS) repository. It integrates with APT and automatically commits changes to /etc when packages are installed or upgraded. Placing /etc under version control is considered an industry best practice, and the goal of etckeeper is to make this process as painless as possible.

Install etckeeper by entering the following in a terminal:

sudo apt install etckeeper

The main configuration file, /etc/etckeeper/etckeeper.conf, is fairly simple. The main option is which VCS to use and by default etckeeper is configured to use git. The repository is automatically initialized (and committed for the first time) during package installation. It is possible to undo this by entering the following command:

sudo etckeeper uninit

By default, etckeeper will commit uncommitted changes made to /etc daily. This can be disabled using the AVOID_DAILY_AUTOCOMMITS configuration option. It will also automatically commit changes before and after package installation. For a more precise tracking of changes, it is recommended to commit your changes manually, together with a commit message, using:

sudo etckeeper commit "Reason for configuration change"

The vcs etckeeper command allows to run any subcommand of the VCS that etckeeper is configured to run. t will be run in /etc. For example, in the case of git:

sudo etckeeper vcs log /etc/passwd

To demonstrate the integration with the package management system (APT), install postfix:

sudo apt install postfix

When the installation is finished, all the postfix configuration files should be committed to the repository:

```
[master 5al6a0d] committing changes in /etc made by "apt install postfix"
Author: Your Name <xyz@example.com>
36 files changed, 2987 insertions(+), 4 deletions(-)
create mode 100755 init.d/postfix
create mode 100644 insserv.conf.d/postfix
create mode 100755 network/if-down.d/postfix
create mode 100755 network/if-up.d/postfix
create mode 100644 postfix/dynamicmaps.cf
create mode 100644 postfix/main.cf
create mode 100644 postfix/main.cf.proto
create mode 120000 postfix/makedefs.out
create mode 100644 postfix/master.cf
create mode 100644 postfix/master.cf.proto
create mode 100755 postfix/post-install
create mode 100644 postfix/postfix-files
create mode 100755 postfix/postfix-script
create mode 100755 ppp/ip-down.d/postfix
create mode 100755 ppp/ip-up.d/postfix
create mode 120000 rc0.d/K01postfix
create mode 120000 rcl.d/K01postfix
create mode 120000 rc2.d/S01postfix
create mode 120000 rc3.d/S01postfix
create mode 120000 rc4.d/S01postfix
create mode 120000 rc5.d/S01postfix
create mode 120000 rc6.d/K01postfix
create mode 100755 resolvconf/update-libc.d/postfix
create mode 100644 rsyslog.d/postfix.conf
create mode 120000 systemd/system/multi-user.target.wants/postfix.service
```

create mode 100644 ufw/applications.d/postfix

For an example of how etckeeper tracks manual changes, add new a host to /etc/hosts. Using git you can see which files have been modified:

sudo etckeeper vcs status
and how:
sudo etckeeper vcs diff
If you are happy with the changes you can now commit them:
sudo etckeeper commit "added new host"

Resources

- See the etckeeper site for more details on using etckeeper.
- For documentation on the git VCS tool see the Git website.

Munin

Installation

Before installing Munin on *server01* apache2 will need to be installed. The default configuration is fine for running a munin server. For more information see the Apache2 Web Server guide.

First, on server01 install munin. In a terminal enter:

sudo apt install munin

Now on server02 install the munin-node package:

sudo apt install munin-node

Configuration

On *server01* edit the /etc/munin/munin.conf adding the IP address for *server02*:

Note

Replace server02 and 172.18.100.101 with the actual hostname and IP address for your server.

Next, configure munin-node on server02. Edit /etc/munin/munin-node.conf to allow access by server01:

allow ^172\.18\.100\.100\$

Note

Replace 172.18.100.100\$ with IP address for your munin server.

Now restart munin-node on server02 for the changes to take effect:

sudo systemctl restart munin-node.service

Finally, in a browser go to http://server01/munin, and you should see links to nice graphs displaying information from the standard *munin-plugins* for disk, network, processes, and system.

Note

Since this is a new install it may take some time for the graphs to display anything useful.

Additional Plugins

The munin-plugins-extra package contains performance checks additional services such as DNS, DHCP, Samba, etc. To install the package, from a terminal enter:

sudo apt install munin-plugins-extra

Be sure to install the package on both the server and node machines.

References

- See the Munin website for more details.
- Specifically the Munin Documentation page includes information on additional plugins, writing plugins, etc.

Nagios

Installation

First, on $server \theta 1$ install the nagios package. In a terminal enter:

sudo apt install nagios3 nagios-nrpe-plugin

You will be asked to enter a password for the *nagiosadmin* user. The user's credentials are stored in /etc/nagios3/htpasswd.users. To change the *nagiosadmin* password, or add additional users to the Nagios CGI scripts, use the htpasswd that is part of the apache2-utils package.

For example, to change the password for the *nagiosadmin* user enter:

sudo htpasswd /etc/nagios3/htpasswd.users nagiosadmin

To add a user:

sudo htpasswd /etc/nagios3/htpasswd.users steve

Next, on server02 install the nagios-nrpe-server package. From a terminal on server02 enter:

sudo apt install nagios-nrpe-server

Note

NRPE allows you to execute local checks on remote hosts. There are other ways of accomplishing this through other Nagios plugins as well as other checks.

Configuration Overview

There are a couple of directories containing Nagios configuration and check files.

- /etc/nagios3: contains configuration files for the operation of the nagios daemon, CGI files, hosts, etc.
- /etc/nagios-plugins: houses configuration files for the service checks.
- /etc/nagios: on the remote host contains the nagios-nrpe-server configuration files.
- /usr/lib/nagios/plugins/: where the check binaries are stored. To see the options of a check use the -h option.

For example: /usr/lib/nagios/plugins/check_dhcp -h

There are a plethora of checks Nagios can be configured to execute for any given host. For this example Nagios will be configured to check disk space, DNS, and a MySQL hostgroup. The DNS check will be on *server02*, and the MySQL hostgroup will include both *server01* and *server02*.

Note

See ??? for details on setting up Apache, ??? for DNS, and ??? for MySQL.

Additionally, there are some terms that once explained will hopefully make understanding Nagios configuration easier:

- *Host*: a server, workstation, network device, etc that is being monitored.
- Host Group: a group of similar hosts. For example, you could group all web servers, file server, etc.
- Service: the service being monitored on the host. Such as HTTP, DNS, NFS, etc.
- *Service Group*: allows you to group multiple services together. This is useful for grouping multiple HTTP for example.
- *Contact*: person to be notified when an event takes place. Nagios can be configured to send emails, SMS messages, etc.

By default Nagios is configured to check HTTP, disk space, SSH, current users, processes, and load on the *localhost*. Nagios will also ping check the *gateway*.

Large Nagios installations can be quite complex to configure. It is usually best to start small, one or two hosts, get things configured the way you like then expand.

Configuration

• First, create a *host* configuration file for *server02*. Unless otherwise specified, run all these commands on *server01*. In a terminal enter:

```
sudo cp /etc/nagios3/conf.d/localhost_nagios2.cfg \
/etc/nagios3/conf.d/server02.cfg
```

Note

In the above and following command examples, replace "server01", "server02" 172.18.100.100, and 172.18.100.101 with the host names and IP addresses of your servers.

Next, edit /etc/nagios3/conf.d/server02.cfg:

```
define host{
        use
                                generic-host ; Name of host template to use
                                server02
        host_name
        alias
                                Server 02
                                172.18.100.101
        address
}
# check DNS service.
define service {
        use
                                         generic-service
        host name
                                         server02
        service description
                                         DNS
        check command
                                         check dns!172.18.100.101
}
```

Restart the nagios daemon to enable the new configuration:

sudo systemctl restart nagio3.service

• Now add a service definition for the MySQL check by adding the following to /etc/nagios3/conf.d/services_nagios2.cfg:

```
# check MySQL servers.
define service {
    hostgroup_name mysql-servers
    service_description MySQL
    check_command check_mysql_cmdlinecred!nagios!secret!$HOSTADDRESS
    use generic-service
    notification_interval 0 ; set > 0 if you want to be renotified
```

```
}
```

A mysql-servers hostgroup now needs to be defined. Edit /etc/nagios3/conf.d/hostgroups_nagios2.cfg adding:

```
# MySQL hostgroup.
define hostgroup {
    hostgroup_name mysql-servers
        alias MySQL servers
        members localhost, server02
    }
```

The Nagios check needs to authenticate to MySQL. To add a nagios user to MySQL enter:

mysql -u root -p -e "create user nagios identified by 'secret';"

Note

The nagios user will need to be added all hosts in the mysql-servers hostgroup.

Restart nagios to start checking the MySQL servers.

sudo systemctl restart nagios3.service

• Lastly configure NRPE to check the disk space on server02.

On *server01* add the service check to /etc/nagios3/conf.d/server02.cfg:

```
# NRPE disk check.
define service {
    use generic-service
```

host	_name	server02
serv	ice_description	nrpe-disk
chec	k_command	<pre>check_nrpe_larg!check_all_disks!172.18.100.101</pre>

Now on *server02* edit /etc/nagios/nrpe.cfg changing:

allowed_hosts=172.18.100.100

And below in the command definition area add:

command[check_all_disks]=/usr/lib/nagios/plugins/check_disk -w 20% -c 10% -e

Finally, restart nagios-nrpe-server:

sudo systemctl restart nagios-nrpe-server.service

Also, on server01 restart nagios:

sudo systemctl restart nagios3.service

You should now be able to see the host and service checks in the Nagios CGI files. To access them point a browser to http://server01/nagios3. You will then be prompted for the *nagiosadmin* username and password.

References

}

This section has just scratched the surface of Nagios' features. The nagios-plugins-extra and nagios-snmp-plugins contain many more service checks.

- For more information see Nagios website.
- Specifically the Nagios Online Documentation site.
- There is also a list of books related to Nagios and network monitoring:
- The Nagios Ubuntu Wiki page also has more details.

pam_motd

When logging into an Ubuntu server you may have noticed the informative Message Of The Day (MOTD). This information is obtained and displayed using a couple of packages:

• *landscape-common:* provides the core libraries of landscape-client, which is needed to manage systems with Landscape (proprietary). Yet the package also includes the landscape-sysinfo utility which is responsible for displaying core system data involving cpu, memory, disk space, etc. For instance:

```
System load:0.0Processes:76Usage of /:30.2% of 3.11GBUsers logged in:1Memory usage:20%IP address for eth0:10.153.107.115Swap usage:0%Graph this data and manage this system at https://landscape.canonical.com/
```

Note

You can run landscape-sysinfo manually at any time.

• *update-notifier-common:* provides information on available package updates, impending filesystem checks (fsck), and required reboots (e.g.: after a kernel upgrade).

pam_motd executes the scripts in /etc/update-motd.d in order based on the number prepended to the script. The output of the scripts is written to /var/run/motd, keeping the numerical order, then concatenated with /etc/motd.tail.

You can add your own dynamic information to the MOTD. For example, to add local weather information:

• First, install the weather-util package:

sudo apt install weather-util

• The weather utility uses METAR data from the National Oceanic and Atmospheric Administration and forecasts from the National Weather Service. In order to find local information you will need the 4-character ICAO location indicator. This can be determined by browsing to the National Weather Service site.

Although the National Weather Service is a United States government agency there are weather stations available world wide. However, local weather information for all locations outside the U.S. may not be available.

• Create /usr/local/bin/local-weather, a simple shell script to use weather with your local ICAO indicator:

```
#!/bin/sh
#
#
#
Prints the local weather information for the MOTD.
#
#
#
#
Replace KINT with your local weather station.
# Local stations can be found here: http://www.weather.gov/tg/siteloc.shtml
echo
weather KINT
echo
• Make the script executable:
```

sudo chmod 755 /usr/local/bin/local-weather

• Next, create a symlink to /etc/update-motd.d/98-local-weather:

sudo ln -s /usr/local/bin/local-weather /etc/update-motd.d/98-local-weather

• Finally, exit the server and re-login to view the new MOTD.

You should now be greeted with some useful information, and some information about the local weather that may not be quite so useful. Hopefully the local-weather example demonstrates the flexibility of pam_motd.

Resources

- See the update-motd man page for more options available to update-motd.
- The Debian Package of the Day weather article has more details about using the weatherutility.

Puppet

Puppet is a cross platform framework enabling system administrators to perform common tasks using code. The code can do a variety of tasks from installing new software, to checking file permissions, or updating user accounts. Puppet is great not only during the initial installation of a system, but also throughout the system's entire life cycle. In most circumstances puppet will be used in a client/server configuration.

This section will cover installing and configuring Puppet in a client/server configuration. This simple example will demonstrate how to install Apache using Puppet.

Preconfiguration

Prior to configuring puppet you may want to add a DNS *CNAME* record for puppet.example.com, where example.com is your domain. By default Puppet clients check DNS for puppet.example.com as the puppet server name, or *Puppet Master*. See Domain Name Server for more details.

If you do not wish to use DNS, you can add entries to the server and client /etc/hosts file. For example, in the Puppet server's /etc/hosts file add:

127.0.0.1 localhost.localdomain localhost puppet
192.168.1.17 puppetclient.example.com puppetclient

On each Puppet client, add an entry for the server:

192.168.1.16 puppetmaster.example.com puppetmaster puppet

Note

Replace the example IP addresses and domain names above with your actual server and client addresses and domain names.

Installation

To install Puppet, in a terminal on the *server* enter:

sudo apt install puppetmaster

On the *client* machine, or machines, enter:

sudo apt install puppet

Configuration

Create a folder path for the apache2 class:

```
sudo mkdir -p /etc/puppet/modules/apache2/manifests
```

Now setup some resources for apache2. Create a file /etc/puppet/modules/apache2/manifests/init.pp containing the following:

```
class apache2 {
  package { 'apache2':
    ensure => installed,
  }
  service { 'apache2':
    ensure => true,
    enable => true,
    require => Package['apache2'],
  }
}
```

Next, create a node file /etc/puppet/code/environments/production/manifests/site.pp with:

```
node 'puppetclient.example.com' {
    include apache2
```

```
}
```

Note

Replace puppetclient.example.com with your actual Puppet client's host name.

The final step for this simple Puppet server is to restart the daemon:

sudo systemctl restart puppetmaster.service

Now everything is configured on the Puppet server, it is time to configure the client.

First, configure the Puppet agent daemon to start. Edit /etc/default/puppet, changing START to yes:

START=yes

Then start the service:

sudo systemctl start puppet.service

View the client cert fingerprint

sudo puppet agent --fingerprint

Back on the Puppet server, view pending certificate signing requests:

sudo puppet cert list

On the Puppet server, verify the fingerprint of the client and sign puppetclient's cert:

sudo puppet cert sign puppetclient.example.com

On the Puppet client, run the puppet agent manually in the foreground. This step isn't strictly speaking necessary, but it is the best way to test and debug the puppet service.

sudo puppet agent --test

Check /var/log/syslog on both hosts for any errors with the configuration. If all goes well the apache2 package and it's dependencies will be installed on the Puppet client.

Note

This example is *very* simple, and does not highlight many of Puppet's features and benefits. For more information see **Resources**.

Resources

- See the Official Puppet Documentation web site.
- See the Puppet forge, online repository of puppet modules.
- Also see Pro Puppet.

Generated from the online docs at https://ubuntu.com/server/docs on Thu Sep 22 09:05:37 UTC 2022