CANONICAL

Ubuntu Core空中软件更新变革之道

2022年1月

执行概要

可靠、稳健、统一地向各平台传送软件更新并非易事。而低功率、不可访问的嵌入式物联网设备往往采用远程管理方式，可靠、稳健、统一地向它们传送软件更新就更为复杂。

但无论规模大小、发布节奏快慢、更新要求的优先级高低，这些问题在Ubuntu Core和snaps中都迎刃而解。采用Ubuntu Core，更新的传送对象无论是单台设备、特定分组设备，还是数以万计的设备，其重点始终是维持系统的完整性。

本文概括了嵌入式设备软件更新相关的难题，还介绍了Canonical如何利用其无与伦比的专长和经验解决这些难题。

挑战

无论是可穿戴技术、家庭自动化、机器人技术、自动驾驶汽车，还是其他的新兴物联网平台，网络连接始终是物联网能否取得成功的关键。

例如，根据国际数据公司（IDC）最新预测资料，到2025年，这些设备的数量将超过410亿台，并将产生近80ZB的数据。随着连接越来越普遍，我们也面临着一系列的重大责任，即保持设备更新、打补丁、保证设备安全，而这一责任主要取决于物联网供应商选择的更新和传送机制。

几十年间，更新及其管理系统径直替换了系统上的现有文件和二进制数据，而它们往往构成一条长长的链条，环环相扣。

任一环节出错，比如软件错误、停电、文件系统问题甚至是宇宙射线，系统便不再统一，需要专业的人工干预才能恢复正常，其代价不菲。

如果更新流程正常，但更新的内容存在错误而需要还原，问题就更为棘手。对传统系统而言，这并非易事，因为从包存储库下载旧版本，并在损坏的系统上安装旧版本，需要软件包维护人员提供相应的脚本。

此前，更新失败或软件回归需要派遣工程师到问题现场解决，或者召回设备，或者对每台设备进行单独的远程更新。以上方案都将耗费大量资源，导致运营支出增加；而且随着部署规模扩大，方案无法大面积推广。

因而，行之有效的更新解决方案需要囊括以下全部要素：

- **设备自动更新**——服务更新需要一个可靠且自动的完成过程，以迎合可预见的更新节奏和传送更新的时间和方式的可选等级精细控制。
- **更新恢复机制**——很多嵌入式设备安装具有不可访问性，所以更新本身必须十分稳定，更新损坏、阻断、中断或者不完整都不会导致其他问题。
- **关键更新准备**——特定情况下需要临时更新或者关键更新，这时更新系统需要优先安排，并将更新内容重新纳入定期更新服务。
- **冗余**——对于不易模拟或预测的情况，更新系统需要具备足够冗余度，可以处理回滚、无网络引导及自动重新预配。

所有嵌入式物联网系统都面临着这些错综复杂的难题。所以，以前更新服务需要物联网供应商慎重对待、长期投入。后续如果其他优先事项受到重视，先前的有效负荷可能会被中断，质量也会下降，最终用户使用受挫，失去信任，以及动摇潜在用户的信心。

更新问题只能采用全面的方式解决，需要从开发开始，一直到生产和部署。解决方案需要重视保证操作系统映像的完整性，同时无论项目处于哪个部署阶段，映像都应该是安全可靠且可复制的。

解决方案

Ubuntu Core是面向物联网创建的操作系统，具备安全、轻量、可靠，以应用程序为中心的特点。用于生成Ubuntu Core只读根文件系统的包，与用于生成多种Ubuntu发行版的包相同，区别在于包的传送方式和更新方式。传送和更新均由snap处理，snap是一套安全、封闭、独立的跨平台Linux打包系统。

snap包确保基本系统与需要安装的应用程序彻底分离，每个应用程序、应用程序的数据，以及甚至应用程序版本数据也相互隔离。

更新为事务性的，只有两种结果，如果不是100%成功更新，那便是未安装。如果未安装，除记录详细信息外，安装失败不会留下任何其它的痕迹。这意味着应用程序或系统更新期间，系统完全可以正常运行，并处于持续定义良好的状态。必要时，系统即使未能启动，也可恢复或者还原到先前状态。它不同于其他包管理器或者更传统的包管理器，更新失败不会导致系统状态处于无法预测的状态。

这是一套从头开始设计的解决方案，用于解决更新关键软件相关的复杂问题。在实现自动、可靠、安全、透明更新的同时，其还能够对更新部署实现全面而精细的控制和定制。

Ubuntu Core与snap采用作为物联网部署基石的如下三种独特方式，实现了更新方法的革新：

- 系统维护
- 更新机制
- 安全性

安全性

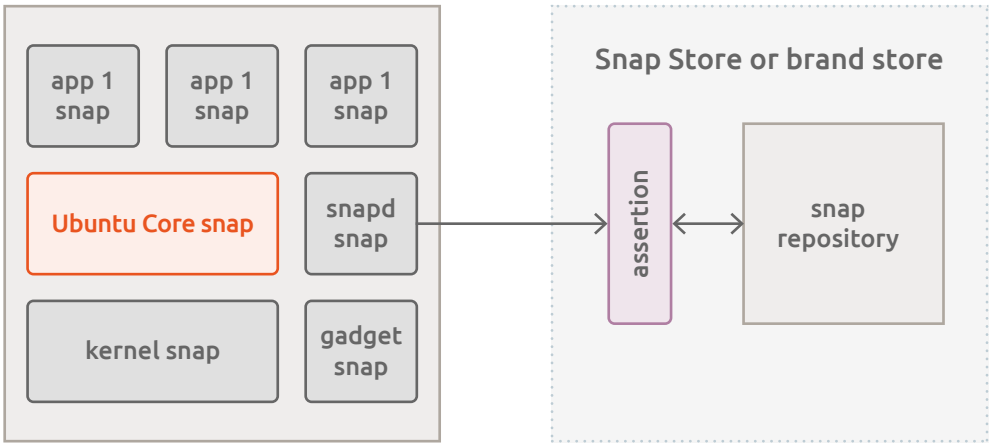
安全性是一切设备都需要考虑的重要因素，而对于主机应用程序在受信任及不可访问环境下自主运行的物联网而言，安全性也必须是一个基本的考虑因素。这些设备不断受到运行在系统上更广泛网络的威胁，而对系统上的软件栈而言，我们始终需要为其减少漏洞。

然而，安全性和漏洞问题经常被忽视，构建平台时很少考虑这两个因素。事实上，往往只在面向客户的核心功能设计、开发、迭代、甚至部署以后，这两个因素才被“塞”进软件栈。如果等到发现漏洞才意识到安全性的重要性，我们便需要为此付出高昂的代价。更糟的是，很多物联网设备以各设备独特的自定义构建软件栈为基础构建，它们不直面问题进行有效更新，而是更容易放弃部署，设备更加脆弱。

Ubuntu Core和snap则另辟蹊径。Ubuntu Core和snap构建于不可变的基础之上，（不可变基础）利用了Linux内核和Ubuntu的共同优势，深受数以百万计嵌入式、桌面和云用户的青睐。Ubuntu Core和snap用于构建设备的平台将安全性放在首位，成绩记录得到公认，实现了物联网部署的革新。两者背后是一种业已得到证明的更新传送机制，及一套支持整个平台的、广泛使用的开源工具。

基本系统只包括必要的内核、init程序、若干重要工具，以及管理snap的守护程序——snapd。snap守护程序将cgroups（控制组）、Seccomp（安全计算模式）和AppArmor等开源技术完美地结合起来，确保snap始终与系统及其他snap隔离，同时仍允许对哪些资源可以访问和共享进行细化控制。

Ubuntu Core、snap应用、Snap应用商店



如果没有明确批准断言（assertion）中分配的权限，snaps不能使用crond、切换其他用户、访问硬件、添加用户、修改setuid/setgid值、改变安全策略、修改系统、修改内核运行时变量、访问敏感的内核syscalls及破坏系统所需的许多其他功能。snap本身如果没有预安装，则总是由经过数字签名断言证明的可靠来源进行传送和更新。

例如，Snapcraft构建工具大大减少了供应商打包自己应用程序的开销，大多数应用程序只需创建一个说明应用程序要求的配置文件。Snapcraft构建框架中整合了这些要求，确保所产生snap是安全、合规且高效的。

这一流程可以大幅降低总拥有成本，缩短产品发布时间，因为构建过程和操作系统的基础已经被广泛使用多年，两者是开放的、高度可用的。

> 举例来说，历史悠久、享有盛誉的日本科技公司Cyberdyne就利用了这些独特功能。Cyberdyne在清扫机器人中使用Ubuntu Core和snap，可以帮助缓解日本劳动力数量减少导致的用工荒问题。
>
> 正如Cyberdyne公司软件工程师Marc Benetó所说，“有了snap，我们就放心了，我们的机器人已经万事俱备了。不论哪个软件版本，我们都能将机器人所需的一切打包，不存在缺少依赖项的风险。”
>
> 本案例研究表明，Ubuntu Core和snap帮助Cyberdyne缩短了新功能和错误修复的推出时间，而自动更新节省了运营开支，大幅节省了带宽。

系统维护

无维护，不安全。但维护并不只是指使一切持续更新且尽可能干净的过程。要进行维护，我们必须首先拥有一个良好的设计。

没有良好的设计，任何部署都很容易陷入未知状态，文件系统的文件散乱，临时文件未经检查，遗弃的配置文件和应用程序不受监控。供应商和用户将趋向于选择最省事的途径，比如要么采用临时文件存储，要么残留更新后的配置或临时文件。

结果，维护和部署更为复杂，加重了维护负担，通常最后维护会变得难以为继。系统不安全了，并且会长期保持不安全的状态，最终只能重置为已知的良好状态。

良好的设计是Ubuntu Core的核心，Ubuntu Core被设计为维护起来轻而易举。

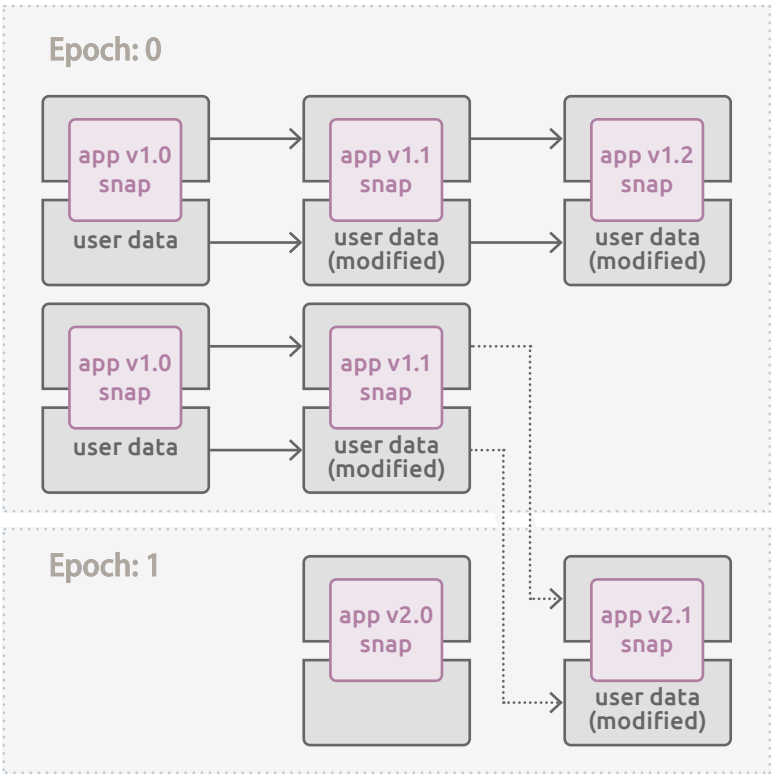
- Ubuntu Core的文件系统是只读的
- 配置和应用程序数据被限制在每个snap中
- 数据始终处于已知状态
- 更新是已签名的，而且是原子式的
- 应用程序受到限制，未经许可不能相互修改

如果没有snap，以上最后一点是办不到的，因为snap是直接载入主机文件系统的单个文件。与核心文件系统一样，其对系统而言是只读的，并且凭借称为“接口”的一组显式权限获取授权，可以访问需要的一切资源。接口的执行采用了久经考验的Linux内核限制功能。

snap本身完全独立，甚至封装了自己的文件系统，即snap包含了应用程序运行所需而Ubuntu Core未提供的一切。Ubuntu Core本身是由snap构建的；存在一个包括Linux内核和特定硬件驱动程序的内核snap、一个描述具体设备型号细节的小工具snap、一个针对snapd守护程序本身的snap。这协调了snap的安装和更新程序，以及如何将定义某个snap权限的断言转化为本地系统特权。

系统还允许同一snap的多个版本共存，且不会产生任何不良影响。这种机制被称为并行安装，同一snap的多个独立实例可以从自己的命名空间内部安装运行。供应商能够同时为新旧版本的应用程序、库或开发平台提供支持，一直到旧版本被弃用。与其他snap一样，每个实例与所有其他实例完全隔离，包括snap的名称、配置、接口连接、数据位置、服务、应用程序和别名。

并行安装和Epoch



此外，snap还具备一个称为“时期”的特性；有了时期，如果一个应用程序的配置或数据格式发生了很大变化，例如发布了一个主要版本，那么旧的、不兼容版本不会自动更新。相反，供应商可以完全控制哪些用户可以获得更新以及何时获得更新，确保经过测试的迁移策略可用时，老用户才进行更新。更重要的是，设备只要已经采用了原旧版本初始化，即会执行这一流程，并将按照上述迁移策略自动更新。

所有这些都被封装进构建和发布过程，可以简便地纳入CI系统；更改提交之后，系统将自动创建和发布snap。轨道和通道等snap的特性可以用来设置版本的稳定性预期，例如，前沿通道/前沿版用于紧跟最前沿特性，而稳定通道/稳定版可能只留给值得长期支持的版本。

更新机制

更新自主进行，既简单又可靠。

新的snap发布到Canonical的全球snap商店或物联网供应商控制的私人/公共品牌商店后，可通过可配置的刷新节奏自动推送到设备上。更新以增量（deltas）形式传输，这是一种差异数据压缩形式，只传输当前已安装的snap与新的snap存在区别的部分。增量更新大大节省了带宽，当数以千计物联网设备数量成倍增加时，将对设备部署产生巨大影响，因为设备可能需要在短时间内完成多项关键更新。

Rigado Cascade: "Snap增量"节省了65%以上的带宽。

Rigardo是美国的一家数据网络供应商，专注于边缘物联网设备。Rigado的Cascade物联网网关提供了可靠的容器化应用程序环境，以及包括蓝牙5、LTE和Wi-Fi在内丰富的无线设备连接方案。这一切的实现都仰赖于强大的Ubuntu Core和snaps。

假设Rigado是snaps基本集典型更新节奏以及设备特有snap特点如下，Cascade设备每月带宽消耗见下表。Rigado的平台可以启用的设备类别众多，所以一家组织可拥有大量由Cascade驱动的设备。拥有1万台设备的组织预计节省的总带宽见下表。

Snap 名称	每月更新（平均）	完整 snap 大小（平均，MB）	增量平均大小（平均，MB）	每月1万台设备带宽（无增量，GB）	每月1万台设备带宽（有增量，GB）
bluez	0.21	3.36	1.4	6.87	2.85
cascade	1.6	0.52	0.22	8.14	3.36
cascade-	0.89	0.08	0.03	0.68	0.63
configuration					
cascade-kernel	1.74	35.55	17.34	605.49	295.28
core	1.83	69.33	20.55	1,240.68	367.74
modem-	0.42	5.77	1.23	23.61	5.05
manager					
network-	0.26	5.9	1.89	15.07	4.83
manager					
pivot-agent	0.43	2.45	2.45	10.32	10.32
rigado-	4.33	3.25	2.1	137.58	88.78
deviceops					
rigado-edge-	1.48	3.58	2.13	51.88	30.8
connect					
rigado-fw-	0.11	0.004	0	0	0
loader					
rigado-network-	1.14	8.45	4.23	94.09	47.13
config					
wifi-ap	0.26	25.86	9.48	66.12	24.25
fictional-app*	1.12	175.48	51.56	1,921.05	564.46
			共计	**4,181.58**	**1,445.47**
			减少量（%）	**65.43%**	

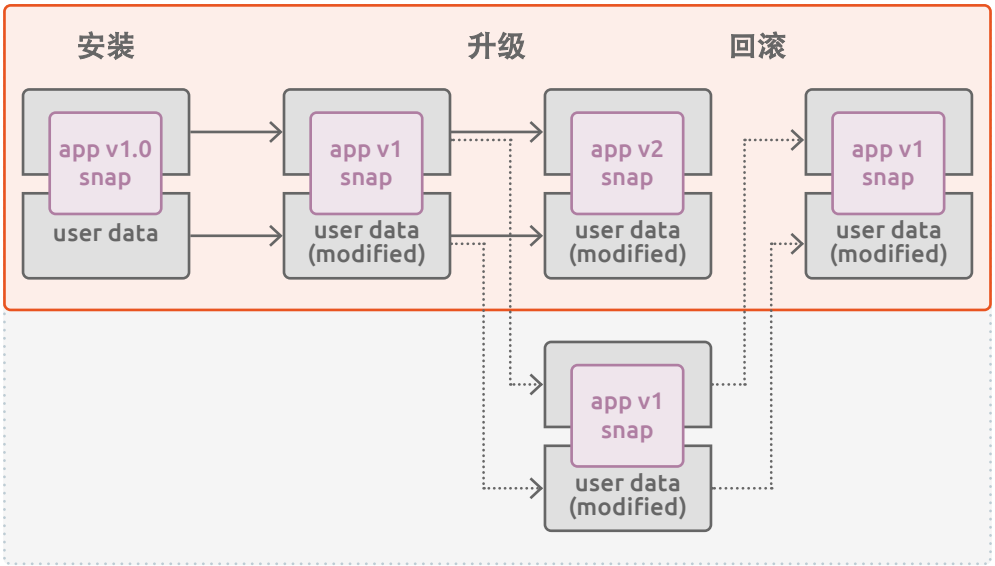
- 假设的fictional-app是基于流行的个人云文件存储解决方案Nextcloud的规模和增量性能，但其更新节奏与Cascade设备上其他snap的平均值相匹配。
- Rigado-fw-loader设备上提供的只有一个版本；它不会下载更新，带宽需求为零。
- Pivot-agent是snap的一个实例，相比完整下载，增量形式并未节省带宽（实际上更大，所以我们会放弃增量形式）。在这种情况下，Snap商店传送完整下载，也因此带宽一列，无论是否采用增量，该snap的数值相同

数据摘自白皮书《利用增量更新优化物联网带宽》

更新期间，snap的数据始终是安全的。每个snap都有自己的snap专用和分段数据存储区域，该区域通用的可写操作系统文件和snap守护进程本身的状态分开。snap被更新时，无论它在堆栈中处于什么级别，它的数据都会被复制，新snap可以访问数据。如果更新事务失败，不管什么原因，snap（包括其数据）都可以回滚到更新之前的状态。

在处理内核和可启动的基础snap要格外小心。安装新内核或可启动的基础snap时，Ubuntu Core会启用特殊的"尝试启动"模式。如果随后重新启动失败，系统会自动恢复到以前的工作内核或基础snap。同样地，每个版本snap各有数据目录，这意味着当某些（数据）被破坏且需要回滚时，前一个版本的数据也会被恢复。新版本未检测到的和不兼容的变化都不会破坏回滚。

Ubuntu Core中的事务性更新



默认情况下，Ubuntu Core会将snap处于停用状态的前两个版本在本地存档。如果新版snap出现问题，只需使用"snap revert problematic-snap"，系统就能恢复之前的工作版本。而且，由于每个存档的snap都包括各自的数据，回滚也将恢复更新前的数据。这意味着，即使新版snap存在未检测到的数据吞噬错误，利用存档的先前版本也可以恢复数据（除非snap发布者选择不使用该功能）。

不断演进的更新

低功率、不可访问的嵌入式系统往往采用远程管理方式，可靠、稳健、统一地向它们传送软件更新较为困难。

有了Ubuntu Core和snap，更新问题便会迎刃而解。对于不断发展的物联网部署，采用这套解决方案，无论部署规模多么庞大，也无论更新多么频繁，复杂性和不确定性都不会增加，用户可以完全放心。

其优势不仅体现在传送更新的方式上，也体现在计划、安排和发布方式的可预见性上。此外，这些优势可以帮助降低设备的总拥有成本，而且相比设计构建工具、更新机制和传送基础设施，供应商从产品设计到产品部署，所需时间大幅缩短。

如欲深入了解Ubuntu Core的使用，欢迎联系我们。

延伸阅读:

- Ubuntu Core
- 嵌入式Linux
- 利用增量更新优化物联网带宽
- Ubuntu Core——安全性
- Ubuntu Core为物联网安全之选的原因
- AppArmor介绍

ubuntu
Delivered by Canonical