CANONICAL

The transformation of the DevOps journey in IoT

Using Ubuntu Core and snaps to accelerate the Linux development cycle

April 2019

Executive summary

Traditional development methods do not scale into the IoT sphere. Strong inter-dependencies and blurred boundaries among components in the edge device stack result in fragmentation, slow updates, security issues, increased cost, and reduced reliability of platforms. This paper showcases the use of Ubuntu Core and snaps as an alternative to the existing model, with a strong focus on isolation, reliability and an accelerated development cycle.

Introduction

Over a period of just a few years, the world of computing has shifted from a rigidly defined model where dedicated machines utilise content provided by centralised servers to one where the majority of electronic devices are considered to be computers, regardless of form factor or purpose. The ubiquitous nature of this change is popularly known as the internet of things (IoT). It defines a global network of interconnected systems, from classic computing nodes to home appliances, vehicles, sensors, and a range of other devices that, until quite recently, could have been considered dumb, e.g. lacking internet connectivity.

In 2017, there were already about 8 billion IoT devices in the market, and Gartner predicts the figure to rise to 21 billion interconnected devices by 2020 [1], mostly in the consumer market. The IoT revolution has transformed the digital landscape in that it has blurred the previously well-defined boundaries between lifecycle management and security. Moreover, it has introduced significant scalability challenges into the development sphere.

Currently, the end target of software development in the embedded Linux, and to a large extent, the IoT domain consists of monolithic images that tightly couple the hardware platform, firmware, operating system kernel, core libraries, and the user-space components. The strong inter-dependencies among the different parts in the stack allow IoT vendors better control of their devices, but they also make development and validation more complex.

- **Product boundaries and complexity** - Whenever there is a change in one of the components, the entire sequence of testings needs to be executed to ensure there are no adverse changes or regressions in the platform. A further complication stems from the fact there are multiple elements in the stack with their own independent development cycles, including board support package (BSP) vendors, device image vendors and third-party software application vendors. For instance, a change in the board or chipset code will trigger the validation for the entire platform. Likewise, security patches for the application will also necessitate a lengthy testing procedure. In turn, this leads to fragmentation and delays in product release management.

- **Fidelity of the end state** - Platform reliability cannot be guaranteed following changes in any one of the components of the stack. The current inability to control the entire stack poses significant strain on product vendors, resulting in a disruption to the wider industry.

- **Updates and security** - In turn, the need for additional validation and testing can introduce extra burden and cost in the product development. This issue manifests in two vectors - fragmentation and slow release cycles, leading to gaps in the overall product security.

- **Slow release cycles** - As a result of update constraints, the IoT landscape includes a significant percentage of devices that are not up to date. This creates a security problem for end users especially since customers expect new features on old hardware - an issue that largely did not exist in the traditional world.

- **Fragmentation** - Not all IoT devices of the same type can be updated at the same time, and sometimes the updates need to be deferred indefinitely leading to a management process complexity for vendors, and in turn, to some extent, for developers building things for IoT.

This reality places a major strain on IoT players who need to contend with varying cycles and priorities in the development stack, limiting their flexibility to innovate and introduce changes into their products, both on the hardware and software sides. The expansion into new verticals is limited by the ability to execute in a timely fashion and this complexity is only expected to increase with the growth of the internet of things.

As a contemporary stopgap measure, IoT vendors and software developers have adopted several possible methods to improve the control and pace of their product release cycles. One notable way to reduce the complexity in multi-component stacks is through the use DevOps - a refined development paradigm that combines development with operations in a single and streamlined process. While the term DevOps is used liberally across the industry, it does have significant potential in the the world of IoT where the development pace exceeds the efficiency envelope of the conventional release practices. However, DevOps alone cannot solve the complexity and dependency that exists in monolithic IoT products.

At Canonical, we believe the integral solution to this fundamental problem is the decoupling of components in a reliable and predictable fashion which will reduce the inter-dependency, improve security and allow for faster development cycles. We propose snaps - self-contained, isolated applications that bundle all the necessary libraries and dependencies - separate from the rest of the system.

Snaps architecture & overview

Snaps have become a popular package format for a wide range of Linux applications. The fundamental concepts behind snaps are isolation and reliability.

- Each snap is a standalone, self-contained application that bundles most of their needed libraries and runtimes and can be updated and reverted without affecting the rest of the system. Snaps interface with the underlying system through a snap base package. This also ensures portability and reproducibility of the application's behaviour, as snaps do not need to be modified to work on different platforms.

- Each snap consists of a squashFS file system containing application code and declarative metadata that defines how the application should be handled. It has a read-only file system and, once installed, a private writable area.

- By default, snaps are confined and cannot access system resources. A fine grained mechanism of interfaces allows snaps to communicate with the rest of the system when needed.

- Snap updates are automatic and transactional which offer a higher level of compliance and security than the traditional update methods, whereby patches can easily be deferred or skipped, leading to gaps in update coverage. Likewise, missed or broken updates often result in an inconsistent system state. With snaps, if an update fails for whatever reason, changes will be rolled back to the previous version ensuring the application continues running without a loss of service.

A system capable of running snaps consists of a snapd background service, snap user-space component and the Snap Store which is a central repository of snaps offering strong security and cryptographic signatures. The store contents are available to any system running snapd. Moreover, the store features release channels which allow multiple versions of the application to be available at the same time.

Snaps provide several distinct advantages over traditional packaging models by providing a higher level of reliability and security. On their own, snaps do not address all the concerns and challenges prevalent in the IoT world. However, they are a baseline for an extended, refined model that also includes a version of Ubuntu designed specifically for IoT devices and large containers deployments - Ubuntu Core.

Ubuntu Core overview

This tiny, transactional version of Ubuntu uses the same kernel, libraries and system software as the classic edition. However, the fundamental difference is that it relies on snaps as its only delivery mechanism for software and updates, including the kernel.

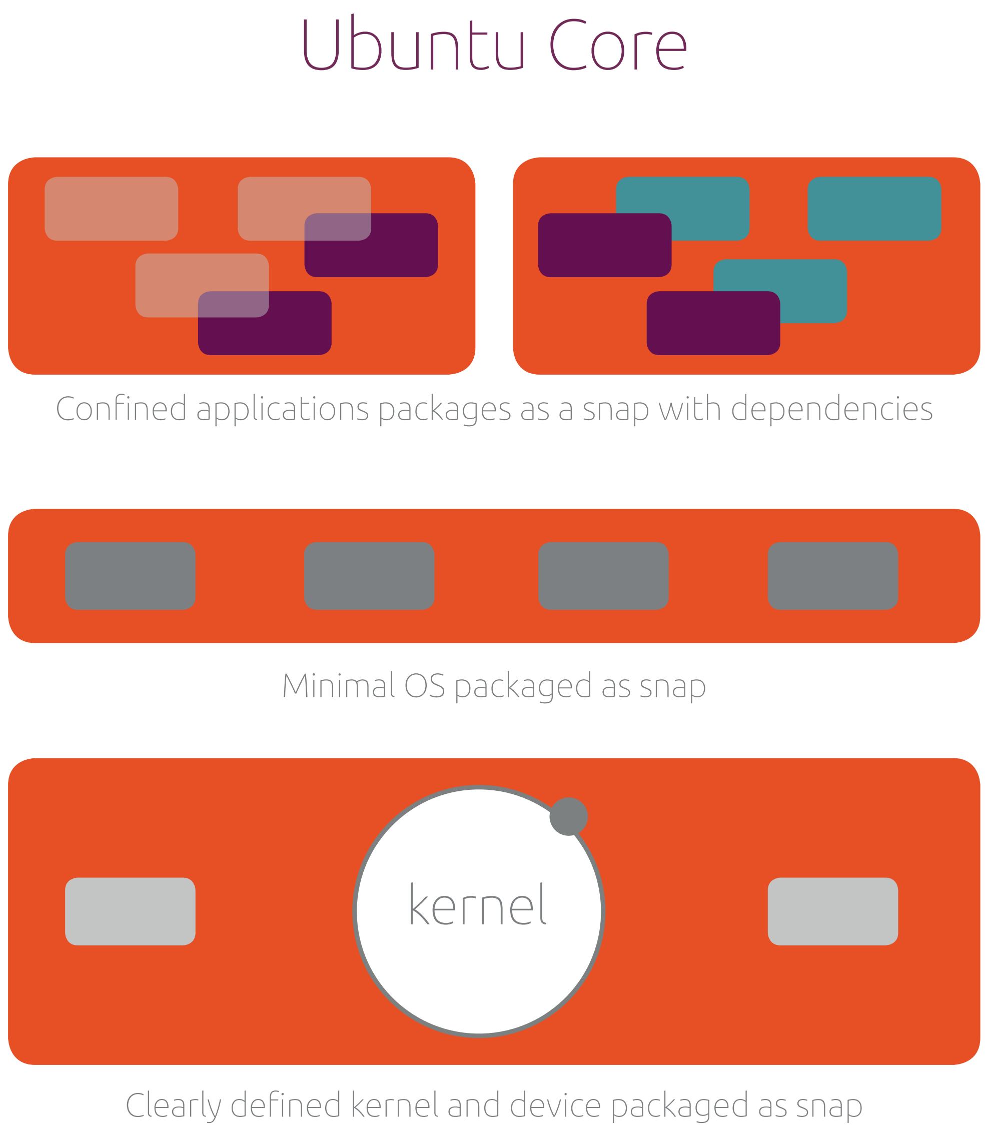
Ubuntu Core
Confined applications packages as a snap with dependencies
Minimal OS packaged as snap
kernel
Clearly defined kernel and device packaged as snap

Figure 1: Snaps ensure separation and confinement of applications from the operating system, minimising dependencies on shared libraries and ensuring that applications run and behave in a consistent manner with subsequent updates.

Building on the confinement and reliability of snaps, Ubuntu Core pivots around security and updates with the purpose of resolving the main challenges that currently exist in the monolithic development model - product boundaries, reliability of the end state and security. The last element is of particular importance as 57% of users expect security from the service provider or the device manufacturer. This remains the primary driver, or rather the blocker, in the proliferation of IoT devices.

The layered approach to security in Ubuntu Core is implemented through several hardening mechanisms, on top of the robust tools already used in classic Ubuntu. Table 1, below, outlines some of these additional security features.

Mechanism	Ubuntu Core
Administrative user (root) password	Disabled
System users login	Disabled
Multi-user support	Limited
Snaps confinement	Strict[1]
Systemd specification	Limited
Trusted Module Platform[2]	Userspace tpm interface[3]

Furthermore, the reliability of the end state is achieved through transactional, atomic updates. In addition to strict isolation of applications from one another, snaps are also upgraded in a way that ensure a complete rollback on failure. As a first step, the original snap data including the writable area are copied. The new updates are then applied in an atomic manner. Only if the update is successful is the new snap version used. Otherwise, the original snap is rolled back and the update deferred. The snap will be updated when a new, revised version of the update becomes available, but importantly, the application will continue working and providing necessary functionality to the end user.

In classic Linux, updates present a risk in that failures can result in an unknown system end state whereby only a subset of necessary libraries might be updated while others could be broken, corrupted or left over at previous versions. Moreover, bugs and regressions in library dependencies provided by one application can also have an adverse effect on all applications that dynamically load these libraries during runtime. This lack of predictability leads to extensive testing and validation which can slow down the development process and increase overall product cost. This is further exacerbated in the IoT domain with the use of monolithic images that contain components from multiple vendors.

1 Snaps are not allowed to use crond scheduling service, change to another user, have unapproved access to hardware, add rsyslog rules, add users, ship setuid/setgid programs, change security policy, modify the system, modify kernel runtime variables, or access sensitive kernel syscalls.

2 Requires hardware support.

3 Future versions of Ubuntu Core may incorporate the use of TPM as part of device identity, measurement and data encryption on supported hardware platforms.

The update mechanism in Ubuntu Core ensures that applications continue working as expected, even in the case of failures. Problems in one snap will not propagate through the system and affect other applications. In turn, this makes validation simpler and faster. The update flow is outlined in Figure 2 below.

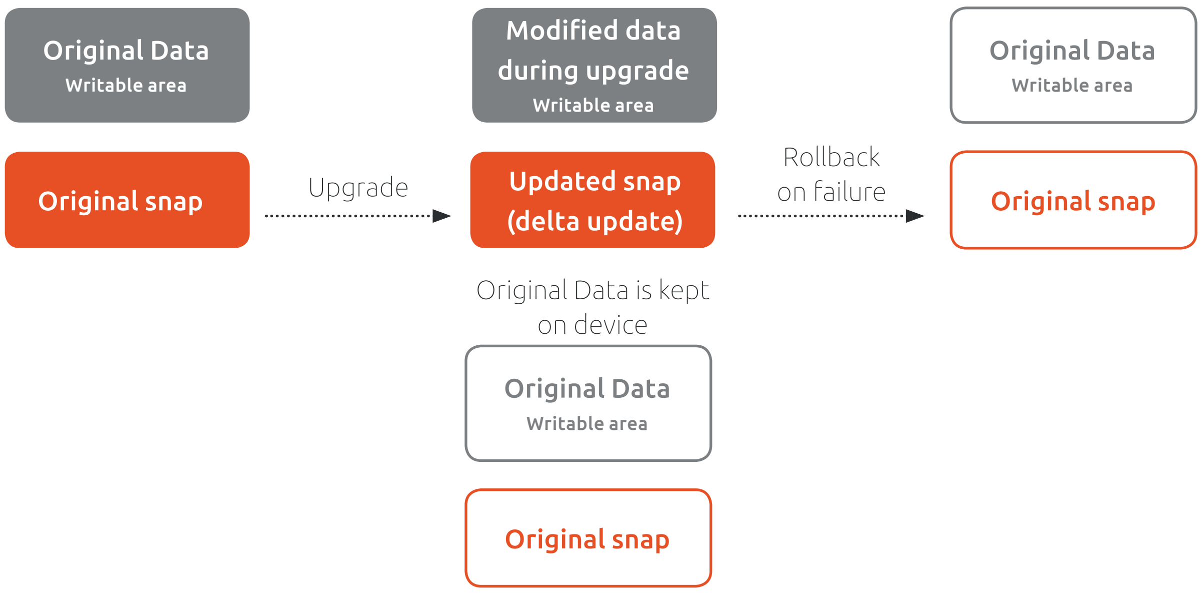

Figure 2: An overview of the snaps deployment mechanism highlighting how snaps are upgraded with the original data kept in place in case of an update failure.

Suitability of Ubuntu Core in IoT

A high level of granularity, control and containment that is necessary for predictable deployment of software are the primary advantages of Ubuntu Core for IoT use. Reliable updates and confinement ensure security and allow vendors to follow a faster release cycle, including both development and target device upgrades. The isolation also means that product components from BSP vendors can be packaged as individual snaps, and no longer necessitate a validation chain for the entire platform.

Another important feature is strong integration and automation. Snaps have been designed with the exponential growth of the IoT world in mind. Quite often, software development is linear, time intensive and includes a large amount of human interaction. On a larger scale, this model breaks down.

With snaps, it is possible to accelerate the software build and testing cycle through the use of continuous integration (CI) systems like Travis CI [2]. Moreover, there are additional integration features, like hosted service and on-premise edge proxy, which can enhance the granularity of control in the development and application build cycle.

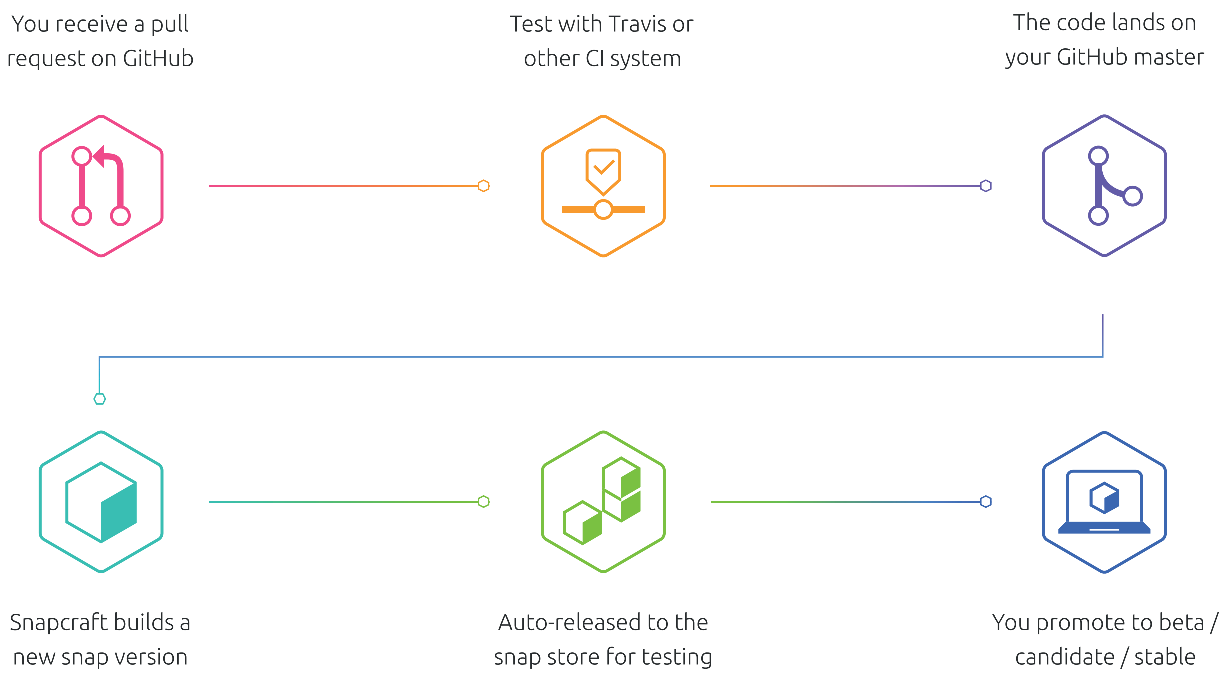
You receive a pull request on GitHub
Test with Travis or other CI system
The code lands on your GitHub master
Snapcraft builds a new snap version
Auto-released to the snap store for testing
You promote to beta / candidate / stable

Figure 3: An example of a possible build/test/deploy cycle using a CI system and snaps. The Snapcraft framework includes channels, allowing for software to be released in a controlled manner, with fine-tuned control of every stage of development.

Case study 1: Rigado

Rigado [3] is an IoT edge connectivity vendor with a broad portfolio of products. They offer edge-as-a-service solutions for commercial IoT and their journey from the classic development model to snaps is an interesting example of the fundamental paradigm shift discussed in this paper.

Rigado's drive to use snaps stemmed from operational challenges in their solution stack. Namely, a typical solution comprises three owners:

1. The operating system
2. Value add software, APIs, management and alerting agents by Rigado
3. Top-level applications provided by customers

Often, these components would be developed and tested independently. The different time tables, priorities and abilities to adapt to changes, implement new functionality, and provide testing introduced significant challenges and complexities into the solutions stack. This made it more difficult for Rigado to provide and scale their products. A large part of this was due to the monolithic nature of the operating system.

In the pre-snap environment, other versions of their Linux operating system required complete image updates, regardless of the nature of the change (core operating system, agents, APIs, or end-user applications). In turn, this necessitated a frequent image update process and it would often fall upon end customers to implement the change, whereas they may not be the most suitable owner for the task.

Furthermore, in many cases, customer applications often did not have as many planned updates as the system and possible API updates suggested to keep their devices up to date with the latest features. This problem ties into the asynchronicity of hardware and software update cycles discussed earlier.

Looking across the technology landscape for possible solutions and workarounds, Rigado were convinced that the isolation (containerisation) model, one of the available options often used for separation and faster software development, was incomplete. Snaps offered an answer to this problem.

Snaps gave Rigado the ability to introduce a proper division of labour and keep each tier of development within their solutions in the hands of the particular owner. As a result, each owner becomes responsible for the updates of their own components with their own software lifecycle and updating schedule. This enabled their ecosystem to scale and still keep the solutions secure. Using snaps, Rigado were able to achieve and improve in the following areas:

- **Shorter time-to-market (TTM)** - The lifecycle of Rigado components in their solutions is a SaaS-like continuous deployment model, with a higher-than-anticipated frequency of releases.

- **Alignment** - The deployment speed enabled by snaps allows Rigado to more quickly introduce and implement changes in edge software, and align it with that of the cloud applications. Previously, there was a misalignment between the cloud and edge deployment cycles that complicated the IoT solutions update cycle. The cloud SaaS model allowed for near continuous deployment of fixes and new features, but the edge environment was only being updated once every 5-6 months on average. Snaps fixed this discrepancy.

- **Clear ownership** - With snaps, Rigado can now draw a clear line between the ownership parts which also helps in troubleshooting because, quite often, the root cause of different issues are easier to find. Rigado are able to update and fix parts of the solution independently, without affecting the other parts. The divide & conquer method speeds the time to resolution of most typical problems.

- **Improved security** - With the classic model, Rigado solutions were on a quarterly cadence to keep the operating system updated for its customers, and many of them were not able to keep up with the pace of change. As such, a large number of customers were six months to a year behind on important system fixes and patches. This left their devices open to attack and vulnerable. With the separation of lifecycles and reliability afforded by snaps, Canonical can keep them continuously up to date and more secure.

Previously, the customers followed a linear model; they wrote their software, built a complete image, tested the whole solution for each and every feature, and once all these would be finished, they would upload the new solution. This was typically a long process. Snaps require a different programming, QA and deployment process. At any point in time, there can be several software combinations from each of the three solution owners running on the system simultaneously. This transition is a significant one. It was also an important lesson that Rigado is trying to incorporate into a best practice for helping their customers move from a monolithic environment to snaps.

Click here to hear Rigado's CEO discuss the benefits of Ubuntu Core and snaps

Case study 2: Fingbox

Fing [4] is the software vendor behind Fing App, a network scanning and troubleshooting tool. They also manufacture Fingbox - a complementary hardware device that plugs into routers and allows users to monitor, secure and troubleshoot their networks.

Early on, speed and reliability of updates were prioritised as crucial factors during the development of Fingbox so that the company's engineers could focus on the core product with minimal investment in the maintenance of the operating system. Crucially, the company required a highly robust update mechanism, driven by strong security considerations, whereby they could push new versions of software within hours of release [5][6].

The Fing engineering team decided to use Ubuntu Core and snaps as the foundation for their product which allowed them to speed up development time, save budget and resources and ensure a future-proofed product went to market. The results of this transition are best illustrated in Figure 4 below.

The graph shows the Fingbox version update cycle over a sample period of 30 days, with a coverage of about 30,000 devices. During this period, Fing successfully released and pushed three major versions of the application to the target devices.

Each update cycle covers a period of roughly a week between releases. Each time, more than 95% of devices were upgraded in a consistent manner in less than a day with the remainder switched over to the new version over the following days. This illustrates the reliability of the snap mechanism as Fing were able to sustain a consistent, continuous, fast update cycle and perform version upgrades in a short time. This is also indicated by the steep change from one release to another - an implicit indicator of the quality of the released software versions as rollbacks to previous snaps would be indicated by a larger percentage of older releases after the weekly cycle.

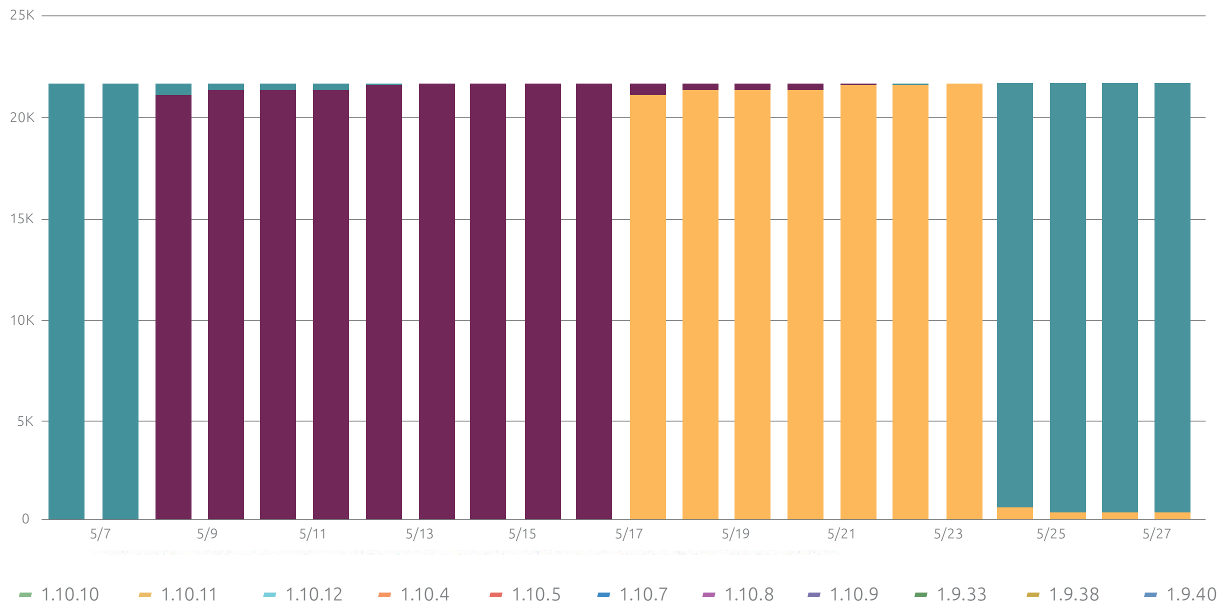

Figure 4: Fingbox updates, tracked over a period of 30 days for 22,000 devices.

The results are promising in that when the frequency and the speed of change seen here are compared to similar scenarios across the industry, like phone manufacturers, there is often a more gradual, staged pace of deployment, constrained both by the underlying architectural challenges as well as the software stack dependencies.

This case study highlights the benefits of using snaps on a large scale. It can allow hardware manufacturers and software vendors to push the boundaries of their development, with more focus on the reliability of their product due to the diminished need to control the operating system stack. Moreover, Ubuntu Core provides the necessary foundation for such work, thanks to its robust, secure architecture and the transactional nature of its updates mechanism.

Into the future

Today, switching a hardware platform is a complex undertaking that requires all components in the monolithic image to be fully aligned and tested. With Ubuntu Core, only the operating system needs to be tailored to the new hardware. The functionality of snaps remains unaffected, making changes largely transparent, and offering significant leeway in technological and commercial choices.

These features position Ubuntu Core as an ideal driver for a large number of future business models in the IoT space. As this domain continues to grow, so will the number of different hardware solutions and their permutations. Working with monolithic, rigid images for each one creates an unscalable scenario in which companies will not be able to provide necessary security updates and feature upgrades for their edge devices. There is already a significant percentage of devices past their warranty or support date still actively used online. Without security updates, these systems pose a risk both to the users and the stability of the internet infrastructure as a whole as they are often exploited and used in large IoT botnets [7].

A de-coupled model, with strong isolation and reliable updates, as exemplified by Ubuntu Core and snaps does offer a solution to these pressing issues. On one hand, it allows developers to focus only on their software without having to think whether an unknown third-party library or a small system change is going to break their functionality. On the other, it also gives hardware vendors a reliable, flexible hardened operating system to work with which can lead to faster innovation and improved security.

We believe the IoT world to be one of rapid change. It differs from the classic Linux and the pace of change exceeds the ability of current methodologies to cope. Looking forward, vendors will have to adopt a more modular approach that provides better security, better isolation and an easier way to rapidly iterate on software development. Ubuntu Core and snaps, both as individual concepts as well a unified solution are well positioned to provide the necessary framework for this fundamental shift.

Conclusion

Reigning in the chaos of the digital world is an elusive game of cat and mouse between vendors and users and it has never been more apparent than today. The IoT revolution is blurring the boundaries of accepted conventions. The monolithic model is breaking down and there is a strong need for strict delineation between hardware and software development.

Snaps provide a flexible solution that minimises the application dependencies on the lower tiers in the software stack. They offer a higher degree of certainty that applications will behave in a consistent and predictable manner, regardless of any other updates or changes on the system. This allows developers to focus on enhancing the core product, and in the long run, provides confidence and security to the end user. These features are essential for the growth and success of any potential IoT model.

The Rigado and Fingbox stories also highlight the advantages of Ubuntu Core and snaps over the classic scenario, with fast, reliable development environments, continuous integration and sustainable release cycles. Both companies are enabled to push updates in a consistent manner, and maintain both functionality and security. They both serve as robust examples for the future landscape of the IoT world.

Further reading:

Webinar: An introduction Ubuntu Core 18
www.ubuntu.com/core
www.snapcraft.io

References

[1] https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

[2] https://travis-ci.org/

[3] https://www.rigado.com/

[4] https://www.fing.io/

[5] https://www.brighttalk.com/webcast/6793/336519

[6] https://blog.ubuntu.com/2018/06/28/fing-snaps-up-30000-customers-with-a-secure-future-proof-iot-device

[7] https://www.csoonline.com/article/3258748/security/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html

© Canonical Limited 2019. Ubuntu, Kubuntu, Canonical and their associated logos are the registered trademarks of Canonical Ltd. All other trademarks are the properties of their respective owners. Any information referred to in this document may change without notice and Canonical will not be held responsible for any such changes.

Canonical Limited, Registered in England and Wales, Company number 110334C Registered Office: 12-14 Finch Road, Douglas, Isle of Man, IM99 1TT VAT Registration: GB 003 2322 47

ubuntu
Delivered by Canonical