CANONICAL

A guide to Linux for embedded applications

February 2022

Contents

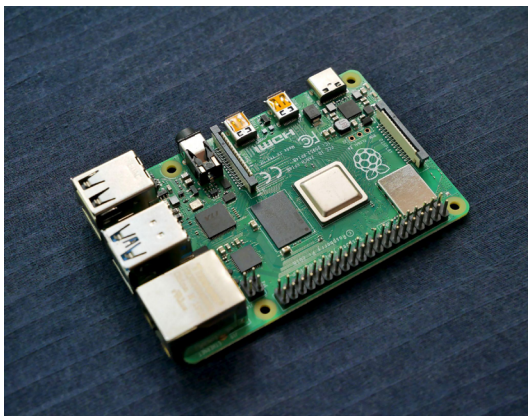
> *"Hello everybody out there using minix -*
>
> *I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. [...] It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(."* [1]

The above is an excerpt from Linus Torvalds' original announcement of what is now known worldwide as the Linux kernel, dated August 1991. In hindsight, it is unbelievably modest now that we are in a much better position to appreciate the full revolutionary extent of his post. Today, more than 96.4% of the top 1 million server applications run on Linux [2]. Across public clouds, 90% of the workloads use Linux as their OS, with Ubuntu being the most popular Linux distro [3], [4]. Moving past its initial debut as a general-purpose **operating system** (OS) for i386-based hardware, Linux has seen widespread adoption for embedded systems ever since.

This whitepaper aims to provide the reader with a broad overview of topics spanning the world of Linux, embedded systems and the **Internet of Things (IoT)**. After looking at what embedded Linux is, this whitepaper will explain the rationale behind its popularity and will conclude with the most pressing challenges facing enterprises within the embedded landscape.

What is embedded Linux?

From a limited number of programs constituting the OS to a full-blown system, blurred lines appear when reflecting on what constitutes embedded Linux. Rather than reciting a hackneyed definition, this whitepaper will consider three distinct perspectives, i.e., the system, the kernel, and the distribution, to help paint a more clarifying picture.



[6]

The system

Embedded Linux systems appear in multiple shapes and forms. They vary from one another and are widespread across virtually every technology segment. Progress in semiconductor technology has substantially helped spur Linux's adoption in the embedded world, with many reasons accounting for such growth. **System on a Chip (SoC)** hardware with low-power and small footprint is increasingly becoming developers' premier processor choice. Also, flash memory prices decreased over time, providing the necessary mass storage for the Linux filesystem[5]. Consequently, almost every embedded system can run Linux – and it does!

An embedded system typically performs a dedicated function, is resource-constrained, and comprises a processing engine. At the risk of over-simplifying matters, one can delineate three broad categories based on the size of an embedded Linux system: [7]

- Small-sized systems comprise a low-powered **central processing unit (CPU)** with at least 2 MB of **read-only memory (ROM)**, and 4 MB of **random-access memory (RAM)**.
- Medium-sized systems have around 32 MB of ROM, 64 MB of RAM, and a medium-powered CPU.

- Large-sized embedded systems have powerful CPUs and a larger memory footprint.

Embedded systems lighter than the above cannot incorporate microprocessors with on-chip memory management unit (MMU) hardware. Consequently, developers of smaller embedded systems, usually incorporating MMU-less microprocessors, could not leverage Linux in their designs until the advent of µClinux. µClinux, merged into mainline v2.5.46 in 2002 [8], uses a flavour of the Linux kernel and uClibc, a stripped-down C-library, and can support hardware without an MMU (CONFIG_MMU).

Nowadays, Linux's presence in embedded systems is ubiquitous and spans virtually across all spending categories, from automotive to home energy management. As costs decline, analysts expect considerable innovation in the embedded consumer space [9]. The table below reports some example systems relying on embedded Linux.

Product	Function	Linux Kernel	RAM	Flash
BioEntry R2 [10]	Biometric access control	3.x [11]	32MB	32MB
ZIPABOX2 [12]	Smart home server	4.x [13]	256MB	4GB
Alloy Smart Hub [14]	Home automation	4.1x [15]	256MB	4GB

As a recap, an embedded Linux system denotes an embedded system running on the Linux kernel. Let us focus on the remaining two pieces.

The Linux kernel

The Linux kernel is a member of the family of Unix-like OS kernels, with AT&T Bell Labs devising the first version of Unix back in 1969. Being proprietary, Unix, the first portable OS, stimulated the inevitable development of free and open-source alternatives like Linux and, among the many others, FreeBSD, NetBSD, and OpenBSD.

At system startup, the Linux kernel loads into RAM and stays in memory throughout the session duration. As the first program to load, it is the fundamental core of an OS in that it has complete control over everything occurring in the system.

Similar to how user commands occur in the so-called user space, the kernel executes processes and handles interrupts in kernel space.

Up to kernel version 2.5, one could understand the expected support level, i.e., stable or in development, of a Unix-like Linux kernel based on its number. Three numbers in dotted notation denoted the kernel version, e.g., x.y.z., with the second referring to the support level: x.even.z for stable kernels and x.odd.z for development or experimental versions. The version numbering of the Linux kernel changed with 2.6, with the second number no longer denoting stable or development versions.

The quickest way to identify the Linux kernel running on a machine is to type *$ cat /proc/version_signature* on the terminal:

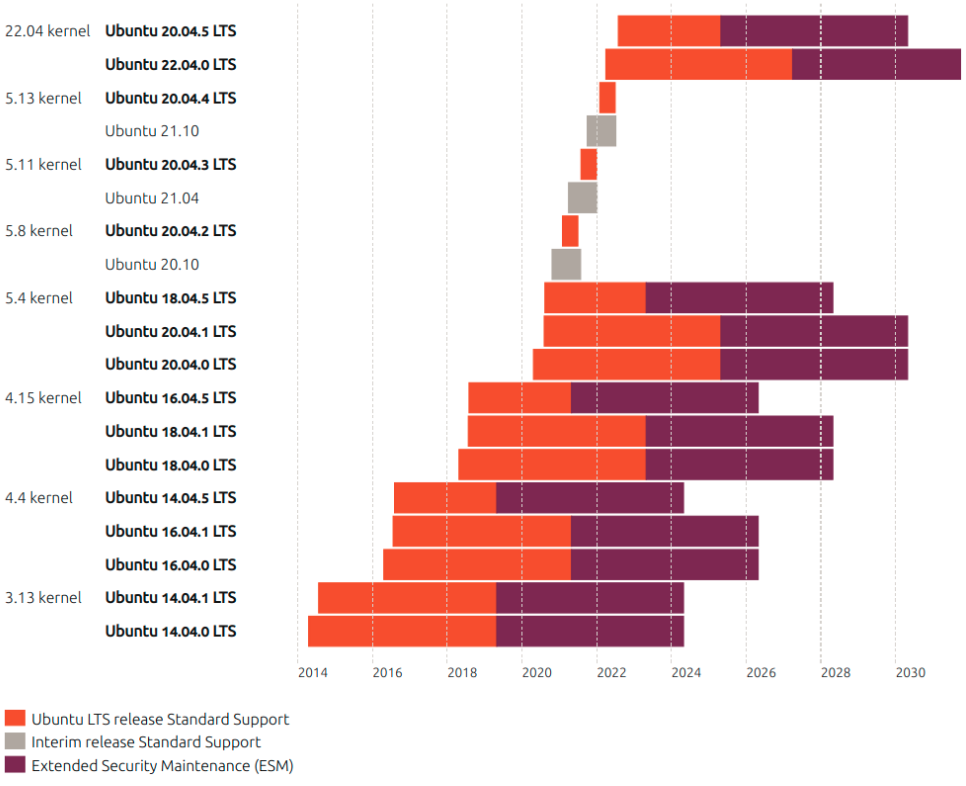
```
edoardo@edoardo:~$ cat /proc/version_signature
Ubuntu 5.4.0-94.106-generic 5.4.157
```

The Ubuntu Linux kernel release is, in the above case, *5.4.0-94.106-generic*. The kernel version is *5.4*, the same as the upstream stable kernel version.

.0 is an obsolete parameter left from older upstream kernel version naming practices. *-94* denotes the **application binary interface (ABI)**[1] bump for this kernel, whereas *.106* is its upload number. *-generic* denotes the default Ubuntu kernel and, finally, *5.4.157* is the mainline kernel-version.

One thing to note is that Linus never shipped an "embedded version" of the Linux kernel. As an embedded developer, you may not require a tailored kernel for your system and might rely on an official release instead. However, it is often the case that you may need a kernel configured to support your custom hardware, as the kernel build configuration found in an embedded device usually varies from the one in a server or workstation.

Canonical, the world's leading provider of enterprise open source solutions, publishes Ubuntu releases based on the latest Linux kernel with a regular cadence, enabling the community, businesses, and developers to plan their roadmaps with access to the newest open-source upstream capabilities. Version numbers are YY.MM, i.e., releases of Ubuntu get a development codename (e.g. 'Focal Fossa') and are versioned by the year and month of delivery - for example, Canonical released Ubuntu 20.04 in April 2020.



1 The Linux kernel ABI denotes exported functions that drivers require to operate in kernel space.

The distribution

The final perspective to define an embedded Linux system is that of a distribution (or “distro”). As an umbrella term, it usually denotes software packages, services, and a development framework on top of the OS.

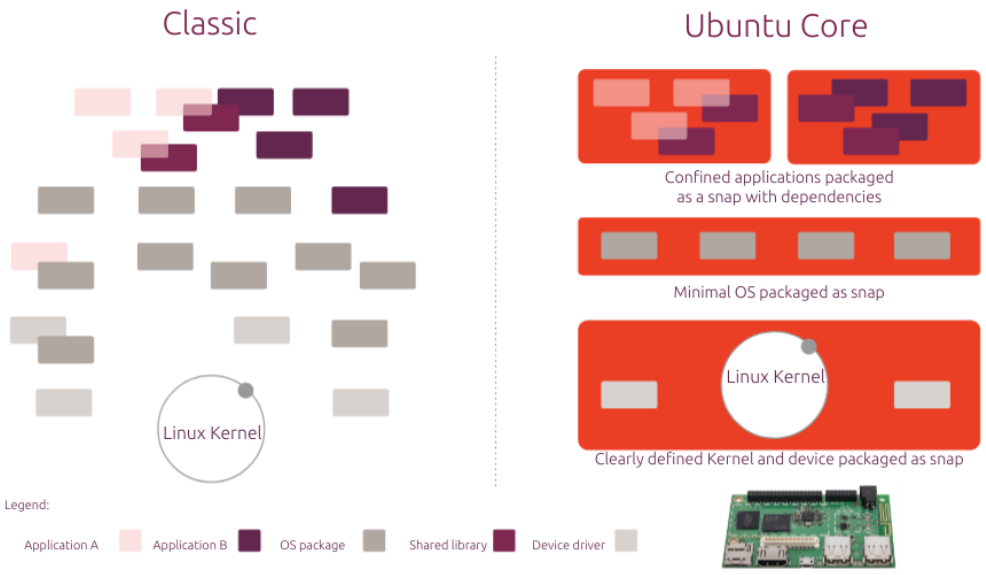
After boot, the Linux kernel mounts a root file system with system libraries, critical files, and loadable modules. Hence, even a tiny embedded Linux distribution comes with the necessary executable target binaries, tools, and utilities like glibc and busybox, populated in a directory structure on a root file system.

A glance at distrowatch.com provides an understanding of how ample the choice of different Linux distributions is. The reason behind the proliferation of so many distros is rooted in the open-source philosophy behind Linux. With the freedom of choice in using and modifying the software for whichever intent one may have, come all the Linux distribution options.

Of course, not all Linux distributions are suitable for an embedded environment. Arguably, a distribution for embedded applications differs greatly from a desktop environment’s one. While both may include startup scripts to initialize network interfaces, a desktop Linux distro will likely support a full-blown GUI stack that may instead be lacking in an embedded environment.

A **Snap** is a containerised software package built as a .yaml file, easy to build and deploy without modification across all Linux distros.

Software vendors like Canonical provide commercial Linux distributions for embedded applications. Ubuntu Core, the flavour of Ubuntu for embedded devices, is an example of an embedded Linux distro designed to be the most secure platform for connected devices from first boot. The embedded Linux OS provides apps as Snaps, a compressed filesystem with a single metadata file describing the security profile and desired integration. The Ubuntu Core image is an all-snap edition of Ubuntu. With Ubuntu Core, the base OS is a Snap, suited for highly reliable and secure large-scale appliance deployments. Ubuntu Core 20, the minimal version of Ubuntu 20.04 LTS for IoT devices and embedded systems, strengthens security with full disk encryption, device recovery, and secure boot.



Embedded applications and microcontrollers

The picture of the embedded landscape would not be complete without the mention of microcontrollers. A vanilla version of the Linux kernel will not run on an MCU due to storage requirements and performance considerations, e.g., expected CPU frequencies lower than 200 MHz. Linux requires an MMU and will thus not run on a Cortex-M or other popular architectures used for microcontrollers. There are, of course, exceptions to the rule, as in the case of µClinux mentioned above.

As we have seen previously, the choice of OS and underlying systems can make a world of difference. Many developers of microcontrollers-based embedded systems experience the daily struggle of flashing code to their boards to discover it does not work and lacks debugging output. A painful board bring-up process paired with the reality of having to port code and implement a novel software stack due to the lack of drivers infrastructure, complete the picture, making the whole process a rather irritating experience. The possibility of innovating more and toiling less has had many engineering teams considering migrating to the world of Linux. The Linux ecosystem, known for its extensive debugging tools and drivers infrastructure, would enable them to reduce time-to-market from engineering specs to a working product.

Interoperability, security, languages, and library support call for rapid adoption of Linux in virtually any embedded device. However, if you come from the microcontroller world, you know there's one more piece to complete the puzzle: *real-time*. Indeed, the limited priority scheduling in the Linux kernel is possibly the main hindrance left facing microcontrollers-based engineering teams when pondering adopting Linux for their embedded applications.

Embedded Linux and Real-Time

Three common ways to describe an embedded OS capable of responding to events within a specified deadline are:

- **Hard**, in those cases where system failure occurs if the deadline is unmet
- **Firm**, when the system can withstand a few missed deadlines
- **Soft**, with the system degrading if the deadline is unmet but can continue its operations

Real-time in the kernel context denotes a deterministic response to an external event, aiming to minimize the response time guarantee.

A common misconception is that real-time must result in optimized performance, usually stemming from video applications so described because of the lack of perceived latency. Although they are performant enough to remove any human notice of deadline failures, the latter are often just best-effort systems. Rather than the performance connotation around real-time, one needs to consider the consequences of a missed deadline when assessing if they warrant hard real-time. For instance, a deadline on the order of seconds is unlikely to be missed on a powerful multi-GHz CPU with proper tuning.

If an embedded system requires real-time, Linux developers usually rely on the PREEMPT-RT patch. PREEMPT_RT, hosted at the Linux Foundation, is the name of one of the patches to implement a priority scheduler and other supporting real-time mechanisms. Although it has become the de facto Linux real-time implementation, many others are available, e.g., Xenomai, RTAI, and RT-Linux.

Although preempt-full Linux kernels provide real-time performance, this may come with an operational cost. A real-time OS minimises response latency to events, not optimised throughput. On the other hand, so-called low-latency Linux kernel configurations offer near real-time performance without the overhead of real-time alternatives. Enterprises moving into real-time often regard latency as more important than throughput, whereas a low-latency kernel provides a better balance for situations where the latter is an important consideration. The low-latency Ubuntu Linux kernel enables a well-balanced solution reducing the overhead while maintaining responsiveness with entirely upstream code. It provides a lower maintenance burden than real-time alternatives in that PREEMPT_RT is an intrusive patchset that may not be compatible with all required drivers and may require debugging and reworking. On the other hand, low-latency is a configuration flavour of mainline, with the kernel working as intended. Furthermore, the low-latency Ubuntu Linux kernel provides the highest preemption level available in the mainline kernel (PREEMPT).

Unless it does not meet an exact preemption specification, a low-latency solution efficiently services most low-jitter, low-latency workloads when coupled with a 1000 Hz tick (HZ_1000) timer granularity. Dedicated devices (e.g., life-supporting medical equipment) are instead better suited to a preemptive solution than a balanced one (reducing overhead while maintaining responsiveness). If the latency requirements of an embedded application are tight and the consequences of a missed deadline are catastrophic, then PREEMPT_RT may be more appropriate. On the other, low latency can be a good fit in case of less stringent latency needs and less dire consequences, or if the hardware handles most of the real-time performance.

Whether a use case needs PREEMPT_RT or not depends on the latency requirements of the embedded application and the consequences for missing a deadline. Knowing those metrics, device vendors should be able to make an informed decision.

The Low latency and real-time kernels for telco and NFV report compares a low latency Ubuntu kernel to an 18.04 kernel patched to be fully preemptive, or real-time, with the generic kernel as a baseline. The conclusions from that analysis are reported below:

Real-time kernel	Low-latency kernel
• Fully preemptive, provides deterministic response time to service events • Extreme latency-dependent use cases • Failure if missed deadline • Higher power usage, reduction in throughput	• **Soft** real-time, attempts to meet expected deadlines (missing is not fatal) • Lower system overhead, less costly to maintain, no extra kernel patches • More favourable for throughput and available CPU for userspace processes

Developers have traditionally overlooked Linux for real-time systems because of its limited priority scheduling and non-deterministic behaviour. As patches like PREEMPT_RT remove unbounded latencies, this is no longer the case.

By becoming better at delivering real-time performance, engineering teams can simply no longer afford to be flippant about adopting Linux.

Why use embedded Linux?

The post-pandemic tech world mandates open standards and interoperability to ensure an agile approach in the embedded landscape. From being open-source to scalability, developer support, and tooling, myriad reasons justify why Linux is the perfect candidate for an embedded system. As an in-depth discussion for each would require extensive treatment, this whitepaper only discusses a few prominent ones:

- Hardware support
- Networking
- Modularity
- Commercial support

Hardware support

Linux runs on a wide range of processors, not all shipped in embedded systems. The 32 and 64-bit ARM, x86, MIPS, and PowerPC are the most well-known processor families on which Linux runs.

Processors below 32-bit do not support Linux, ruling out some embedded systems.

While a 32-bit processor can store 2^{32} (one bit is either a 0 or a 1, hence there are two possible combinations) values, a 64-bit processor can intuitively handle more memory addresses. For a more in-depth discussion on hardware support, you can check the rich ecosystem of platforms for Ubuntu Core here.

Networking

Install NetworkManager on Ubuntu Core right now and get busy configuring your network devices by following this guide.

Support for WiFi, **mobile broadband (WWAN)**, or Ethernet connectivity are just some of the networking capabilities expected out-of-the-box in most consumer products. Whether the need is to configure the system's IP address or SSH into the embedded device, Linux supports a rich stack of networking protocols. System network services like the NetworkManager snap can help configure network devices.

Modularity

Confirming its presence at the bleeding edge of IoT workloads, Canonical innovates by releasing Ubuntu Core images crafted for purpose-built processors for embedded devices.

Modularity is the third reason behind Linux's prevalence in embedded systems. With several software packages coming together to form a Linux OS stack, developers often desire to tailor it to their platform and purpose.

As part of a vibrant ecosystem driving innovation at the edge, hardware vendors and its IoT technology partners like Canonical collaborate to promote the deployment of high-performance embedded Linux devices.

Commercial support

A sometimes overlooked consideration when choosing an OS for an embedded system is the availability of commercial support. Freely available, community-maintained build systems, such as Yocto and Buildroot enable developers to create custom Linux distros for several hardware architectures. Unfortunately, the various challenges that community projects do not address are bound to surface when shipping embedded Linux in production.

Embedded Linux vendors fill this gap by providing enterprise-grade support and expertise, substantially reducing time to market.

Canonical

Canonical publishes Ubuntu, provides commercial services for Ubuntu's users, and works with hardware manufacturers, software vendors, and cloud partners to certify Ubuntu. Ubuntu Core is Ubuntu for IoT and Edge, the best-in-class enterprise-grade solution designed for production and operations at a mass scale with Long Term Support. Ubuntu Core is an OS with a modular and simple architecture, built on snaps, the new universal Linux packaging format. Canonical's team builds, patches, and maintains Ubuntu Core for embedded devices.

Several disadvantages await embedded developers when rolling their own Linux distro versus relying on a commercially supported solution like Ubuntu Core. It's not only expensive for an enterprise to provide its own set of fixes, security patches, and hardware testing – it's also distracting from its core business.

Microsoft

Microsoft offers Windows 10 IoT Core for high-end applications and Azure Sphere in the low-end. Azure Sphere, branded a Linux-based solution for microcontrollers, currently targets a single certified chip produced by Mediatek and can run real-time capable applications on bare metal or real-time cores [16], [17]. However, Microsoft does not provide runtime for the two Cortex-M, reserved for customer code usage and open to developers who may take an existing **real-time operating system (RTOS)** for their real-time code [18].

Linux runs on the Cortex-A, with Microsoft providing an SDK to develop and deploy containerised high-level applications. Processing intense apps may require adding external RAM/FLASH, as space on the device for high-level apps is limited to 256KB of user RAM and 1 MB Flash for customer app binary [19].

As choice in the available embedded OSs keeps expanding, listing them all would be a futile endeavour. Several players have a role in the commercial enterprise embedded Linux landscape: for completeness, one shall also mention Red Hat Enterprise Linux, SUSE with SLE Micro, and Wind River Linux.

Even if one were to ignore the obvious reasons for choosing Linux for an embedded system, from hardware and language support to extensive modularity, would there be any argument left for nudging the most reluctant developer towards the OS? The answer is a clear and resounding yes, expressed in just three letters: **IoT**.

Embedded applications and IoT

How does the IoT fit within the existing embedded landscape? Succinctly, embedded systems form the backbone of large-scale IoT deployments. How so? The IoT comprises a network of Internet-connected devices (the "things" in the "Internet of Things," otherwise called "embedded systems").

With the urge to have everything connected driving the spread of embedded devices, new standards, from Wi-Fi 6 to the 5th generation mobile network, act as the much-needed enablers.

The advent of IoT

The explosive growth of the IoT, with its myriad tightly-embedded connected devices, has brought about a stringent set of hitherto unaddressed demands and requirements to Linux.

Analysts forecast 25.1 billion connected devices by 2023 [9], generating a $1.1 trillion IoT revenue opportunity by 2025 [20]. While the actual figures may differ depending on which market research one looks at, the overall message is clear. The adoption of IoT devices, driven by the proliferation of cheap boards and coupled with the pervasiveness of network connectivity, is affecting virtually all sectors of society.

At the risk of adding some unwarranted doom and gloom to the rosy picture depicted so far, all-new challenges are in store for the embedded Linux developer.

Key considerations for adopting IoT

An embedded Linux OS *on its own* is not the end-all-be-all solution developers thought it was. A turn-key security and update solution is a must to reap the promises of the Industry 4.0 revolution via the next wave of connected devices. This whitepaper will discuss the following considerations for adopting IoT:

- Security
- Software updates

Let's tackle these challenges one at a time, starting with security.

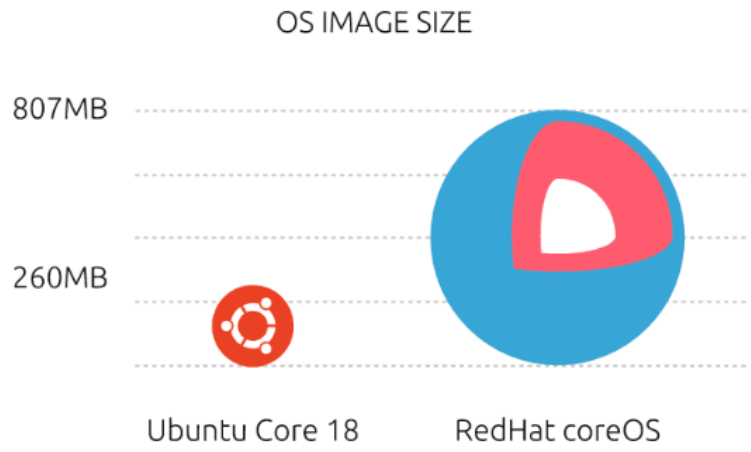
Security

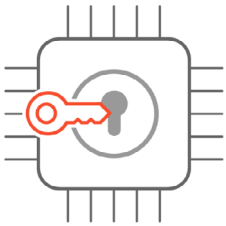
Security considerations notoriously affect every layer of the software stack — the embedded OS being front and centre — as an unpatched vulnerability can put the device manufacturer's reputation on the line.

Whereas safety concerns frequently plague legacy, proprietary code, developers often consider open-source Linux software safe due to its inherent transparency and regular reviews by the community. Despite all that, open-source software is not necessarily secure from attacks, as the Log4Shell incident highlighted [21].

With the proliferation of cybersecurity concerns, consumers expect manufacturers to make strides towards shipping secure embedded devices. Furthermore, the modern pervasive connectivity of our everyday lives exposes embedded devices to ever-increasing malicious attacks.

Making the open-source landscape and the embedded Linux ecosystem more secure and robust is at the heart of Ubuntu's mandate. The first step in this direction lies in reducing the surface of the attack and stripping the embedded Linux OS down to its bare essentials. Not only would fewer packages leave more disk space for your applications and data, but they would also lead to fewer forced changes and bugs to fix. Ubuntu Core is the most lightweight Ubuntu Linux release, ideal for embedded applications.





The security challenge is, however, not yet solved. The deployed fleet need to be able to resist low-level boot attacks, guaranteeing the integrity of the boot firmware whilst ensuring the devices only run the purposefully-shipped software. Also, manufacturers must protect the data integrity of the embedded devices in case of physical access.

Security spans a variety of aspects, from mission-critical support to strict confinement, not mentioned here: to solely rely on Linux for an embedded device may not prove sufficient. Manufacturers invariably find themselves looking for an enterprise-grade solution designed for production and operations at a mass scale with Long Term Support.

Every device, even the tiniest one, should be trustworthy, and Ubuntu Core, the most secure embedded Linux ever, was born with such a premise. A secure boot mechanism in the embedded Linux OS proves necessary for that. At the same time, private key-based cryptographic signatures in Ubuntu Core are the most reliable technique to protect sensitive data confidentiality, authentication secrets, and software intellectual property.

Software updates

The second pivotal requirement brought by the proliferation of tightly connected devices is the need for software updates.

A device manufacturer can prevent security breaches by being up-to-date with the latest software patches. In much the same manner, **over-the-air (OTA)** remote updates with fixes to the security vulnerabilities are necessary once deployed devices in the field are compromised.

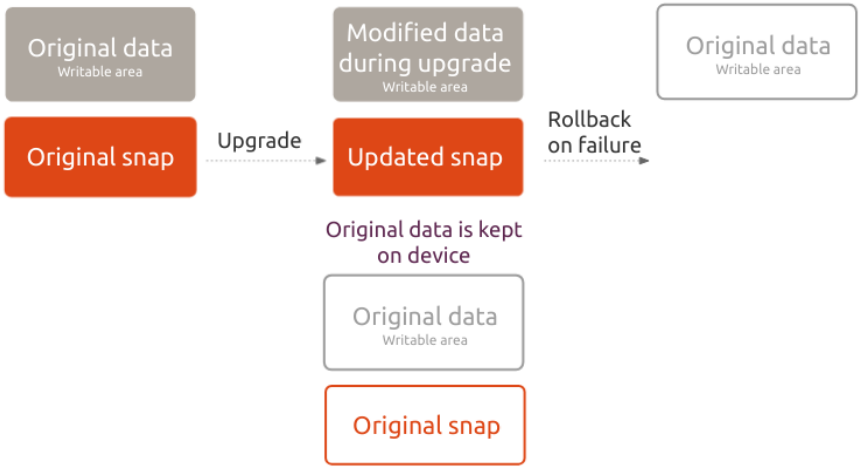
Software updates are not a one-time effort that firms can engage with on an ad-hoc basis. Device manufacturers with a reputation for reliability and security must rely on enterprise-grade, production-tested software distribution machinery to deliver a constant stream of ultra-reliable updates to their Linux kernel, OS, and application. Legacy update mechanisms are not adequate when dealing with large-scale, low-power fleets of Linux devices. The new update paradigm is transactional, OTA, and minimises networking traffic. Read how Ubuntu Core, the new standard for embedded Linux, handles the software update problem for embedded systems.

IoT management

As manufacturers acquaint themselves with the advantages of relying on embedded Linux, they subsequently start questioning the burden of maintaining their systems. From monitoring the embedded device's health and managing applications to capturing telemetry data and application logs, remotely managing Linux-capable IoT devices is crucial. With the rapid spread of connected systems being too wide to ignore, manually managing them is an error-prone endeavour.

IoT management (IoTM) platforms simplify and automate the management tasks by connecting to the embedded devices and querying their states. However, effectively implementing such a solution is a burdensome task as one needs to design the OS with managing applications in mind from the ground up. As is intuitively understood, this is not a typical embedded Linux feature to be shipped at will but is instead a paradigm shift realized by adopting a "management-by-design" approach.

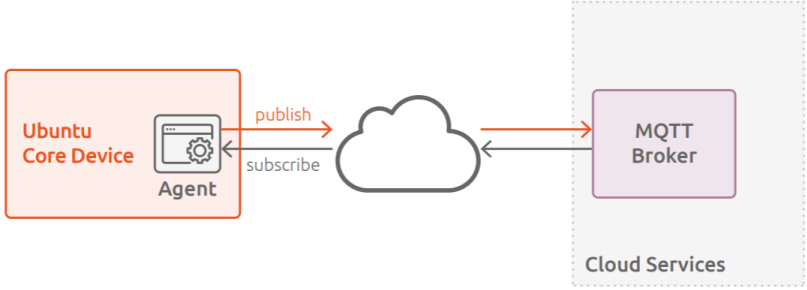
Here Ubuntu Core raises the bar for embedded Linux. Ubuntu Core seamlessly enables the transition from OTA updates to device management by remotely installing, removing, and configuring applications. Each app then operates in its sandbox on the device and ships via OTA deltas following the self-healing mechanism of the OS updates.



Ubuntu Core provides the snapd REST API to manage applications on the device. The API enables the communication over a local socket between an installed application and snapd, the daemon that manages the snap.

A privileged snap in the form of a pre-installed agent on the device provides a link between the snapd REST API and an MQTT broker on the cloud for remote device management.

This whitepaper further covers deploying a simple, central IoT management solution on Ubuntu Core and all things secure IoTM.



Conclusion

After establishing its presence among servers and clouds, Linux is now the OS of choice for embedded systems. Its role within the embedded landscape, together with the challenges faced by the developer in the ecosystem, should be, if not clear, at least better known.

There are obvious reasons to choose Linux for an embedded system, from hardware and networking protocol support to extensive configurability and modularity. However, developers must also pay close attention to the issues of security and updatability if they are to reap the benefits of the explosive IoT growth. Luckily, the kernel experts at Canonical, the commercial sponsor of Ubuntu, are here to help.

That we are in the midst of the fourth industrial revolution should be no surprise. Daily reminders of cutting-edge connectivity and robotics automation help stir the creativity of future engineers and developers. The potential of embedded devices to propel novel efficiencies in legacy verticals is undeniable, but one cannot stress enough the resulting critical challenges. Security is paramount for the successful operation of any IoT system, as device manufacturers must carefully consider the privacy and regulatory needs of the end-user. Similarly, a steady stream of OTA updates to the Linux kernel and the applications enables devices to be free of critical vulnerabilities and patched with the latest security fixes.

Borrowing from the expertise needed to make Ubuntu the most widely deployed server OS on the Internet [22], Ubuntu Core is the best embedded Linux OS tailored to IoT devices and optimised for security and updates. Delve into its documentation today.

References

1: https://groups.google.com/g/comp.os.minix/c/dlNtH7RRrGA/m/SwRavCzVE7gJ
2: https://web.archive.org/web/20150806093859/http://www.w3cook.com/os/summary/
3: https://www.rackspace.com/en-gb/blog/realising-the-value-of-cloud-computing-with-linux
4: https://ubuntu.com/cloud/public-cloud#:~:text=Close-,Ubuntu%20on%20public%20clouds,of%20enterprise%2Dgrade%20commercial%20support.
5: https://jcmit.net/MemoryDiskPriceGraph-2021Oct.jpg
6: https://unsplash.com/photos/rZKdS0wI8Ks
7: https://www.oreilly.com/library/view/building-embedded-linux/9780596529680/
8: https://web.archive.org/web/20130725041808/http://marc.info/?l=linux-kernel&m=103645181102114&w=2
9: https://www.gartner.com/en/documents/3840665/forecast-internet-of-things-endpoints-and-associated-ser
10: https://www.supremainc.com/en/hardware/compact-fingerprint-reader-bioentry-r2.asp
11: https://www.supremainc.com/uploadfiles/Technical_Resources/20200529104141218.pdf
12: https://www.zipato.com/product/zipabox2
13: https://www.zipato.com/wp-content/uploads/2019/02/Zipabox2_Data_Sheet_v7_preview.pdf
14: https://alloysmarthome.com/smart-hub/
15: https://go.smartrent.com/rs/254-CGZ-341/images/alloy-smarthome-user-manual.pdf
16: https://www.mediatek.com/products/mt3620
17: https://docs.microsoft.com/en-gb/azure-sphere/app-development/applications-overview
18: https://azure.microsoft.com/en-us/blog/azure-sphere-s-customized-linux-based-os/
19: https://docs.microsoft.com/en-us/answers/questions/183716/azure-sphere-user-accessible-storage.html
20: https://www.gsma.com/iot/wp-content/uploads/2018/08/GSMA-IoT-Infographic-2019.pdf
21: https://ubuntu.com/blog/log4j-vulnerability-2021
22: https://w3techs.com/technologies/history_details/os-linux

Interested in a detailed comparison of built vs bought embedded Linux? Read the Embedded Linux: make or buy? whitepaper.

If you want to improve usability, security and time-to-market for your enterprise devices, listen to the webinar introducing Ubuntu Core 20.

And finally, check the app store for Linux

Canonical is the publisher of Ubuntu, the OS for most public cloud workloads as well as the emerging categories of smart gateways, self-driving cars and advanced robots. Canonical provides enterprise security, support and services to commercial users of Ubuntu. Established in 2004, Canonical is a privately held company.

ubuntu
Delivered by Canonical