White Paper

A new vision for the edge: The Secure Edge-Native Architecture (SENA)

A new solution architecture for edge computing to integrate security from silicon to software, from deployment to ongoing management.

Authors

Ettore Di Giacinto

Head of Open Source, Spectro Cloud

ettore@spectrocloud.com

Bryan Rodriguez

Principal Engineer & Edge Innovator, Intel

bryan.j.rodriguez@intel.com

Michael G. Millsap

Director of Edge Platforms, Intel

michael.g.millsap@intel.com

Abstract

Edge computing is becoming increasingly popular. As a model it offers advantages in terms of performance and network efficiency. By locating computation closer to the humans interacting with the physical world, it accelerates many data-intensive or real-time use cases.

From AI-enabled medical software at hospitals, smart shopping and hospitality, to drones maintaining power grids or inspecting crops, today's rich container-based applications use vast amounts of data generated at remote endpoints. The desire to push those new applications to improve speed and accuracy is driving today's aggressive edge transformation.

However edge environments come with unique challenges that make it difficult for teams to efficiently deploy, provision, operate and manage at scale. Unlike controlled environments such as data centers and clouds, edge locations introduce a set of new requirements that conventional hardware and software solutions cannot address: little to no supervision, unpredictable or complete lack of connectivity, sizing constraints and an increased attack surface. In order to address those new requirements, today's edge requires purpose-built solutions based on the tight coordination between hardware and software.

In this white paper we introduce the Secure Edge-Native Architecture (SENA), a new solution architecture for edge environments to address these challenges and considerations. Although SENA refers specifically to Kubernetes-based edge deployments, its principles can be directly applied to edge computing environments based on other technologies.

The SENA solution architecture can be broken down into four main areas:

- **Deploying and on-boarding of trusted edge devices**
- **Provisioning a verified software stack**
- **Operating the edge runtime with confidence**
- **Managing edge locations at scale**

In the following pages we will look at the architectural requirements and key concepts ultimately leading to a reference framework for implementing SENA in any modern infrastructure for integrated security from the silicon to the application.

This paper is for anyone designing and deploying edge computing environments, including IT Operations, Platform Engineering, DevOps, or IT and Cloud architects.

Table of contents

Key security requirements for modern edge computing

Deploying applications on edge devices presents unique challenges for security, ranging throughout the lifecycle of the infrastructure. The edge is different from a data center. Edge computing devices may be in motion, baking in the sun mounted on a utility pole, out on an oil rig connected via satellite or powered by batteries ... and likely never behind security guards in a locked cage.

Edge security challenges can be broadly classified into three categories:

- **Physical security:** edge devices are often located in remote and physically insecure locations, making them vulnerable to physical attacks.
- **Network security:** communication channels between edge devices and central management planes can be insecure, making them vulnerable to cyber attacks.
- **Access:** edge devices may be managed by multiple stakeholders, including third-party service providers, making it difficult to maintain consistent security policies across the entire ecosystem.

In addition to security, teams looking after edge deployments and locations have to take into consideration sites with sizing constraints, intermittent or no connectivity, including designs where edge locations need to be "air-gapped" by design.

Depending on the use case, edge locations might also be completely unsupervised, or manned by IT or other staff that is not skilled to deal with the components of a cloud-native stack, such as Kubernetes, which comprise multiple layers of heterogeneous components that need to be maintained.

Finally, the sheer scale of typical edge deployments — across potentially thousands of sites — significantly increases complexity and risk.

All of these challenges make it extremely difficult for organizations to optimize their decisions on combinations of hardware and software components, while maintaining flexibility and efficiency in order to address their specific edge use cases.

The primary objective of the **Secure Edge-Native Architecture (SENA)** is to establish a new industry standard in the form of a well-defined framework for solving these challenges, approaching them from a holistic perspective, and leveraging the best of hardware and software working in unison.

SENA describes best practices for onboarding devices, provisioning a verified software stack, and securely operating workloads at boot and runtime. It also accounts for a wider view of how operations teams can securely manage the complete lifecycle of their deployment.

The white paper is organized into two sections:

- **The first section** identifies and discusses the necessary requirements and key concepts for a modern edge architecture.
- **The second section** explores how SENA incorporates those concepts into architecture components and design principles that come together to address the requirements, with additional considerations for management at scale.

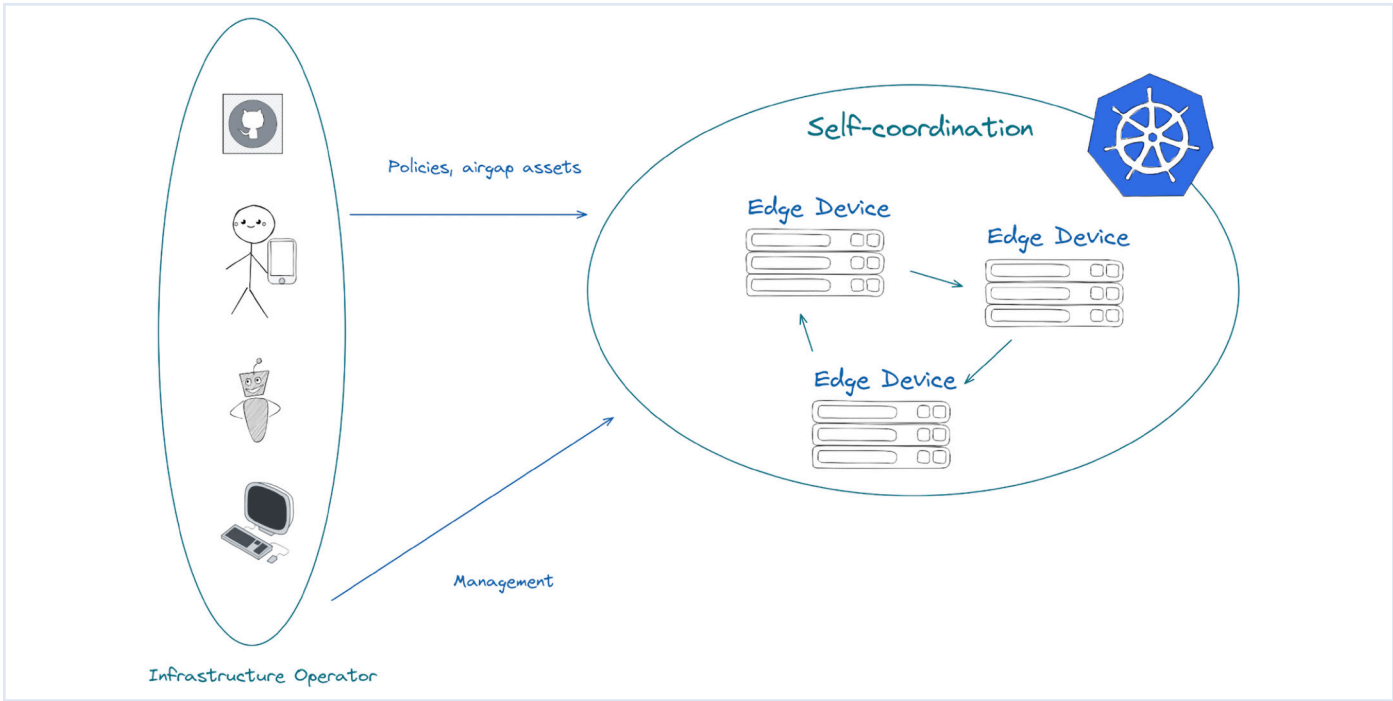
Requirement 1: Deploying trusted devices quickly and easily

Edge computing devices may be deployed in all kinds of scenarios, from smart homes to kiosks, industrial settings to moving vehicles. And each edge computing use case can have different implementation requirements depending on the hardware, network and Kubernetes cluster topology needed for its workloads.

This section will discuss some of the foundational requirements and considerations when deploying edge devices.

Flexibility and standardization

A key requirement is flexibility during deployment, to allow for both customizations to the software stack, as well as being able to scale to thousands of machines with ease through standardization of the process.



Flexibility also extends to preparation of edge computing hardware. There are several options for deploying nodes at the edge, depending on an organization's requirements: pre-configuring nodes before shipment, pre-imaged machines from the vendor, or "just-in-time" provisioning for example with the help of an attached infrastructure.

Any edge computing environment may be heterogeneous at initial deployment, or become so over time. As a result, it's important to choose software ingredients that enable the most flexibility for customization and manageability at scale.

Trusted remote device onboarding

Onboarding is the act of "associating" the device with a management plane or service — in this case, with Kubernetes itself or with a Kubernetes management platform.

In order to be safely onboarded and integrated into the environment, the system needs to trust the device, which means verifying that it has the necessary security credentials and that it has not been impersonated by a bad actor.

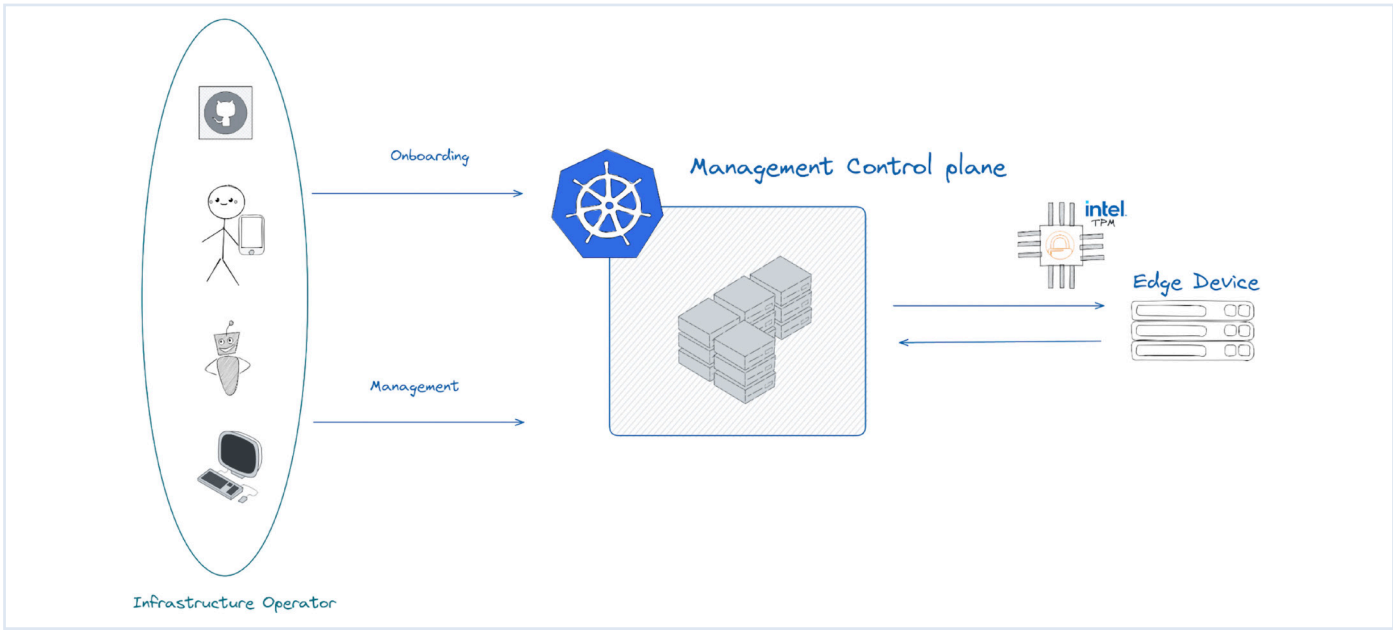
Traditionally, this was achieved by having a trusted administrator onsite to provide credentials, such as keys, passwords and certificates, in a secure manner. But with the distributed nature of edge systems, getting trusted administrators to each site can be extremely costly and slow down deployments.

FIDO Device Onboard (FDO) is a device onboarding scheme from the FIDO Alliance[1], that enables the secure provisioning of secrets in an automated manner. FDO enables devices to be remotely, yet securely, onboarded without the need to send trusted administrators.

Zero-touch device registration

Whether there is a management plane or platform to onboard devices or not, edge devices should be easy to set up and enroll/register. The easier the process, the less cognitive load on the user that could lead to human error and, ultimately, security incidents[2].

There are many approaches to automating and simplifying the authentication flow, including UI onboarding, code scanning, QR code, single auto-installing binaries, automatic rollout via network. Any of these are preferred over multi-step and manual configurations.



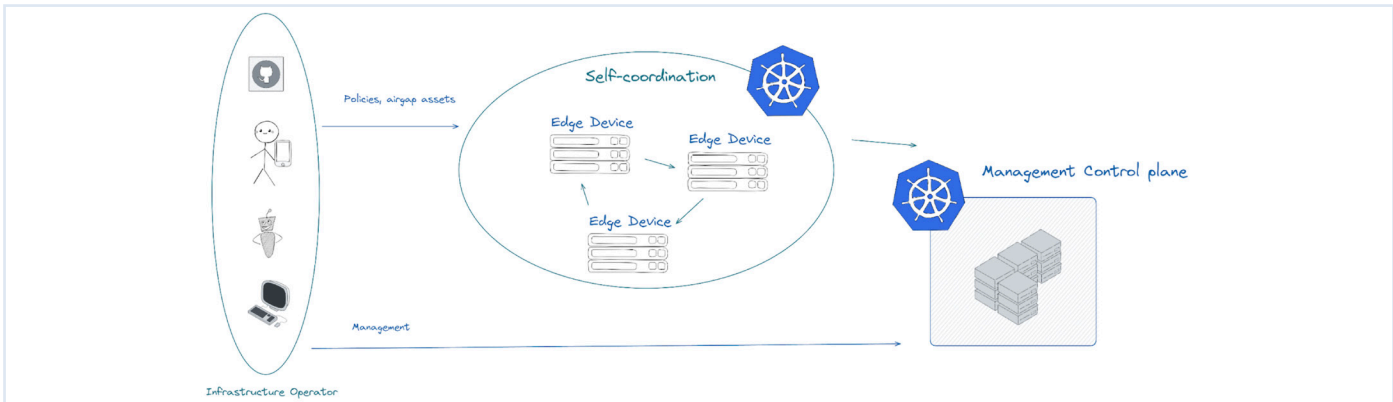
Autonomous provisioning for disconnected environments

Edge architectures should be designed so they are autonomous: they continue to function even when network connectivity back to the management control plane is constrained or uninterrupted.

There are many scenarios where connectivity will be absent:

- Due to the nature of the location, for example in rural areas
- By design, such as in autonomous vehicles
- Where security and regulation mandate “dark” or “air-gapped” sites, isolated from the public internet or even a private WAN
- In the event of a temporary network outage or central management plane downtime

In such scenarios, edge devices must be able to autonomously coordinate, or at least be able to provision without need for a remote party in a separate network. For example, it may be possible to configure a node on the same network segment as a management plane.

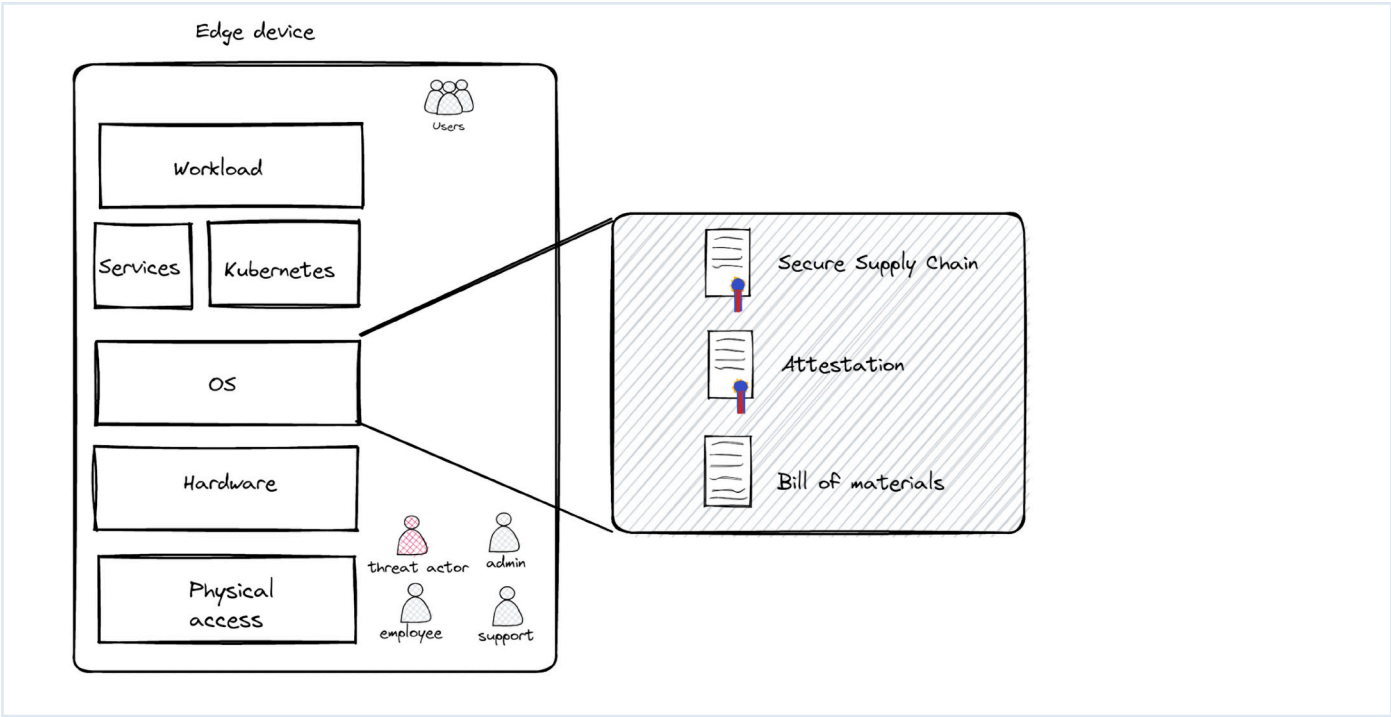


Requirement 2: Provisioning a verified software stack

Deploying the trusted device hardware and onboarding it into management is only the first step. Next we need to verify the full software stack that we plan to deploy to the device: all the software artifacts, from OS to application workloads.

The architecture needs the ability to transparently track all components of the software infrastructure in order to avoid supply chain attacks.

Devices need to be activated only with a verified Operating System (OS), an attestation of provenance, and a bill of materials for all the components on the system.



At the same time, developers must retain choice and flexibility over their software stack to allow for customizations when choosing hardware appliances.

We recommend that the overall design must comply with Supply Chain Levels for Software Artifacts (SLSA) standards, and provide Software Bill Of Materials (SBOM) artifacts to allow traceability and attestation of the OS.

Supply Chain Level for Software Artifacts (SLSA)

The Supply Chain Level for Software Artifacts (SLSA) [3] is designed to address the problem of software supply chain attacks, where malicious actors compromise the security of software components during the development, testing, or distribution phases.

The SLSA model defines a set of security requirements for software artifacts that must be met at each stage of the supply chain, including provenance, integrity, and authenticity.

SLSA also provides a set of trust levels that can be used to classify the security posture of software artifacts, enabling organizations to make informed decisions about which components to use in their systems.

SLSA has been widely adopted by industry leaders, including Google[4][5], Microsoft[6], and Red Hat[7].

Software Bill of Materials (SBOM)

Modern software is extremely complex, and any given application may include many libraries or modules — perhaps created by third parties — and containing known vulnerabilities.

The Software Bill of Materials (SBOM) provides transparency and accountability in the software supply chain, allowing stakeholders to understand the origin and security of the software components they use.

It does this by documenting all the various components that make up a software product, including their dependencies and relationships. Common Vulnerabilities and Exposures (CVEs) and security alerts for each of these components are proactively scanned for and reported, and remediated as necessary.

SBOM has become a critical component of software development and management.

Using SBOM and SLSA together

SBOM and SLSA can be effectively used in tandem.

By integrating the SLSA model into the software development process, a signed SBOM can be directly attached to published artifacts. This allows for the seamless linking of software components to their provenance and integrity, streamlining vulnerability tracking. The use of these tools in conjunction can significantly enhance the security and accountability of the software supply chain.[8]

We can take this further when we compose the entire deployment stack — OS, Kubernetes and other software elements — as a single artifact like a container image. This image can be signed and attested, allowing us to use existing policy agents to enforce security profiles across the whole stack and effectively block malicious or tampered payloads during upgrades[9].

Requirement 3: Operating the edge runtime

Once a trusted device is provisioned with a verified software stack, then comes the application runtime. In this section we will specifically discuss requirements for securing execution and isolating workloads from unauthorized access.

In order to achieve a truly secure system, it is critical to prevent physical and remote tampering or threats to the firmware, and ensure the integrity and privacy of user data is accessible remotely or locally only by the authenticated actor.

In addition, edge devices must not be susceptible to physical alteration through peripherals such as malicious USB sticks or remote access[10].

Application workloads have to be isolated as much as possible from the underlying OS and from other processes, regardless of privilege level, running on the same device, avoiding memory tampering, or cold boot attacks.

We suggest that a modern edge architecture must fulfill the prerequisites of immutability, integrity and atomic upgrades and align with Confidential Computing principles, including workload isolation and privacy through hardware protections with encrypted workloads. In order to achieve these prerequisites, hardware with secure attestation capabilities is required.

Immutable OS for consistent state

Edge computing devices are often deployed in environments where they are subject to external factors such as temperature fluctuations, power outages, and physical damage, as well as potential unauthorized access.

These factors can lead to unintended modifications to the system, such as configuration drift and inconsistencies that are difficult to diagnose and troubleshoot.Internal factors can also cause configuration drift, for instance an upgrade may generate files that are not tracked by the OS.

In mission-critical edge workloads, any deviation from desired configuration could lead to downtime and loss of revenue.

By making the edge OS immutable, we can prohibit modifications to system files and directories, ensuring a consistent and reproducible system state.

Immutability can also be used for testing changes in the infrastructure stack in a lab environment prior to releasing them in production.

Any drift that may occur during testing can be easily identified and addressed, reducing the risk of unexpected issues arising in the live environment. This approach helps to mitigate the impact of human errors or package manager[11] nuances, such as generating untracked files during upgrades, unpredictable merging of configuration files.

Containerized OS for granular control

The OS should be treated in a manner similar to applications running in a data center. It is important to have the ability to control upgrades with the same level of granularity that we are accustomed to when deploying applications in Kubernetes.

This includes using canary or blue-green updates, or allowing for scaling with any upgrade methodology. Therefore, it would be advantageous to have the OS packaged in the same form as Kubernetes applications, i.e., as a container image.

Packaging the OS as a container image also enables the OS to be scanned and analyzed with the same tools that are already available in the container ecosystem.

Verified integrity at boot and during operations

Devices should run only the intended, unmodified OS and application stack.

By proactively monitoring edge devices and assessing their integrity, we can ensure that only trusted devices are allowed to join the network and that any unauthorized modifications are quickly detected and addressed.

Mechanisms such as remote boot attestation ensure that the system hasn't been altered at the time of boot.

Runtime integrity measurement validates the integrity of the system and identifies modifications at any point in time during operational runtime.

Zero-downtime atomic upgrades

Every edge deployment will need regular upgrades, either for functional enhancements or security patches. It is vital to avoid risk and downtime during upgrades, given edge workloads are often mission-critical and lack expert IT personnel on site to recover a failed device.

There are various approaches to upgrading single- and multi-node setups, but in this paper we will look at implementations that rely on Kubernetes as the main controller to apply upgrades to devices. This allows operators to set up a rolling upgrade strategy, where nodes are upgraded one at a time while the others continue to operate as normal, or a blue-green deployment, where the new version of the OS is deployed alongside the current version, and nodes are switched over once the new version is tested and validated.

Beyond ensuring availability, it is also essential for an OS to allow for atomic upgrades, particularly for critical workloads. Atomic upgrades are made possible through the use of a single image-based process that employs a simplified approach for system upgrades. This process involves creating a transactional image, which is then applied to the system.

This approach mitigates exposure to intricate upgrade procedures and minimizes potential vulnerabilities. It involves creating a single image of the upgraded system that includes all the necessary changes and dependencies. This image can then be deployed in its entirety, replacing the previous version of the system.

By using this image-based process, the upgrade operation can be carried out atomically, ensuring that either the entire system is upgraded successfully or none of it is. This atomicity guarantees that there will be no partial upgrades, which can lead to inconsistencies and potential failures.

In addition, using a single image simplifies the management of the upgrade process. There is no need to manage multiple components or worry about compatibility issues between them, since all the necessary changes are included in the image, making the process much easier to handle. Image-based updates prevent "snowflake compute", where configuration drifts over a large period of time can lead to degradation, outages, and even security incidents.

Finally, the single image can be validated before deployment, ensuring that it is functioning correctly and does not contain any vulnerabilities. This validation step further reduces the risk of failures or security breaches during the upgrade process.

Confidential Computing

As defined by the Confidential Computing Consortium12, Confidential Computing refers to "the protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment (TEE)".

TEEs are commonly referred to as Secure Enclaves, and they have three attributes: data integrity, data confidentiality, and code integrity.

Confidential Computing dictates that the installed OS should only be able to access the minimal user data required for running the workload. This data should be safeguarded by hardware so that it can be accessed by the OS only for reading the execution instructions. The OS should not be able to read the data that the workload processes.

This is very important in edge scenarios as devices are exposed both to physical attack vectors as well as remote. Leveraging hardware-specific instructions guarantees that no one can view or alter running workloads on a system, securing data in-use.

Data encryption

Encryption at rest is essential to ensure privacy and integrity of sensitive user data in environments at risk of physical attack.

Encrypted keys should be hardware-protected and only exposed to the OS for decryption if the system is verified to be unaltered. Given edge devices may be operating without reliable connectivity, key management should support both offline and online scenarios.

In an offline deployment, there is no external device or server; the keys used to encrypt the partitions are stored in a cryptographically safe area of the device.

In an online deployment, an external service (for instance a Key Management Server, KMS) stores part of the secrets needed to decrypt the drive. The KMS enables the machine to boot by providing the encrypted secrets, or passphrases to unlock the encrypted drive.

Effective key management is critical in thwarting attempts to steal a node device and decrypt the data. It also helps prevent unauthorized access to data through alteration of the OS at rest or exposure of encryption keys.

Proactive security

Encrypted partitions help keep user data confidential in the event of a device being lost or stolen. But what happens when the attacker attempts to use the device, powers it on and connects it to the internet?

After a security alert has been raised or when a device has been disconnected from the network without notice, we recommend flagging the device for additional proactive measures when it is brought online again, for example quarantining the encrypted partitions or blocking the device from registering again.

The Secure Edge-Native Architecture (SENA)

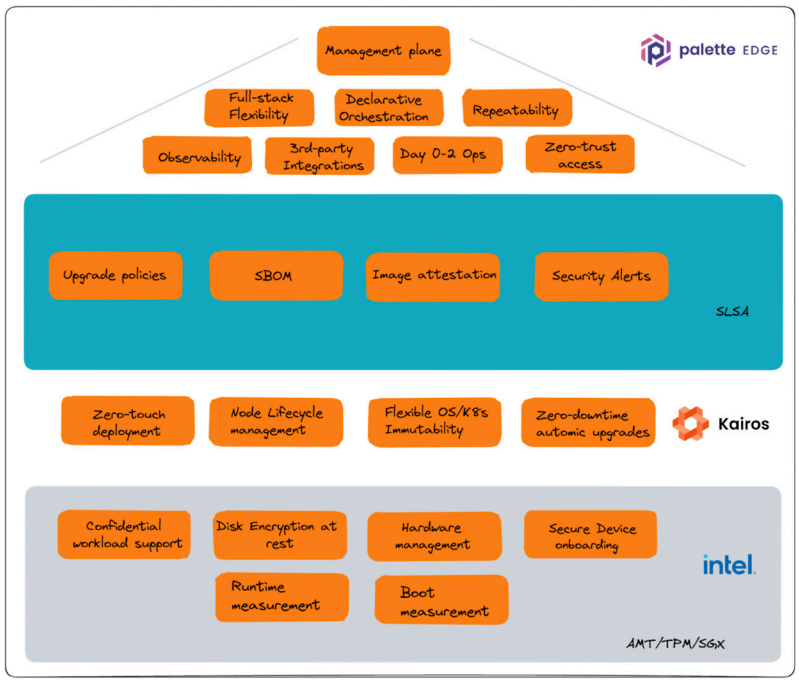
The Secure Edge Native Architecture (SENA) is a comprehensive solution architecture that outlines the tools and practices for a secure modern edge infrastructure.

It addresses the three main aspects of edge security as explored in the previous section of this paper — deployment, provisioning and operating — as well as looking more holistically at how these practices are embedded in an overall management lifecycle at the typical scale of an edge deployment.

A key principle of SENA is to promote the use of universal and industry-acknowledged frameworks, including open source solutions and tools, as well as providing transparent references to capabilities without mandating specific commercial solutions.

In this section we expand on the key concepts and frameworks introduced previously, offering an architecture view and high-level implementation to illustrate how the SENA requirements can be met. To that end we draw on:

- Intel platforms, including Intel® Active Management Technology (Intel®AMT) and Intel® Software Guard Extensions (Intel® SGX)
- The Kairos open source project for creating immutable OS images, along with other open source projects such as Kyverno
- An edge Kubernetes management plane or platform, such as Spectro Cloud's Palette Edge



Introducing Kairos

Kairos is the open-source project that simplifies edge, cloud, and bare metal OS lifecycle management. It offers a unified cloud native API for building immutable, bootable Kubernetes and OS images with the ease of writing a Dockerfile.

With key features such as immutability, security, container-based management, P2P mesh deployment, and distro agnosticism, Kairos empowers users to create their own cloud on-premises with complete control and no vendor lock-in.

With Kairos, managing a Kubernetes cluster with any Linux distribution becomes effortless, providing a streamlined and efficient approach to OS management. To learn more about Kairos, visit https://kairos.io.

Capability 1: Automating the deployment of edge devices

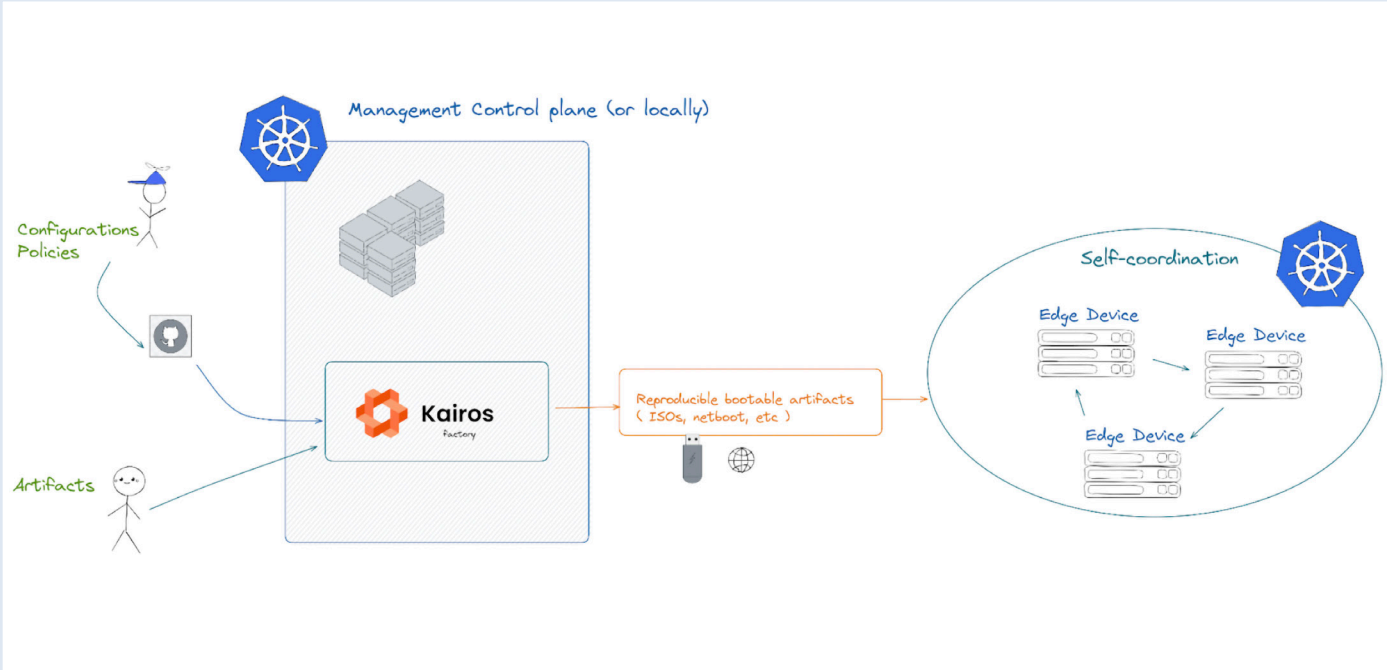
In this section, we will be discussing how different components of the SENA architecture can address the key requirements we introduced relating to deployment of edge devices.

There are a number of solutions and tools that can automate the deployment of edge devices in order to address the need for scale. However, it's worth noting the high fragmentation depending on the Linux distribution adopted.

- Projects like kiwi[13], bootc[14] or Fedora's kickstart provides an automated and programmatic way to deploy a system like SUSE or Fedora.
- Terraform is also a popular tool used to deploy in the cloud, but it can have practical limitations when it comes to bare metal.
- Popular solutions for bare metal include Canonical MAAS and Ironic, which automatically provision and enable horizontal scaling of the infrastructure.
- Cockpit is a popular management interface that is worth mentioning too.

Kairos offers a Kubernetes distribution-agnostic approach without tying to any specific vendor, and scales to several scenarios, providing the required high level of customization.

Its bootstrap mechanism offers a unique solution that simplifies deployment with a decentralized QR code-based process. Kairos enables seamless bootstrapping for any OS from a container image, and can cater for both air-gapped and online installations from a single, simple interface.



The Kairos "factory" can be programmatically orchestrated by Kubernetes: clusters are able to create reproducible artifacts to bootstrap new clusters in a GitOps fashion.

In this model a Kubernetes management plane (such as Spectro Cloud's Palette Edge) can recreate a custom OS artifact, allowing customization of the image used by the devices, and attachment of configuration and policies to the artifacts. This enables scaling configurations and policies to any number of deployments, handling the specifications as custom resources in Kubernetes.

It is possible to generate self-installing artifacts, either locally or with an attached infrastructure: Kairos releases a Kubernetes native API extension controller that allows users to customize images, create specific configurations and finally generate bootable artifacts that can be used to deploy clusters automatically, without any human intervention[15].

If an edge node is not set to be provisioned automatically, a QR code is displayed on the edge device's display. This makes it easy for even non-skilled personnel to install and configure devices without needing any additional software[16]. Additionally, manual installation can still be performed if necessary, either through an intuitive interactive installation or a local web interface. This streamlined process not only reduces the effort required for device management, but also minimizes the potential for human errors.

Kairos can create sidecar artifacts such as ISOs to be used in tandem with the unmodified OS, allowing simpler customization logic without having impact on the management and distribution of customized OS images.

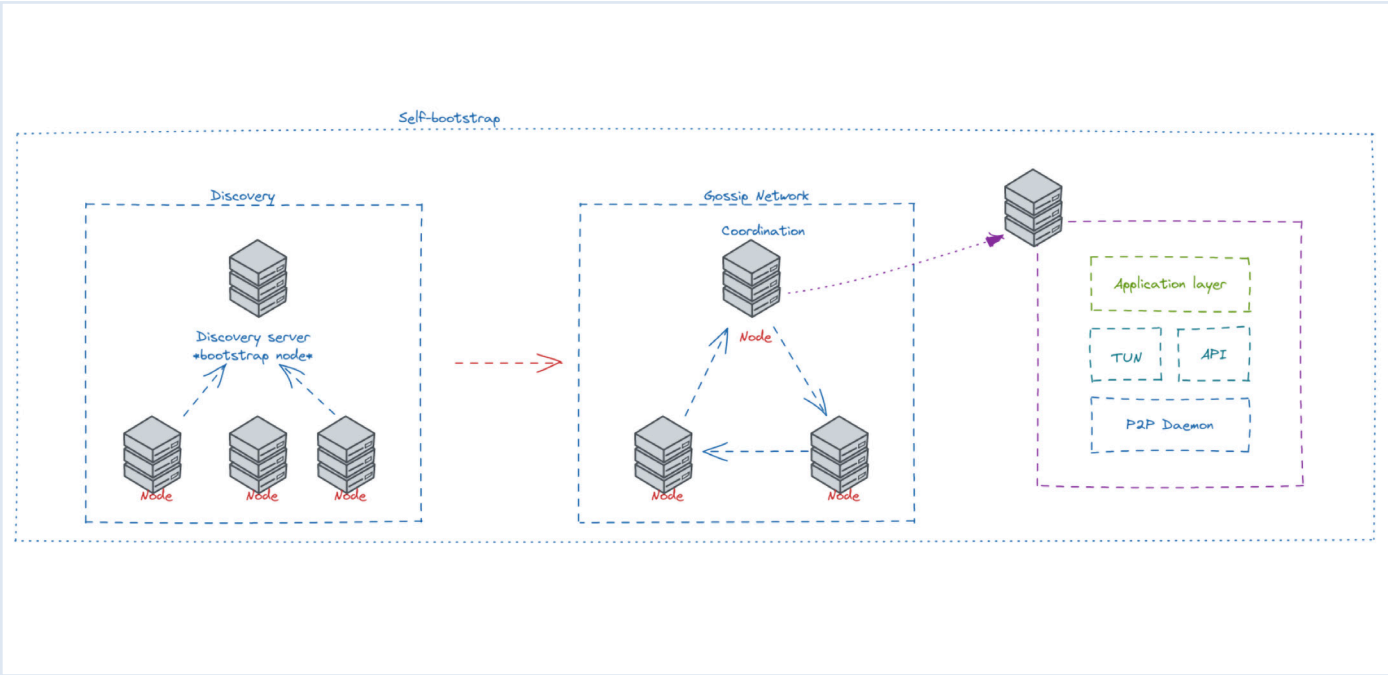
Kairos also provides AuroraBoot, a command-line interface that allows users to generate such artifacts locally instead, without requiring an existing Kubernetes infrastructure[17].

The QR code discussed above allows hosts to communicate via a peer-to-peer mechanism, which makes it an ideal solution for situations where outside connectivity is limited, and minimal local setup procedure is required. Overall, Kairos's bootstrap mechanism is a highly-effective and efficient solution for easing the installation process for non-technical users.

P2P configuration and coordination

To ensure normal operations even in locations where connectivity is not predictable or completely absent by design, devices at the edge shouldn't rely on the existence of a management plane, and if they do, it should be able to sit in the same network segment and enable local bootstrapping.

Kairos uses peer-to-peer technology[18] (P2P) to automatically coordinate and create Kubernetes clusters without the need for a central control management interface[19]. This allows the edge devices in the cluster to communicate and coordinate with each other, ensuring that the clusters are set up correctly and efficiently by assigning appropriate roles depending on the cluster architecture[20].



Kairos also enables defining local policies to enforce specific roles in a cluster infrastructure during the deployment and after, allowing a complete degree of flexibility that scales to the user's desired topologies.

Other implementations, such as Talos's Kubespan[21], use a discovery mechanism and Wireguard Virtual Private Network (VPN) to route traffic to the cluster, providing an automated approach to discover nodes.

Conversely, Kairos's configuration allows for the use of P2P features without the need for a control plane and a VPN. This can be particularly useful in scenarios where coordination is needed but traffic routing is not required. In such situations, the CNI configuration remains untouched and available for additional customization.

The Kairos self-coordinating mechanism allows operators to reorganize the cluster topology, for instance by adding nodes to the cluster after bootstrapping after deployment, or re-organizing roles. This way, the Kairos implementation makes the nodes completely autonomous.

FIDO device onboarding

FIDO2[22] is a set of standards that enable strong authentication using public key cryptography and user verification methods. FIDO2 standards include WebAuthn and CTAP, which allow users to authenticate to online services using devices such as smartphones, security keys, or biometric sensors. FIDO2 standards also provide a high level of security and privacy, as they do not rely on shared secrets or passwords that can be stolen or compromised.

Device onboarding compliant to FIDO2 standards has the advantage of simplifying and automating the onboarding process, while ensuring the identity and integrity of the device and the platform.

One example of a secure device onboarding compliant to FIDO2 standards is the FIDO Device Onboard (FDO) protocol, which was developed by the FIDO Alliance, an industry consortium that aims to address cybersecurity challenges and enable secure online experiences.

The FDO protocol enables devices to securely onboard any device management system using a zero-touch provisioning method. The FDO protocol leverages asymmetric public key cryptography and attestation techniques to establish trust between the device and the platform, without requiring any user intervention or pre-shared keys.

Kairos uses FDO and the Trusted Platform Module (TPM) to allow only trusted devices at silicon level to receive shares of the disk encryption keys. TPM is a specialized hardware component that provides secure storage and cryptographic functions for enhancing the security of a computing device.

TPM-based cryptographically verified challenges allow only a trusted device to use the Kairos Key Management Server (KMS) for storing auto-generated passphrases.[23] The Kairos encryption framework is an extensible system that provides an additional layer of security, allowing FIDO2-enabled devices to act as a gate for the KMS. It leverages the discrete TPM chip, as well as firmware-based TPM solutions such as Intel® Platform Trust Technology (Intel® PTT). The TPM is used to identify a machine and as well provide hardware-protected encryption services.

Capability 2: Verifying the software stack

In this section, we will be discussing how different components of the SENA architecture can address the key requirements we introduced relating to provisioning an edge Kubernetes verified software stack, leveraging SBOM, SLSA and other components.

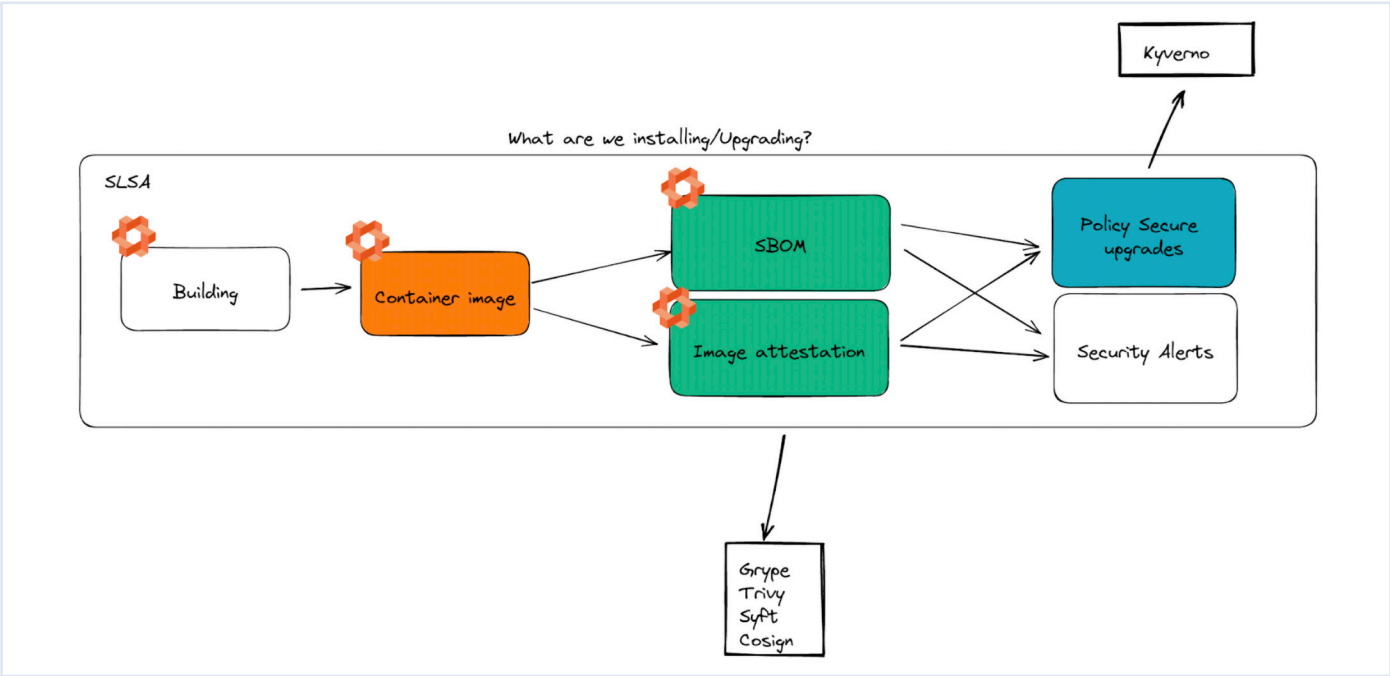
Access to the Open Container Initiative (OCI) ecosystem

SENA recommends leveraging the Open Container Initiative (OCI) ecosystem, allowing for the reuse of existing tools, such as vulnerability scanners, SBOM generators, or policy enforcement software.

By adopting OCI as a common architecture to deliver the operating system, the bootable OS can be packaged as a simple container image that can seamlessly run on any workstation with the container engine of choice.

This makes it easy to iteratively develop, test, debug, and troubleshoot changes to the OS before deploying to production. Additionally, OCI images integrate smoothly with the SLSA framework, enabling tamper prevention, improved integrity, and enhanced security of packages and infrastructure.

Traditionally, systems are analyzed or built using specific tooling that are tied to package managers. For example, Gentoo has its own security advisory database that offers integrated checks with the package manager[24]. Similarly, SUSE employs zypper [25] [26] and RedHat uses yum / dnf [27].



Kairos is an Open Container Initiative (OCI) image-based meta-distribution, and as such has no "opinion" on the OS.

A variety of tools can be used in tandem to generate security reports that can generalize OS targets or, for example, enforce upgrade policies; grype[28] (former anchore-engine[29]), trivy[30], and clair[31], are just a few that can all be used to identify vulnerabilities in container images. Tools like kyverno[32] can be used to enforce stricter policies during upgrades by verifying image attestations.

Integrating with SLSA and SBOM

The SLSA framework defines five levels[33] of abstraction that describe the different types of software artifacts and their relationships within the supply chain.

Satisfying at least level 3 guarantees an extra resistance to specific threats and ensures the auditability of the source and the integrity of the provenance. Kairos achieves level 3: container images are signed with cosign and SBOM lists are available[34], allowing for transparent tracking of component versions, security advisory, and verified attestation of images. Finally, images are built in Github Actions environments.

Cosign and Kyverno

Cosign is an open-source project that provides a solution for container image signing and verification. It uses a decentralized trust model and relies on cryptographic signatures to ensure the authenticity and integrity of container images. Cosign is currently used by Kairos to sign release artifacts, as such attestation and integrity checks can be performed with it. However, verification policies can be enforced in a Kubernetes cluster with kyverno using cosign.

Kyverno is an open-source policy engine designed to automate security and compliance for Kubernetes clusters. It allows users to define and enforce policies for various aspects of the cluster, such as pod security, network policies, and resource quotas. Kyverno's policies are written in simple and flexible YAML syntax and can be applied to the entire cluster or specific namespaces.

The SLSA framework enables users to leverage policy engines and maintain the chain integrity, as such kyverno can be used with Kairos during upgrades, enabling verification of the authenticity and integrity of container images[35] actively protecting from the compromise of container registries or leaking of credentials[36].

Capability 3: Securing runtime and upgrades

In this section, we will be discussing how different components of the SENA architecture can address the key requirements we introduced relating to operating the edge runtime securely.

As discussed previously, the layer between the application and the edge hardware it runs on is particularly vulnerable to attacks.

To mitigate these risks, it is essential to implement measures that ensure the application stack and data are encrypted and protected from tampering, both at-rest and in transit.

Hardware-enabled policy enforcement is key here and can further enhance the security of the edge runtime, adhering to confidential computing standards[37].

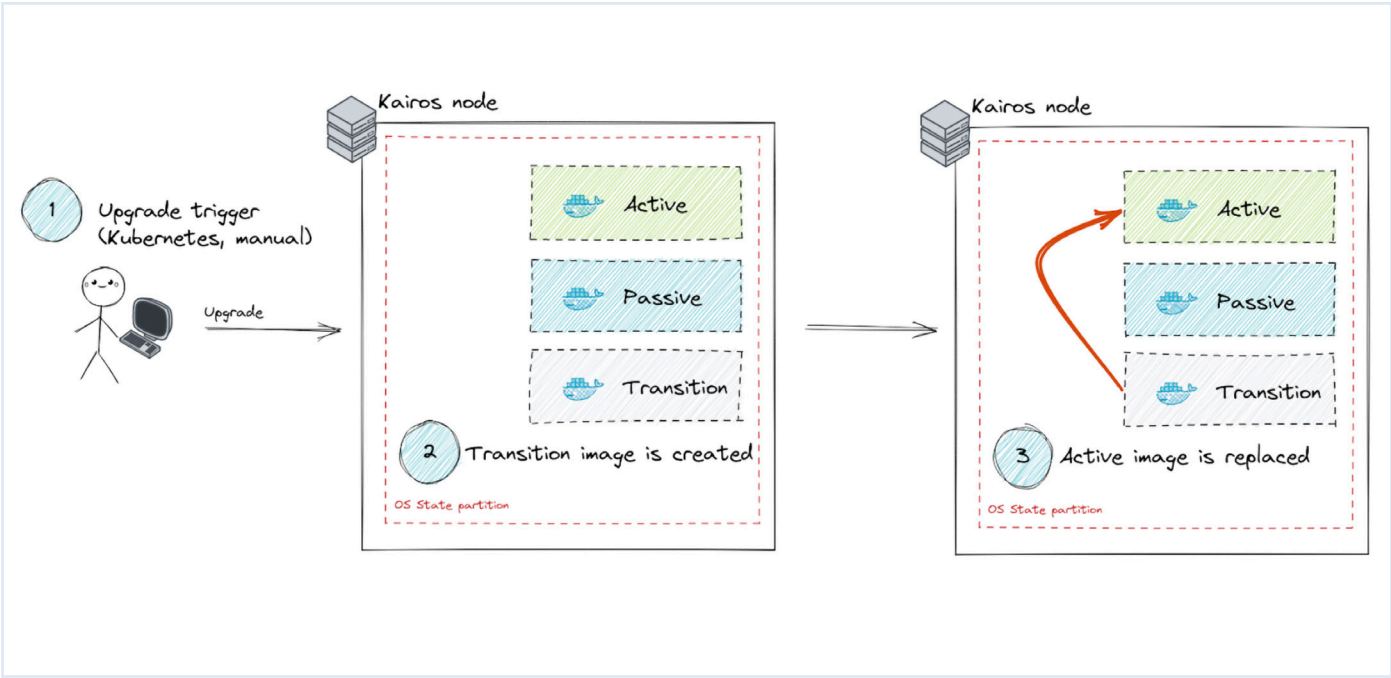
Flexible immutability with Kairos

Several immutable open-source OSes are available: CoreOS, Project Atomic, Ubuntu Core, Flatcar Container Linux, Talos and others.

However, these are all based on specific Linux distributions. Kairos is Linux distribution-agnostic. It can make any distribution immutable, and provides a suite of tools to automate and prepare deployment assets by using Kubernetes.

Kairos is also container based. The OS it outputs is a simple container image that can be inspected with standard tools in the ecosystem. As such it allows the distribution of OS upgrades using a container registry.

Images can be run locally with the user's container engine of choice. Upgrades are handled in an A/B fashion and can be completely orchestrated via Kubernetes as a standard deployment model, also addressing the need for zero downtime.[38]



A/B upgrades are atomic operations that can be applied manually or via Kubernetes. The edge device will create a transition image that will be swapped for the active system, and the active system becomes passive. Automated fallback and boot assessment are in place to automatically boot from the fallback system.

Typical Linux operating systems use a chainloader such as GRUB to boot the system, however GRUB can add security risk to the boot process, extending the attack surface.

We can eliminate this by removing the chainloader altogether in favor of the Unified Kernel Image (UKI)[39,40]. UKI is a single Extensible Firmware Interface (EFI) binary containing the kernel, initrd, kernel cmdline parameters and secure boot signature.

The Unified Extensible Firmware Interface (UEFI) firmware recognizes the single EFI binary as a bootable application. The UKI is an emerging model that allows a single EFI file to load an entire Linux operating system with additional benefits:

- One file to measure, sign, and verify, simplifying maintenance and management.
- True immutability, a single, signed file, that cannot be tampered with.
- Reduced attack surface.

However, it has few drawbacks as well:

- UEFI support is not-homogeneous, and EFI files are usually expected to be smaller than an entire OS image.
- Adding boot loaders might enlarge the surface exposed to attacks.

The Kairos team is actively working on a confidential based variant that uses UKI as its base[41], with a view towards easing out static measurement of components and management of upgrades at scale.

Confidential Computing

As Kubernetes is the orchestrator for containers, the choice of runtime can have a significant impact on security and isolation.

- Standard or default container runtimes, like containerd and CRI-O[42], offer very little isolation from other processes in the system — basically just namespace separation.
- Kata Containers deploy workloads to lightweight virtual machines, and therefore offer a little more isolation from other processes on the system.
- Amazon uses Firecracker in its Lambda service to launch MicroVirtual Machines (MicroVMs)[43].

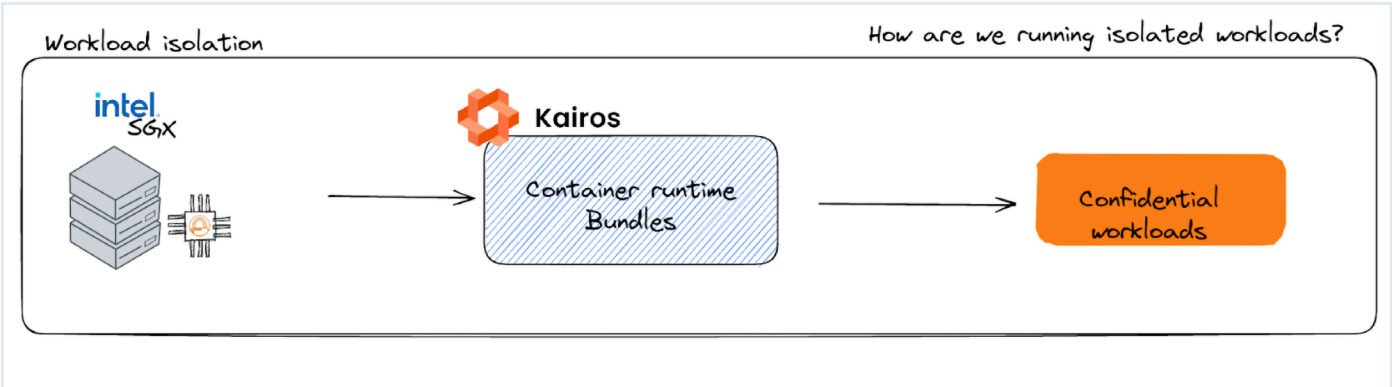
To ensure optimal performance and protection against potential vulnerabilities, it is crucial to choose a runtime that aligns with an organization's security requirements and workload needs.

Gramine Shielded Containers

For applications that need to run containers in Trusted Execution Environments (TEEs), the Gramine[44] project enables unmodified applications to run in secure enclaves, using technology such as Intel® Software Guard Extensions (Intel® SGX).

Gramine Shielded Containers enable unmodified containers to execute applications inside an Intel® SGX enclave, providing complete privacy from the OS, root users, and basic electrical probes of memory or system buses on the motherboard.

Technologies like Intel® SGX enable trusted execution environments to provide robust layers of security for containerized workloads. Kairos has integration directly with Gramine Shielded Containers (GSC), allowing it to transparently run confidential workloads[45] on supported hardware platforms like those with Intel SGX.

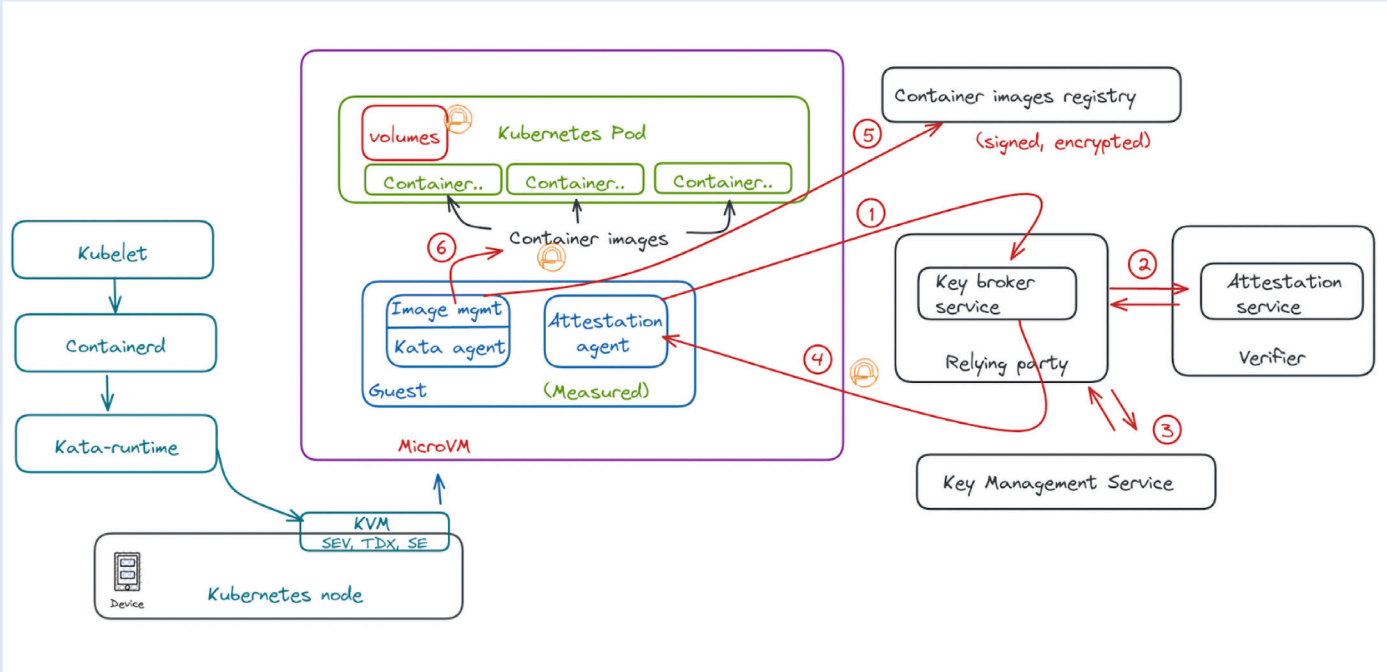


VM-based TEEs

Confidential containers can be implemented with MicroVMs and per-process isolation with Intel SGX and Gramine. VM-based TEE[46]s (e.g. Intel® Trust Domain Extension, Intel ®TDX, or AMD SEV, IBM SE) can be used to build a confidential container software architecture.

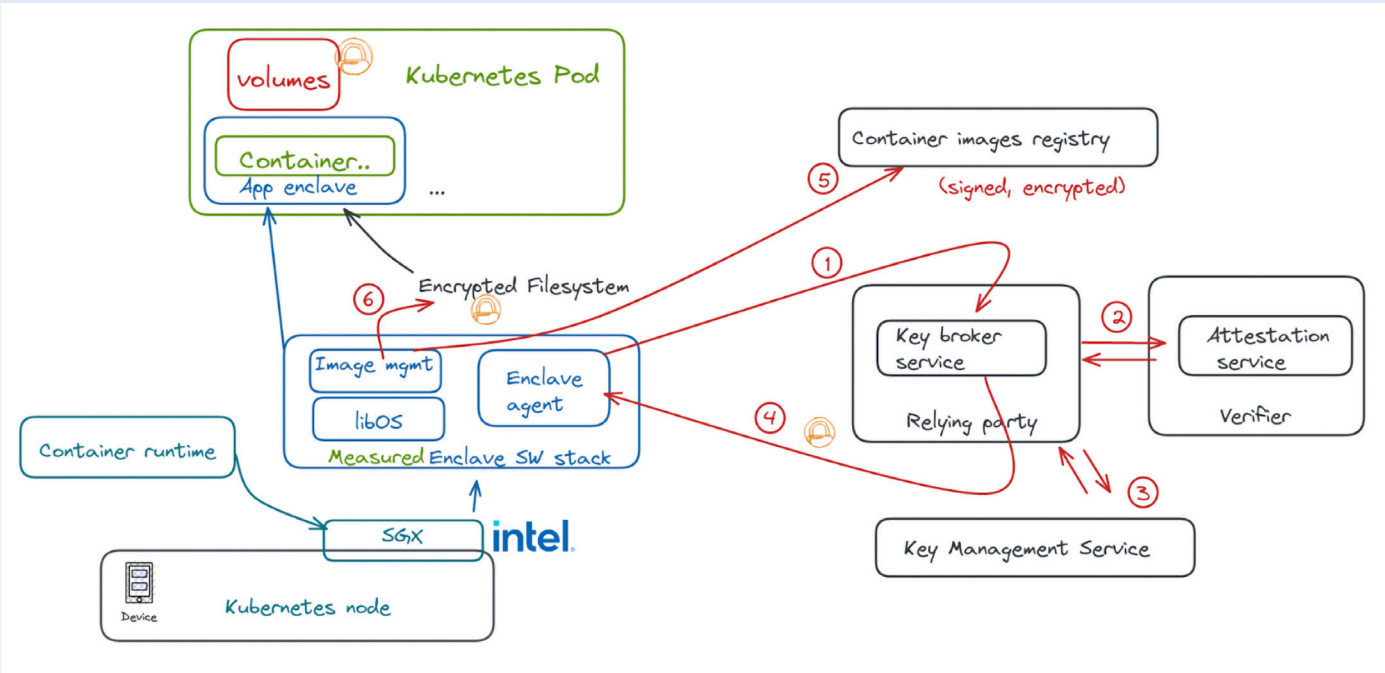
The workflow for deploying a Kubernetes pod with VM-based TEEs in can be summarized in three steps:

1. **Confidential workload preparation:** the user builds the container image, signs/encrypts the container image, and pushes it to the image registry.
2. **Deploying the confidential workload in Kubernetes:** the user deploys the workload and Kubernetes schedules the workload to target hosts having the required capability to run confidential containers.
3. **Execution flow:** The confidential container runtime on the host starts the VM TEE (the enclave). The enclave (agent) performs remote attestation (steps 1-2 in the diagram) and gets the keys required to verify/decrypt the container images (steps 3-4 in the diagram). The enclave downloads the container images (step 5) and verifies/decrypts the container images (step 6). Finally, the enclave starts the container workload inside a MicroVM.



Per-process TEEs

Confidential workloads can be also enabled per-process, leveraging for instance Intel SGX enclaves. Process-based TEEs rely on user applications built with SDKs that allow them to operate in a process enclave. Popular SDKs are Gramine[47], Ego[48] or Edgelessrt[49]. When deploying a Kubernetes pod with a process-based trusted execution environment (TEE), the workflow differs from the virtual machine (VM)-based TEE approach, as the last steps involve interaction between two enclave processes.



The workload needs to be prepared as in VM-based TEEs. The key difference is during the confidential workload execution flow: the confidential container's runtime on the host starts the enclave agent instead of running in a MicroVM.

- The enclave agent performs remote attestation (steps 1-2 in the diagram above) and retrieves the keys required to verify/decrypt the container images (steps 3-4).
- The enclave then downloads the container images (step 5) and verifies, decrypts, and writes them to a local encrypted filesystem (step 6).
- The runtime starts the app enclave, which reads the container bundle from the encrypted filesystem. The secure use of the encrypted filesystem is facilitated by a key exchange between the agent and app enclaves using either sealing or local attestation.

It's worth noting that the process-based TEE workflow is almost identical to the VM-based TEE workflow. However, with VM-based TEEs and Intel SGX, the host operating system runs outside the Trusted Compute Base[50] (TCB) compared to process-based TEEs.

In addition, the process-based TEE requires additional software components (confidential SDKs) to be part of the application, which is not necessary for VM-based TEEs.

To overcome such limitations, Gramine Shielded Containers automatically provide such capabilities transparently to the user, allowing them to run applications without explicit use of the SDKs. This allows users to run unmodified containers confidentially, without rewriting the application logic.

Conversely, process-based TEEs do not require a separate VM and Confidential computing-aware hypervisor[51].

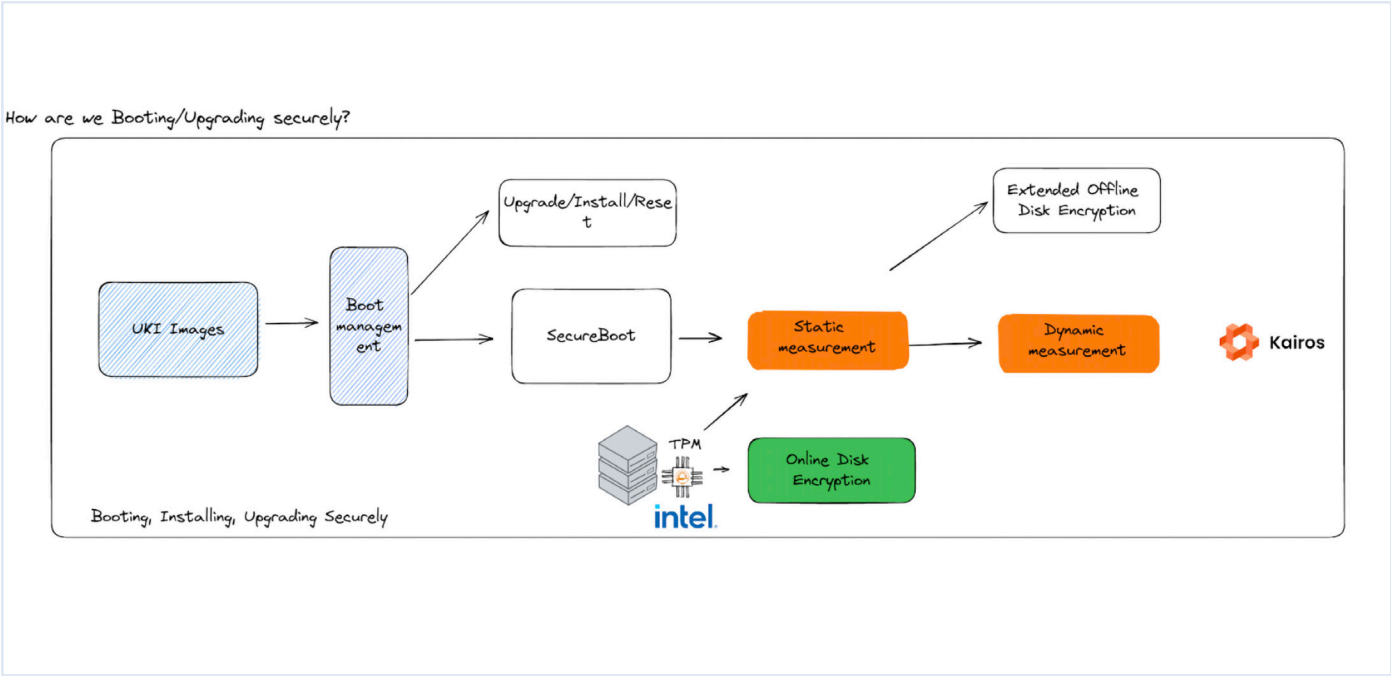
Securing the boot process

Measured boot and verified boot are industry terms for processes used for ensuring a secure boot process.

Measured boot is the process of measuring each code block of the boot chain, obtaining a cryptographic hash of the code block, and then storing that hash securely — typically by extending a PCR in a TPM. The resulting hash can then be used to determine the integrity of the boot chain. This can be performed statically (Static Measured boot) or Dynamically (Dynamic measured boot)

Verified boot is a little different and is used to ensure the code blocks come from a trusted source, which is verified by digital signature.

Secure boot is a mechanism defined by the UEFI, created by the industry. It reflects a combination of measured and verified boot processes to help ensure the integrity of the boot process and help eliminate attacks such as rootkits and bootkits by signing boot artifacts and verifying those during boot. When combined with a hardware root-of-trust, such as Intel® Boot Guard, the chain-of-trust becomes extremely strong.



Booting a secure system begins at power-on, with each code block being verified, prior to execution. Ideally the boot chain begins with a hardware root-of-trust, created in silicon, which verifies the first stage bootloader typically found in firmware.

On a PC, this first stage is implemented by UEFI firmware, which replaces the traditional Basic Input/Output System (BIOS) firmware used in older computers.

UEFI then finds the second stage bootloader (typically on disk) , verifies it, and passes execution to it. This continues until the OS has taken over and the system is completely booted.

In typical systems, the boot chain-of-trust ends with either loading the second-stage bootloader, as with most Linux systems, or with verifying the kernel, as is the case with Microsoft Windows.

There are Linux solutions that continue the chain through the second-stage bootloader and to the Linux kernel. UKI based systems are able to continue the chain-of-trust through the loading of the complete Linux system, all files and applications in the root file system, and kernel parameters.

The advantage of secure boot is that it provides an additional layer of security against malware and other malicious software. It helps prevent attackers from tampering with the boot process and installing malicious software that can steal sensitive data, damage the system, or give the attacker control over the system.

When implementing secure boot, users should avoid:

- Using the well-known Microsoft shim[52]
- Using unsigned or self-signed bootloaders or kernels, which can bypass the UEFI firmware verification process and potentially compromise the security of the system, as attacks at this level of the boot process are becoming more frequent[53]
- Third-party software that modifies the boot process, such as virtualization software or boot managers. These can introduce security risks if not properly configured and signed.

Signed artifacts are the baseline for guaranteeing a trusted boot sequence, and enable Measured boot by preventing the lower part of the stack from being compromised.

Measured boot

The boot sequence of a system is a critical part of its security, as it determines which software components are loaded and executed. Any malicious modification or compromise of the boot sequence can have severe consequences for the system's functionality and confidentiality. Therefore, it is important to verify that the boot sequence has not been tampered with and that it matches the expected configuration.

As mentioned above, measured boot is a security mechanism that measures each component of the boot sequence and stores the measurements in a Trusted Platform Module (TPM). This process can be used to verify the system has not been tampered with or compromised by malicious actors, and helps to prevent the execution of unauthorized code. A typical use of these measurements is the "sealing" of secrets against them, such as disk encryption keys.

During the boot sequence, each of these cryptographic hashes are hashed against existing values in the TPM's Platform Configuration Register (PCR), special memory slots that can only be written by the TPM. This process is called extending the PCR.

The resulting hash is a unique value which can only be replicated if all hashes extended to a particular PCR are identical. By comparing the PCR values with known good values, one can determine whether the system has booted in a trustworthy state or not.

Moreover, one can use the TPM to seal cryptographic keys or credentials to specific PCR values, such that they can only be accessed if the system boots in a certain configuration. This can prevent unauthorized access or modification of sensitive data or resources.

Static measured boot has several advantages over other methods of verifying system integrity, such as:

- Providing a strong root of trust based on hardware rather than software.
- Covering the entire boot sequence from firmware to kernel, rather than relying on a single point of verification.
- Allowing for remote attestation of system state by external entities or services.
- Enabling fine-grained access control and policy enforcement based on system configuration.

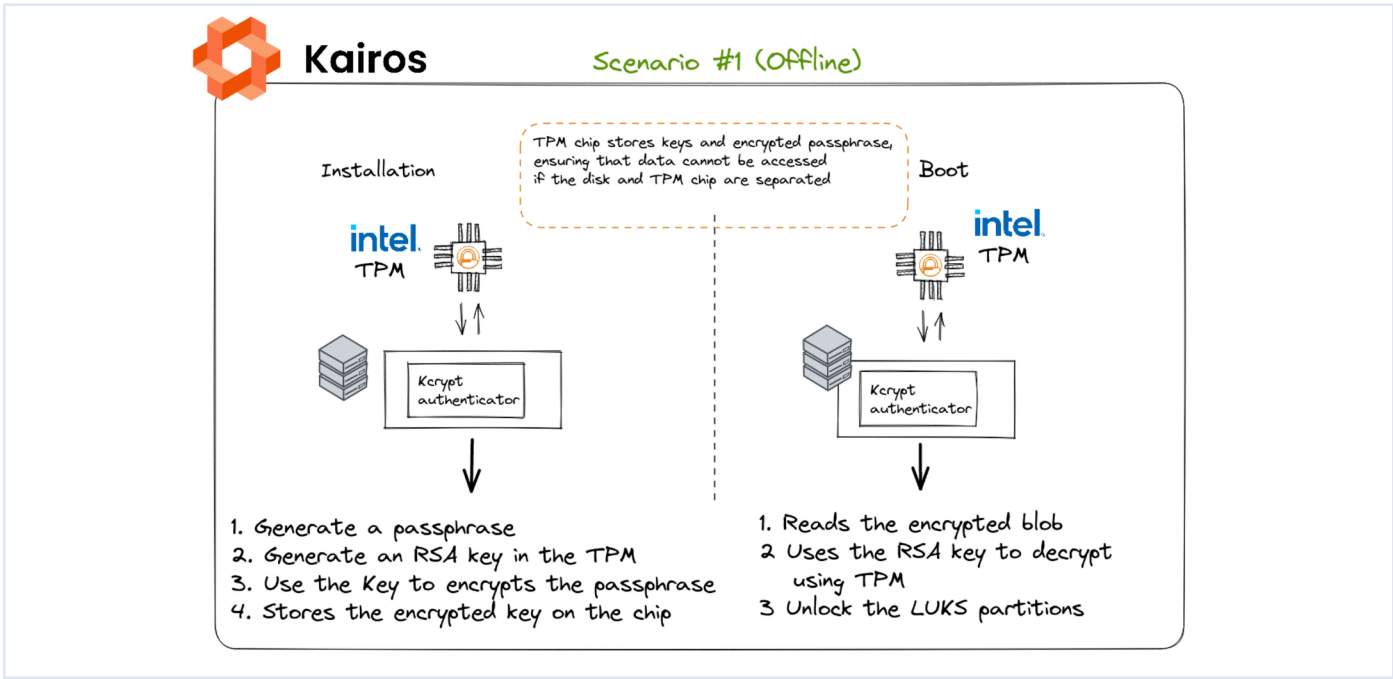
However, static measured boot also faces some challenges and limitations, such as:

- Requiring hardware support and compatibility from both the system and the TPM.
- Depending on the availability and accuracy of known good values for each component of the boot sequence.
- Not accounting for dynamic changes or updates to the system after booting.
- The booting and upgrade process becomes brittle, as the new version of the OS needs pre-computed hashes (so for instance, a bug could brick a system for good).
- Potentially introducing performance overhead or complexity to the boot process.

Static measured boot is a key requirement for modern secure infrastructure, as it guarantees that a device boots only if its configuration hasn't been tampered with. However, it introduces new maintenance challenges that have to be addressed in order to implement them safely at scale.

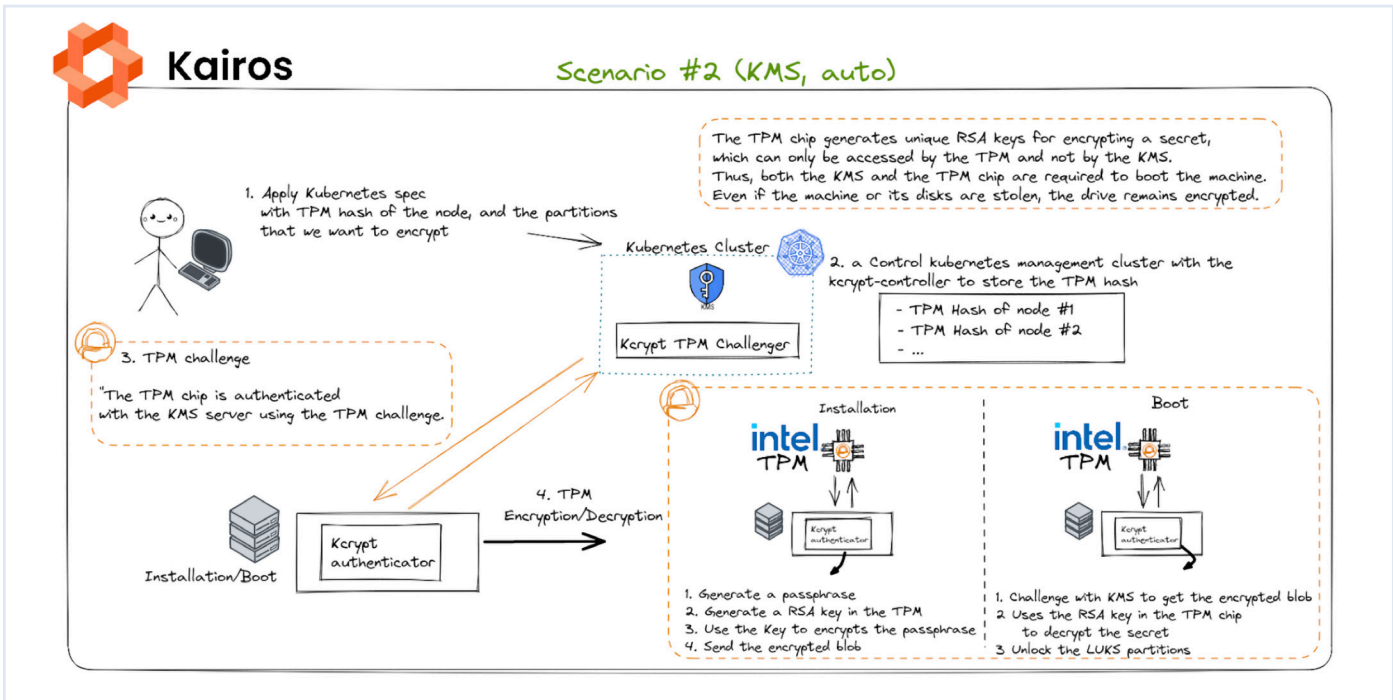
Locking disk decryption and boot using PCR values

The secure boot and static measured boot can be used in conjunction with disk encryption to unlock an encrypted disk or partition only if the device is verified against its TPM. This prevents alteration of the devices by physical factors (such as malicious USB drives attempting to modify the bootable components of a system) and secures data at rest.



TPM enables operators to seal a value in the PCR registries that are used to boot, and by storing a hash of the share can also prevent a system from booting if modified from its original form when being bootstrapped.

The encryption of data partitions can be linked to a remote management interface, enabling machines to encrypt data at rest using keys that are not stored locally, not even as shares in the TPM chip.



Kairos[54] enables nodes to create a secure passphrase that is encrypted using the local TPM chip. The encrypted shares are then transmitted to the Key Management Server (KMS), ensuring that only the local edge node can decrypt them.

To achieve this, the local edge node must first perform a hardware-level handshake using the TPM chip for identification. The operator then applies Kubernetes manifests, specifying which devices are expected to encrypt their drives and partitions.

During installation, the node generates a random share that is used to encrypt the partitions with Linux Unified Key Setup (LUKS). This share is sent back to the KMS and stored as an encrypted blob.

To generate the encrypted blob, the TPM Key Derivation Function (KDF) is used to create unique, private keys that remain only on the chip. These private seeds cannot leave the hardware platform and can only be accessed by the TPM chip. Only edge devices that are authorized can decrypt the contents of the shares stored in the KMS.

Dynamic measurement for attesting integrity beyond boot

Dynamic measurement is a process that involves measuring and attesting the integrity of a system at any point of time during system operation.

A DRTM is a Dynamic Root of Trust for Measurement. The DRTM uses special CPU instructions, such as Intel® TXT or AMD SKINIT, to create a secure execution environment and measure a new component of code. The DRTM extends the measurements to PCRs that are separate from those used by the Static Root of Trust for Measurement (SRTM).

Dynamic measurement can complement static measured boot by accounting for dynamic changes or updates to the system after booting. Dynamic measurement can be used in complement with runtime attestation to provide full OS verification and finer-grained (or application-specific) attestation evidence to external entities or services.

Keylime[55] is an open source project[56] that allows remote boot attestation and runtime integrity measurement solution for cloud and IoT devices. It uses TPM technology to monitor remote nodes and verify their trustworthiness, it verifies the boot state[57] and the runtime state of the system[58], and can protect and provision encrypted payloads[59].

Keylime uses the silicon PCR to run remote attestation to verify the correctness of the system, and also verify its runtime integrity with the Linux Integrity Measurement Architecture (IMA)[60]. IMA is a component of the Linux kernel's integrity subsystem that provides a means of detecting whether files have been accidentally or maliciously altered or executed, both remotely or locally.

The process can be roughly summarized by the following steps:

- **Collect:** measure a file before it is accessed and calculate its hash value.
- **Store:** add the measurement to a kernel-resident list and, if a hardware TPM is present, extend the IMA PCR with the measurement value.
- **Attest:** use the TPM (if present) to sign the IMA PCR value, to allow a remote validation of the measurement list.
- **Appraise:** enforce local validation of a measurement against a "good" value stored in an extended attribute of the file.
- **Protect:** protect a file's security extended attributes (including appraisal hash) against offline attack.
- **Audit:** audit the file hashes and report any discrepancies.

As such, Keylime is providing a full, secure implementation of root of trust verification.

Remote attestation

Remote attestation is a crucial capability of native confidential workloads. It allows a client to verify that an application is indeed running inside an enclave in an up-to-date TEE with the expected initial state. This process is often used to establish attested TLS (ra-tls) connections to an enclave.

To enable remote attestation, software can be built using confidential SDKs such as Gramine, Ego[61] or Edgelessrt[62]. Applications built with such SDKs can then expose primitives for remote attestation checks. For example, MarbleRun[63] is a framework for deploying confidential applications in Kubernetes. It automatically runs remote attestation on the workload, enabling the assessment that the confidential workload is indeed running inside enclaves.

Attestation can be done locally between two processes, or remotely with a management plane running validation checks. The MarbleRun approach is comprehensive and scales well to the cloud as it extends to different SDKs and allows attestation of the whole workload by using a "Coordinator" instead of running attestations for every single workload in the cluster. That serves as a centralized service for remote attestation and provisioning of secrets to applications.[64]

Management at scale

In the previous sections we discussed concepts and architecture components that are not necessarily tied to the existence of a remote management plane or platform beyond Kubernetes itself.

However, we recommend using a consistent management component, particularly when deploying edge computing environments at scale. A management platform can help automate the secure lifecycle operations discussed above, but just as importantly can simplify vital operational tasks such as observability and user management that contribute to a healthy overall security posture.

Leveraging a management platform can:

- Help minimize human error across repeated tasks
- Significantly accelerate time-to-market for infrastructure and apps, including patches and upgrades
- Decrease time-to-remediation for problems
- Enforce governance and compliance.

It can also provide an additional level of control, unlocking numerous scenarios that would otherwise require human intervention, such as handling software failures and preventing malicious attempts to compromise device integrity.

In this section we will be elaborating on some of the key components and characteristics required for an edge management plane or platform such as Spectro Cloud's Palette Edge. Note that the availability of a management plane can generally be assumed, but should not be a prerequisite. Edge devices must be resilient to failures, able to self-coordinate and operate autonomously even when the control plane is either absent or out of reach.

Introducing Palette Edge

Palette Edge is a comprehensive Kubernetes management platform. It enables teams to manage the full software stack of Kubernetes-based edge deployments, including the OS and any additional integrations required. Palette Edge incorporates native capabilities to address any day 0 to day 2 operational requirement, based on a decentralized architecture for unlimited scale.

spectro cloud

Choice in OS, Kubernetes distributions and integrations

Flexibility and avoiding lock-in is a key SENA requirement. Similar to the edge locations themselves, when a management plane exists as part of the overall architecture, it should support multiple Kubernetes distributions and operating systems, ideally out-of-the-box, with minimal to zero configuration required.

In addition, for users that want to bring their own integrations, access to community projects and other commercial solutions should be not restricted.

Orchestration across the complete stack

Many of SENA's requirements target the elimination of configuration drift at the level of an individual node or device, because configuration drift risks downtime and vulnerabilities.

Similarly, one of the critical requirements of a management plane is to ensure consistency across multiple devices and clusters, and across the full software stack.

This can be achieved only by an orchestration engine capable of deploying all components of the edge stack — OS, K8s distribution and any additional software elements — but also maintaining their state in a declarative manner, with frequent reconciliation loops in place. A modern management plane should have native orchestration capabilities.

Repeatability

It is inefficient and risky for operators to design and deploy one-off configurations for edge Kubernetes clusters, or to imperatively update those configurations over time.

It's essential therefore for a management plane to become a repository for reusable blueprints of Kubernetes stacks for different use cases, which can be versioned over time. A one-to-many approach where users can deploy and manage these "profiles" can significantly reduce the operational effort and cost required, especially when managing hundreds or thousands of locations.

To ensure compatibility and successful deployment of these profiles, automated pre-deployment tests across all layers of the stack are also required.

Lifecycle operations

While many of the SENA controls we've seen so far focus on the initial deployment of physical edge hardware and the software stack, and the boot and runtime of the device, there's a world of risk to manage in the rest of the edge lifecycle.

A complete set of natively integrated day 0 to day 2 capabilities is essential for effective management of edge locations. Operations teams need ready access to:

- Run security scans against their cluster software stack
- User management controls to ensure minimal privileges
- Backup and restore for the clusters and user data

Observability

Of particular note in daily lifecycle operations is observability.

Operations teams need clear dashboards to manage their fleet of edge locations, with the ability to monitor health at scale with live status updates and advanced filtering and tagging.

These updates provide real-time information of all devices, enabling users to quickly identify any issues that may arise and take proactive actions. It also allows users to track the performance of individual devices, helping them to identify patterns and trends that can inform future decision-making.

- Monitoring can help identify potential issues before they become critical, allowing for proactive intervention and remediation. Proactive actions could be used in tandem, for example by disabling access to confidential data on the device.
- Logging provides a detailed record of system events, which can be used for troubleshooting, forensic analysis, and compliance reporting.
- Alerting enables quick response to important events or incidents, helping to minimize downtime and mitigate risk.

Integration with common tools

We've already seen some of the advantages of leveraging common tools and frameworks, such as using OCI registries to manage OS container images.

Similarly, it is essential for a management platform to integrate with existing operational workflows and tools.

The platform must support common frameworks such as GitOps, Infrastructure as Code (IaC), and CI/CD tooling, to enable scaling to thousands of locations while maximizing management capabilities.

Furthermore, the platform should integrate effortlessly with an organization's ITSM (IT Service Management) and other external tools.

Zero-trust and granular RBAC

The zero-trust model is a security concept that assumes that all users, devices, services and networks are untrusted and should not be granted access to resources until they are authenticated and authorized.

The zero-trust model requires continuous verification of identity, device posture, and network conditions before access is granted to resources.

In the context of Kubernetes edge devices, a management plane using the zero-trust model would eliminate the notion of trust in the system, reducing the attack surface and increasing the overall security posture of the Kubernetes environment[65]. As opposed to using credential-based or certificate-based authentication paradigms, a real zero-trust model should be based on token-based authentication with the addition of an external Identity Provider (IDP) such as Okta, Azure AD or PingIdentity.

A zero-trust model management control plane used in tandem with granular Role-Based Access Control (RBAC) provides improved visibility into the Kubernetes environment. All requests are authenticated and authorized before being executed, which provides a comprehensive view of all requests made to the system, allowing for better monitoring and auditing of system activity.

Remote management

On-site support for edge devices can be costly or simply impossible to deliver. Organizations need the ability to remotely manage and control nodes at the edge.

There are many "in-band" management solutions that allow operations teams to remotely manage devices, but these rely on software agents running on the edge system. If the agent, or operating system, becomes unresponsive, then the device is no longer manageable, and a technician must typically be sent to reboot.

Furthermore, in-band solutions are unable to perform remote resets and power cycles of machines, which are necessary to help minimize downtime and reduce the impact of hardware failures and reduce the need to send technicians onsite.

To achieve low-level access to the hardware, especially when the operating system is absent or in an unknown state, out-of-band device management solutions have been employed. One of the most common implementations is the Intelligent Platform Management Interface (IPMI). To control the IPMI interface, a Baseboard Management Controller (BMC), which is a device with its own microcontroller, is typically added to the system motherboard. The BMC either runs proprietary management software, or popular open-source BMC software like Redfish®[66] and OpenBMC[67].

Since BMC implementations can be complicated and expensive, they are rarely found on enterprise client devices (PCs and laptops) or IoT devices. To address this gap, Intel® Active Management Technology[68] (Intel® AMT) was created as part of Intel vPro® platforms for their client CPUs. Intel® AMT provides many of the same OOB management features, such as device reset, power on, and remote KVM, that BMCs provide—but the capability is built into the CPU and chipset.

To make Intel® AMT easier to integrate for use in IoT solutions, Intel created the Open AMT Cloud Toolkit[69]. As an open source project, the Open AMT Cloud Toolkit makes it easier for IT departments and independent software vendors (ISVs) to adopt, integrate, and customize Out-of-band Management (OOB Management) solutions for Intel vPro® Platforms.

Decentralized architecture and self-healing

A foundational design principle that can impact the whole edge architecture is the level of dependency between the management plane and edge locations. As discussed previously, edge locations have to always maintain a certain degree of autonomy.

While this is especially important for sites that are either air-gapped or have intermittent connectivity, it's also best practice for clusters to depend as little as possible on the central management plane. For example, when scaling to hundreds or thousands of sites, bandwidth and performance can grind to a halt unless each cluster can do things like enforce policy locally.

In addition, a decentralized architecture can result in faster upgrades as they can happen simultaneously in parallel; conventional architectures with tight coupling between the management plane and the edge nodes must run upgrades serially, which is unworkable at scale.

The device nodes must be capable of operating without needing continuous connection to the management plane, allowing for the creation of independent networks. Therefore, the system should also be flexible and capable of readapting its scope as much as possible.

Nodes should have automatic failover policies in place and self-healing that can be enforced remotely to avoid any service disruption. A deployment architecture that is composed of several edge devices, or even a single one, should be capable of tolerating Byzantine failures[70,71] and be resilient to split-brains.

These systems should be configured for high availability of the infrastructure, ensuring maximum operational time. To achieve this, additional "smart agents" can be implemented to proactively reconfigure the system and reconcile it to a desired state. These agents can communicate with the management plane or operate autonomously.

The management plane must also support flexible deployment options itself, being able to support use cases where it needs to be hosted near the edge devices themselves. This is particularly important in situations where low-end devices need to be managed on-site.

Conclusion

Regardless of your industry, there is a use case for edge computing that could transform your business. And as rich, computationally intensive applications like AI/ML blossom, the edge is only going to become more important. But securely deploying and managing an edge infrastructure at scale is not easy.

Various complexities and problems must be considered, including hardware management, workload isolation, and protection against physical tampering. Managing the infrastructure is also critical, and processes, operations, and human interaction should be streamlined as much as possible to avoid security breaches and human errors.

In this paper we have defined a robust set of requirements and capabilities needed for any secure edge infrastructure, across three key areas: onboarding the device hardware, provisioning the software stack, and actually operating the device, in terms of boot, runtime and data storage.

We have gone further and scoped out ways that organizations can meet these requirements today, with a new framework called the Secure Edge-Native Architecture. SENA is not a product you can buy off the shelf: it is a guideline for architects and software engineers that provides a secure foundation and highly scalable design for meeting edge requirements.

Nor is SENA opinionated or restrictive about the technologies you choose to assemble. We have discussed several open-source (and commercial) software solutions that fill gaps in edge security and can help redefine security within the context of new hardware technologies.

SENA takes a holistic approach across the edge lifecycle and relies on close integration between hardware and software to deliver a more comprehensive security solution for the edge.

By embracing SENA, you can enhance your security posture, complying with well-defined standards, and start building your edge infrastructure on open-source building blocks backed by giants.

Next steps

This white paper is a first salvo in an ongoing conversation about edge security. We welcome any feedback and contribution to the discussion — you'll find the email addresses of all three authors on the cover of this document. If you'd like to take your first steps at working with the technologies explored in this paper, here's where you can start:

For Kairos, you can download code and images or contribute at kairos.io or https://github.com/kairos-io/kairos

For SENA-ready hardware, look for any Intel processor with SGX, visit ark.intel.com or see the list directly right here.

For the Palette Kubernetes management platform, you can get started for free, learn more or engage on the Spectro Cloud Slack community, all at spectrocloud.com.

About the authors

Ettore di Giacinto is Head of Open Source at Spectro Cloud. He's spent more than 15 years as a developer focused on contributing to and maintaining open source projects, including Cloud Foundry, openSUSE and Gentoo Linux. Most recently Ettore was staff software engineer at SUSE, as lead engineer and architect for the Rancher Elemental project. He is passionate about OSS for the edge and cloud.

Michael Millsap is a 22-year Intel veteran, with almost 30 years in the industry. Having spent most of his career in IoT system and security architecture, he now manages an edge server business in Intel's Network and Edge Group.

Bryan J Rodriguez is a Principal Engineer at Intel innovating new technologies and changing the future of edge computing. Primarily focused on scaled deployments with the least amount of operational cost and with intuitive adoption on autonomous functionality. With nearly 30 years of experience, Bryan brings a wide perspective with focused proficiency and execution in edge and cloud solutions space.

About Spectro Cloud

spectro cloud

Spectro Cloud uniquely enables organizations to deploy and manage Kubernetes in production, at scale. Its Palette enterprise Kubernetes management platform gives platform engineering and DevOps teams effortless control of the full Kubernetes lifecycle even across multiple clouds, data centers, bare metal and edge environments. Ops teams are empowered to support their developers with curated Kubernetes stacks and tools based on their specific needs, with granular governance and enterprise-grade security.

Co-founded in 2019 by CEO Tenry Fu, Vice President of Engineering Gautam Joshi and Chief Technology Officer Saad Malik, Spectro Cloud is backed by Stripes, Sierra Ventures, Boldstart Ventures, Westwave Capital, Alter Venture Partners, Firebolt Ventures, T-Mobile Ventures and TSG.

Spectro Cloud is a CNCF-certified Kubernetes service provider, a member of the Linux Foundation LF Edge project, and a member of the Confidential Computing Consortium.

For more information, visit https://www.spectrocloud.com or follow @spectrocloudinc on Twitter.

About Kairos

Kairos

Kairos is an open-source project which brings Edge, cloud, and bare metal lifecycle OS management into the same design principles with a unified cloud native API. With Kairos, users can build immutable, bootable Kubernetes and OS images for their edge devices as easily as writing a Dockerfile.

For more information, visit https://www.kairos.io or follow @Kairos_OSS.

About Intel

intel.

Founded in 1968, Intel's technology has been at the heart of computing breakthroughs. We are an industry leader, creating world-changing technology that enables global progress and enriches lives. We stand at the brink of several technology inflections—artificial intelligence (AI), 5G network transformation, and the rise of the intelligent edge—that together will shape the future of technology. Silicon and software drive these inflections, and Intel is at the heart of it all.

For more information, visit intel.com.

Glossary

Air-gapped: A security measure that physically isolates a system or network from external networks or untrusted sources to prevent unauthorized access.

AMD SEV - AMD Secure Encrypted Virtualization: A security feature in AMD CPUs that provides hardware-based memory encryption for protecting data of virtual machines from unauthorized access.

Byzantine failures: Byzantine failures refer to a type of failure in distributed computing systems where nodes exhibit arbitrary or malicious behavior, such as sending contradictory or misleading information, in an attempt to disrupt the correct functioning of the system.

CNI - Container Networking Interface: A standard API for networking in container runtimes, allowing containers to communicate with each other and with the host system.

Confidential Computing: A security concept that protects sensitive data and computations by isolating them in secure enclaves or trusted execution environments, preventing unauthorized access.

CVE - Common Vulnerabilities and Exposures: A publicly disclosed vulnerability in software or hardware that may pose a security risk, usually identified by a unique identifier and managed by a vulnerability database.

DRTM - Dynamic root of trust Measurement: A security feature in modern computer systems that establishes a trusted measurement of the system's integrity during runtime or boot time, typically used for securing virtualized environments.

EFI - Extensible Firmware Interface: A standard firmware interface for modern computers that replaces the traditional BIOS (Basic Input/Output System), providing a more advanced and flexible way to boot and configure the system.

FDO - FIDO Device Onboarding: A mechanism that enables secure device provisioning and ownership, establishing trust between devices and cloud services.

FIDO - Fast Identity Online: A set of open authentication standards that enable passwordless and multifactor authentication for online services, providing enhanced security and user experience.

GitOps: A software development and deployment approach that uses Git as the single source of truth for managing and automating the configuration, deployment, and monitoring of applications in a declarative and version-controlled manner.

Gramine: Gramine is a lightweight library operating system, formerly called Graphene, designed to run a single Linux application in an isolated environment. When combined with Intel® SGX, Gramine enables an unmodified application to make use of SGX and run securely in a secure enclave.

Gramine Shielded Containers: GSC uses Gramine, which is a library OS that enables unmodified applications to run in secure enclaves, and enables the deployment of complete Docker containers that are then protected by running within Intel® SGX secure enclaves.

Hardware-based encryption: A security feature that uses dedicated hardware components to encrypt and decrypt data, providing faster and more secure encryption than software-based methods.

IBM SE - IBM Secure Execution: A security feature in IBM Z mainframes that provides a protected environment for executing sensitive workloads, isolating them from the rest of the system for enhanced security.

IDP - Identity Provider: a service that can create and manage digital identities to allow for access for various systems.

Immutable OS: An operating system that is designed to be read-only and cannot be modified, providing enhanced security and protection against unauthorized changes or tampering.

Initrd - Initial RAM Disk: A temporary file system used during the boot process of a Linux-based operating system to load necessary drivers and modules for accessing the root file system.

Intel® Active Management Technology: Intel® AMT is a hardware and firmware solution that enables remote, out-of-band management of devices. This technology ensures that devices can be managed, even when the operating system is unresponsive, or when the system is powered off.

Intel® PTT - Intel® Platform Trust Technology is technology built into Intel platforms that offers the capabilities of discrete TPM 2.0. Intel® PTT provides a trusted environment for secure storage of cryptographic keys and other sensitive data, as well as the functions necessary to support Secure Boot.

Intel® SGX - Intel Software Guard Extensions: A security feature in Intel CPUs that provides hardware-based memory encryption for creating secure enclaves within applications to protect sensitive data and computations.

Intel® TDX - Intel's Trust Domain Extensions (TDX) safeguard the privacy of sensitive guest virtual machines (VMs) by implementing isolation of guest register state and encryption of guest memory, effectively defending against attacks from the host and physical sources.

IPMI - Intelligent Platform Management Interface: A standardized interface for out-of-band management of computer systems, allowing remote monitoring and management of hardware components, such as servers or network devices.

ISV - Independent Software Vendor: A company that develops and sells software products that are designed to run on third-party hardware or software platforms.

ITSM - IT Service Management: A set of practices and frameworks for managing and delivering IT services in a structured and organized manner, aligning IT with business needs and requirements.

KDF - Key Derivation Function: A cryptographic function that derives one or more secret keys from a single secret value, typically used to strengthen the security of passwords or other sensitive data.

Kernel (O.S.) - The core component of an operating system that manages system resources, provides system services, and acts as an intermediary between hardware and software components.

Kernel Commandline - A configuration parameter passed to the kernel during boot, specifying various settings and parameters for the Linux kernel.

KMS - Key Management Service: A centralized system for generating, storing, and managing cryptographic keys used for encrypting and decrypting data in a secure and controlled manner.

Kubernetes: An open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Kyverno, grype, trivy, syft: These are open-source security tools used in the container ecosystem for vulnerability scanning, image analysis, and policy enforcement to ensure the security of containerized applications.

Linux IMA - Linux Integrity Measurement Architecture: A security framework in Linux-based operating systems that provides integrity measurement and appraisal of system files and processes to detect unauthorized changes or tampering.

LUKS - Linux Unified Key Setup: A disk encryption specification for Linux-based systems that provides a standard format for encrypting and decrypting data on disk partitions.

MicroVM, Firecracker, Kata, Containerd, Cri-O: These are container runtimes or virtualization technologies used in cloud computing environments for lightweight and secure deployment and management of containerized applications.

OCI - Open Container Initiative: An industry standard for container image format and runtime, providing interoperability and portability of containerized applications across different platforms and runtimes.

Open AMT Cloud Toolkit - Open AMT Cloud Toolkit provides an open-source, modular, microservices implementation for supporting the integration of Intel® Active Management Technology. With Open AMT, a device management solution can quickly and seamlessly integrate Intel® AMT functionality.

OS - Operating System: A software system that manages computer hardware and software resources, and provides common services for computer programs.

P2P - Peer-to-Peer: A distributed networking architecture where computers or devices communicate directly with each other, without the need for a central server.

PCR - Platform Configuration Register: A register in a Trusted Platform Module (TPM) that stores measurements of system components to establish a trusted baseline for verifying system integrity.

RA-TLS - Remote attestation TLS : A security network protocol that leverages remote attestation between the two parties to establish a secure connection.

Redfish: A standard API for remote management of computer systems and data center infrastructure, providing a modern and scalable way to monitor and control hardware components.

SBOM - Software Bill of Materials: A list of all the components and dependencies used in a software application, providing transparency and accountability in the software supply chain.

SDK - Software Development Kit: A set of tools, libraries, and documentation that developers use to build software applications for specific platforms or frameworks.

Secure Enclave: A hardware-based security feature in modern CPUs that provides a separate and isolated area for securely storing and processing sensitive data, such as encryption key or securing data in-use.

SLSA - Supply Chain Levels for Software Artifacts: A security framework that establishes a set of verifiable requirements for ensuring the integrity and security of software supply chains.

SRTM - Static root of trust measurement: A security feature in modern computer systems that establishes a trusted measurement of the system's integrity during boot time, typically used for securing the system's firmware and boot process.

TPM - Trusted Platform Module: A dedicated hardware chip that provides secure storage of cryptographic keys and other sensitive data, and performs secure cryptographic operations, such as encryption and decryption.

Trusted Execution Environment: A secure and isolated environment in a computer system that provides a protected area for executing sensitive computations for data in-use or storing confidential data.

UEFI - Unified Extensible Firmware Interface: An updated version of EFI (Extensible Firmware Interface), which is a standard firmware interface for modern computers, providing advanced boot and configuration capabilities.

UKI - Unified Kernel Image: A single EFI binary that contains bootable assets for a Linux distribution.

VPN - Virtual Private Network: A technology that extends a private network over a public network, providing secure remote access and communication between users or devices.

Endnotes

1 https://fidoalliance.org/
2 http://act-r.psy.cmu.uploads/2017/03/Halbruegge2016-CognitiveStrategies.pdf
3 https://slsa.dev/
4 https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
5 https://cloud.google.com/blog/products/application-development/google-introduces-slsa-framework
6 https://download.microsoft.com/download/3/c/9/3c922792-e6d8-460c-a073-44afea713b92/Microsoft%20Response%20to%20OMB%20Questions%20on%20Implementation%20Guidance%20for%20Federal%20Procurement%20of%20Secure%20Software.pdf
7 https://www.redhat.com/en/blog/SLSA-framework-measuring-supply-chain-security-maturity
8 https://docs.sigstore.dev/cosign/other_types/#sboms-software-bill-of-materials
9 https://www.bleepingcomputer.com/news/security/amazon-ecr-public-gallery-flaw-could-have-wiped-or-poisoned-any-image/
10 https://arstechnica.com/information-technology/2023/03/unkillable-uefi-malware-bypassing-secure-boot-enabled-by-unpatchable-windows-flaw/
11 Linux distributions typically are operated by package managers whose responsibility is to install, remove and upgrade software. Popular package managers for instance are dnf(Fedora/RedHat), zypper(openSUSE), or apt(Ubuntu).
12 Confidential Computing Consortium – Linux Foundation Project
13 https://github.com/OSInside/kiwi
14 https://github.com/containers/bootc
15 https://kairos.io/docs/installation/automated/
16 Such as command line interface, or an external service
17 https://kairos.io/docs/reference/auroraboot/
18 https://kairos.io/docs/installation/p2p/
19 For instance, a remote management platform, or help of external tools
20 For instance, High Availability setups.
21 https://www.talos.dev/v1.3/kubernetes-guides/network/kubespan/
22 https://fidoalliance.org/fido-alliance-creates-new-onboarding-standard-to-secure-internet-of-things-iot/
23 https://kairos.io/docs/advanced/partition_encryption/#scenario-automatically-generated-keys
24 Glsa-check can be used, advisory published here: https://security.gentoo.org/glsa
25 https://www.suse.com/support/kb/doc/?id=000016886
26 https://www.suse.com/security/cve/
27 https://access.redhat.com/documentation/it-it/red_hat_enterprise_linux/8/html/managing_and_monitoring_security_updates/identifying-security-updates_managing-and-monitoring-security-updates
28 https://github.com/anchore/grype
29 https://github.com/anchore/anchore-engine
30 https://github.com/aquasecurity/trivy
31 https://github.com/quay/clair
32 https://github.com/kyverno/kyverno
33 https://slsa.dev/spec/v0.1/levels
34 Kairos SBOM lists are generated by https://github.com/anchore/syft
35 https://kairos.io/docs/upgrade/kubernetes/#verify-images-attestation-during-upgrades
36 https://thehackernews.com/2022/12/serious-attacks-could-have-been-staged.html
37 https://confidentialcomputing.io/
38 https://kairos.io/docs/architecture/container/
39 https://github.com/uapi-group/specifications/blob/main/specs/unified_kernel_image.md
40 https://0pointer.net/blog/brave-new-trusted-boot-world.html
41 https://github.com/kairos-io/kairos/issues/347
42 CRI-O is a Kubernetes Container Runtime Interface for OCI runtimes
43 https://www.amazon.science/blog/how-awss-firecracker-virtual-machines-work
44 https://github.com/gramineproject/gramine
45 https://kairos.io/docs/advanced/coco/
46 A trusted execution environment (TEE) is an area of the system which is not accessible by other processes or by the OS.
47 https://gramine.readthedocs.io/en/stable/attestation.html
48 https://docs.edgeless.systems/ego/
49 https://github.com/edgelesssys/edgelessrt
50 Trusted Computer Base is the critical set of firmware, hardware and software components that can undermine a system's security in case of bugs or vulnerabilities.
51 https://github.com/confidential-containers/documentation/blob/main/architecture.md
52 https://github.com/rhboot/shim can be used in conjunction to load Microsoft CA certificates https://jfearn.fedorapeople.org/fdocs/en-US/Fedora_Draft_Documentation/0.1/html/UEFI_Secure_Boot_Guide/sect-UEFI_Secure_Boot_Guide-Implementation_of_UEFI_Secure_Boot-Shim.html
53 https://arstechnica.com/information-technology/2023/03/unkillable-uefi-malware-bypassing-secure-boot-enabled-by-unpatchable-windows-flaw/
54 https://kairos.io/docs/advanced/partition_encryption/
55 https://www.ll.mit.edu/news/keylime-provides-root-trust-scale
56 https://keylime.dev/
57 https://keylime.readthedocs.io/en/latest/user_guide/use_measured_boot.html
58 https://keylime.readthedocs.io/en/latest/user_guide/runtime_ima.html
59 https://keylime.readthedocs.io/en/latest/user_guide/secure_payload.html
60 https://linux-ima.sourceforge.net/
61 https://docs.edgeless.systems/ego/
62 https://github.com/edgelesssys/edgelessrt
63 https://github.com/edgelesssys/marblerun
64 https://gramine.readthedocs.io/en/stable/attestation.html#edgeless-marblerun
65 https://www.nist.gov/publications/zero-trust-architecture
66 https://www.dmtf.org/standards/redfish
67 https://www.openbmc.org/
68 https://www.intel.com/content/www/us/en/architecture-and-technology/intel-active-management-technology.html
69 https://github.com/open-amt-cloud-toolkit/open-amt-cloud-toolkit
70 https://pmg.csail.mit.edu/papers/osdi99.pdf
71 See Glossary

intel spectro cloud Kairos