
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

From Silicon to Service: Ensuring Confidentiality in Serverless GPU Cloud Functions

Democratizing Confidential Computing with a Lift-and-Shift Strategy Using Confidential Containers and GPUs Powered by Kata

Zvonko Kaiser NVIDIA

Recap: Confidential Containers with GPU and RAG LLMs

Confidential Containers for GPU Compute

- Road to Confidential Computing
- Kata Containers the Why – How – What
- GPU enablement stack
- Virtualization Reference Architecture
- Confidential Containers
- Confidential RAG LLMs

Recording:
https://www.youtube.com/watch?v=a3HzBmPuw5g&list=PLj6h78yzYM2N8nw1YcqqKveySH6_0VnI0&index=124

Expanding Use Cases & Role of the Working Groups

Confidential Containers Use Cases WG

KubeCon CloudNativeCon North America 2024

Members:

Cheng Jang Thye

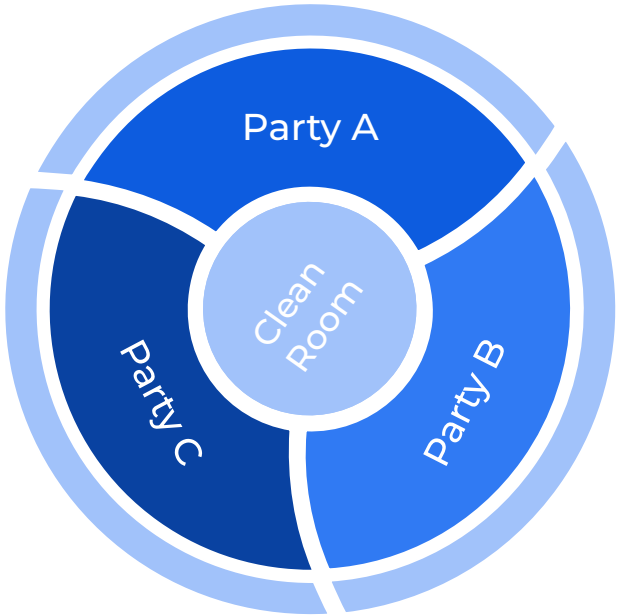
Chester Chen

Isaac Yang

Pradipta Banerjee

James Magowan

Zvonko Kaiser



Use Cases:

Federated Learning

- Data Clean Room
- Multi Party Computing

Trusted Pipeline (supply chain)

- SBOM, provenance
- CI/CD, repeatability

RAG LLMs, NIMs

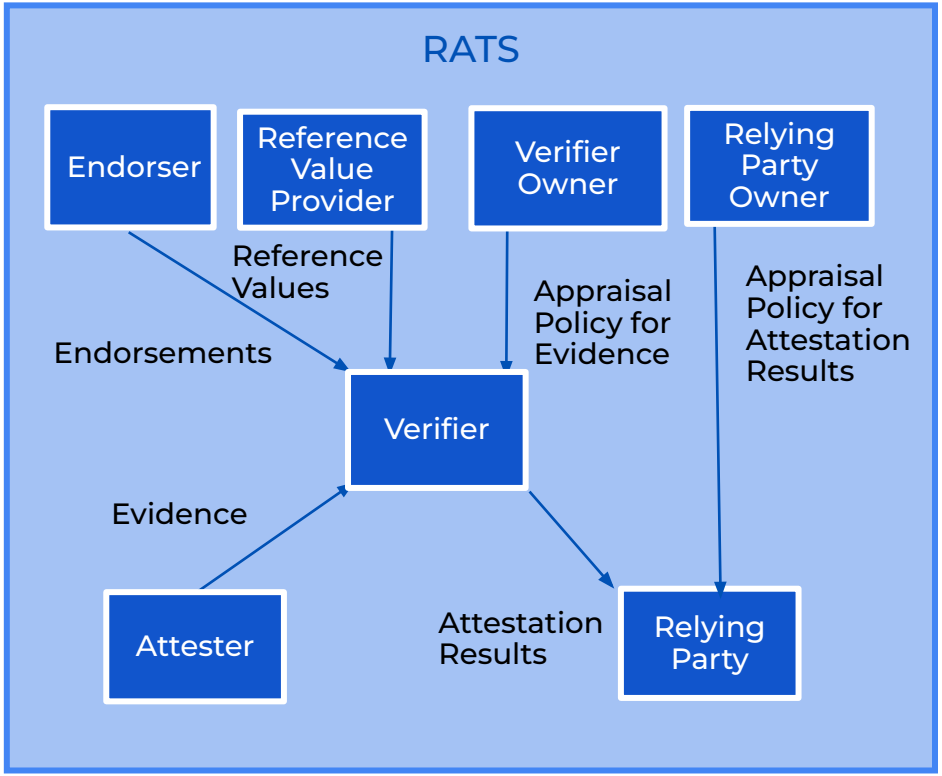
- Generative AI

https://docs.google.com/presentation/d/1Qgr9ef0mDmXTgn_t-dr0CqZ2wts0ecgN9t1mUT9GR6Y/edit#slide=id.p

Confidential Containers Trustee WG

Key New Aspects:

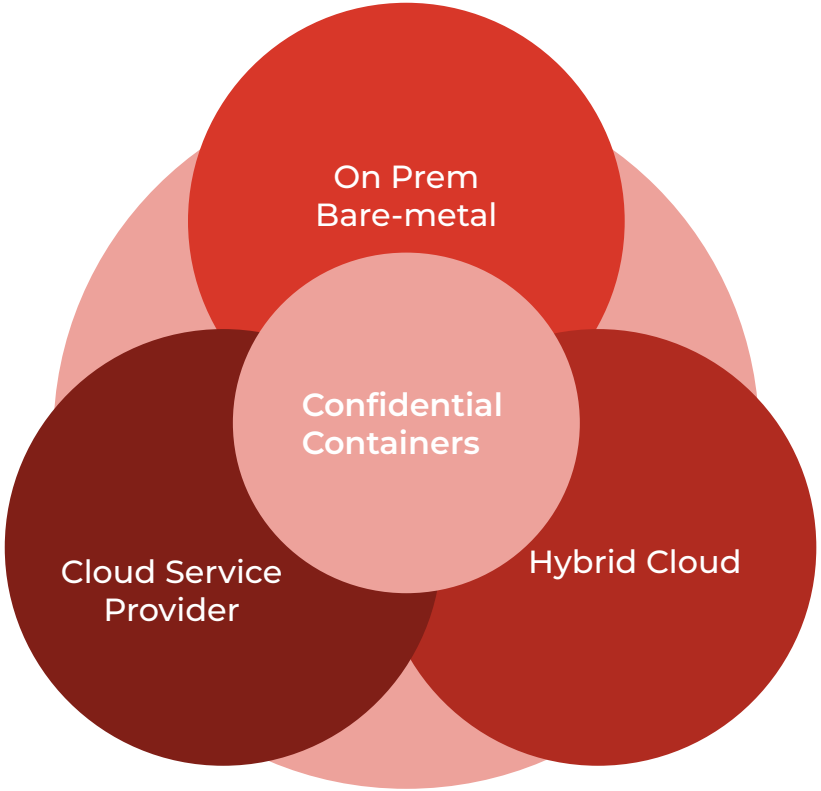
- Periodic attestation runs
- Multiple persona support
- Composite attestation support (TDISP Attestation timing attack)
- Runtime integrity
- Global state attestation
- Identity management



Democratizing Confidential Computing

Confidential Containers Lift-and-Shift

- Run on any HW (SNP, TDX, CCA, CoVE, SE, ...)
- Run on any infrastructure on-prem, CSP or hybrid
- Run on any model of deployment, serverless, managed Kubernetes

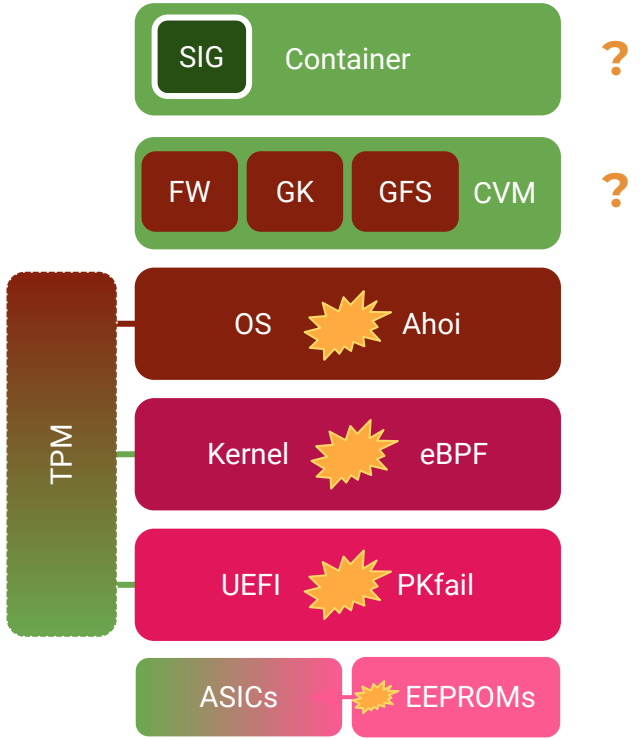


Full Stack Attestation

Threat Model – Do not trust the Host

Untrusted runtime:

- PKfail – Untrusted Platform Keys Undermine Secure Boot on UEFI Ecosystem
- Ahoi Attacks – Disrupting TEEs with Malicious Notifications
- eBPF – Who Watches the Watcher



eBPF and other tools are expected to ensure integrity, yet they also depend on the integrity of the system that they're monitoring.

Kurt Gödel's incompleteness theorem

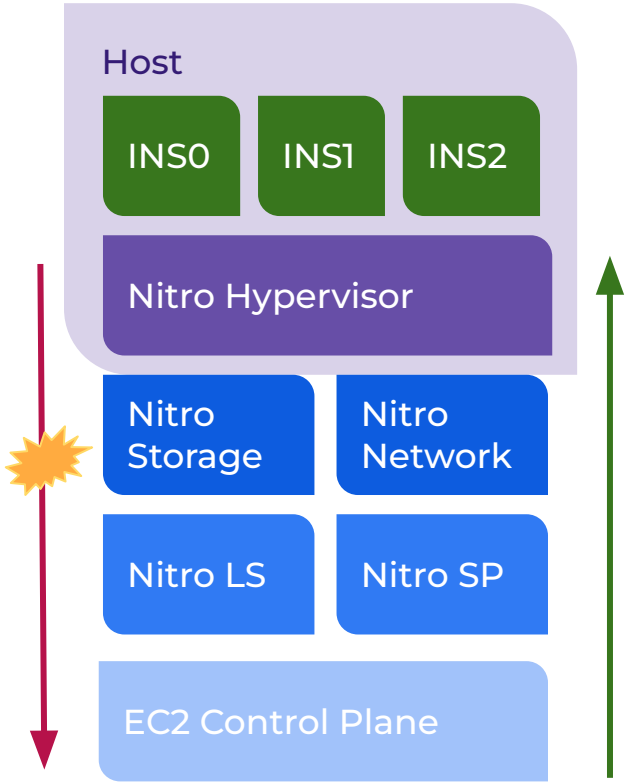
Gödel demonstrated that in any **sufficiently complex formal system**, there are propositions that **cannot be proven** true or false within that system; you need an external or higher-level system to verify them.

Similarly, in the context of securing a runtime, any attempt to prove the integrity of the runtime environment from *within* itself creates a **circular dependency**, as noted. This makes it impossible to establish absolute trust without relying on an "external" or higher level of verification.

Cloud Runtime Verification

AWS Nitro:

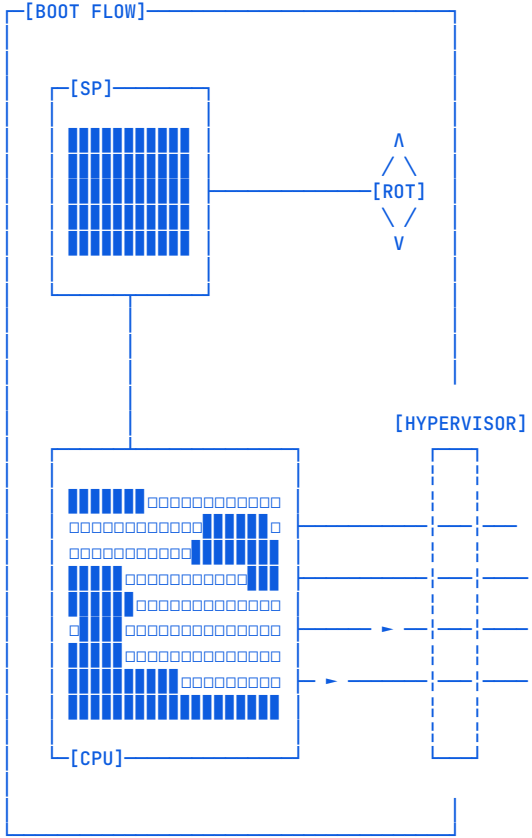
- **Dedicated Hardware & Offloading:** Nitro offloads networking, storage, and management to dedicated hardware, reducing the hypervisor footprint and isolating sensitive workloads.
- **Trusted Platform Module (TPM):** Provides hardware-based attestation and encryption capabilities.
- **Isolation & Performance:** Enhanced isolation for virtual machines, with high-performance support for sensitive data and confidential computing workloads (e.g. Nitro Enclaves).
- **Out-of-band Management:** Instances are managed without the hosts reach



On-Prem Runtime Verification

Oxide Computers:

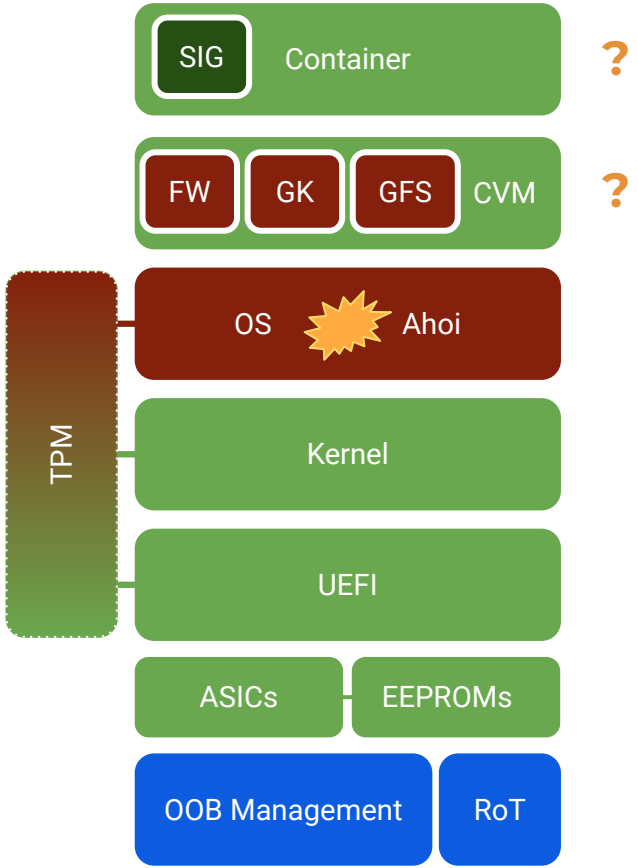
- **First Instruction Integrity:** Root of trust cryptographically validates that its own firmware is genuine and unmodified
- **Extending Trust Through Boot:** Processor in reset, measuring firmware and save them in RoT, release CPU only upon successful attestation
- **Extending Trust Between Devices:** RoT provisioned with a PK and certificate signed by Oxide
- **Trust Quorum:** Verifiable secret sharing
- **Secure Secret Storage:** A dedicated storage service that prevents storing secrets in RAM and enforces strict access controls to limit access and usage of sensitive information.



Threat Model – Do not trust the Host

Untrusted runtime:

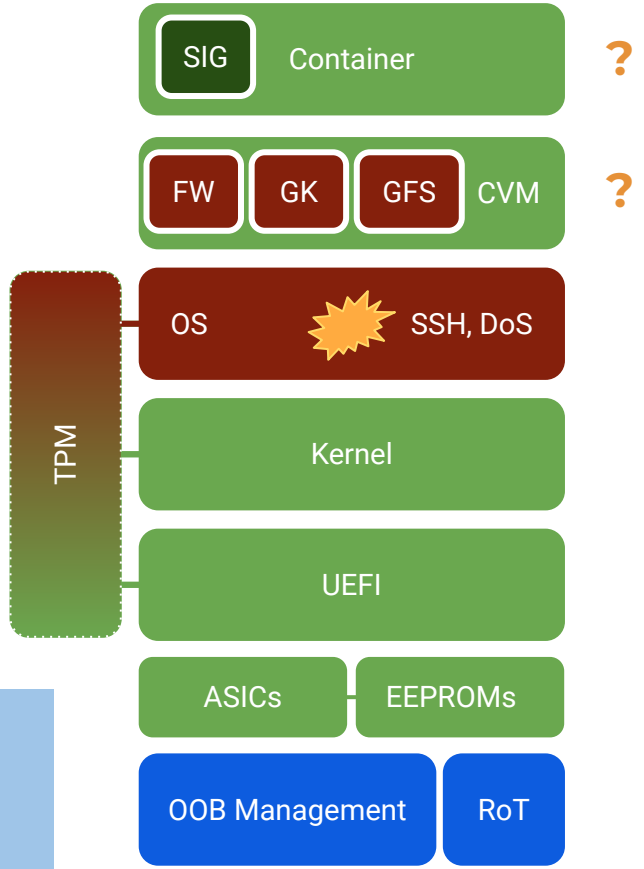
- PKfail – Untrusted Platform Keys Undermine Secure Boot on UEFI Ecosystem
- Ahoi Attacks – Disrupting TEEs with Malicious Notifications
- eBPF – Who Watches the Watcher



Threat Model – Do not trust the Host

Runtime integrity

- dm-verity, fs-verity
  - Merkle-tree, hash of hashes
- IMA/EVM
  - Signed files and metadata
- Immutable OS
  - RHCOS, Ubuntu Core, etc.
- Type-1 hypervisor
  - Do we even need an OS?

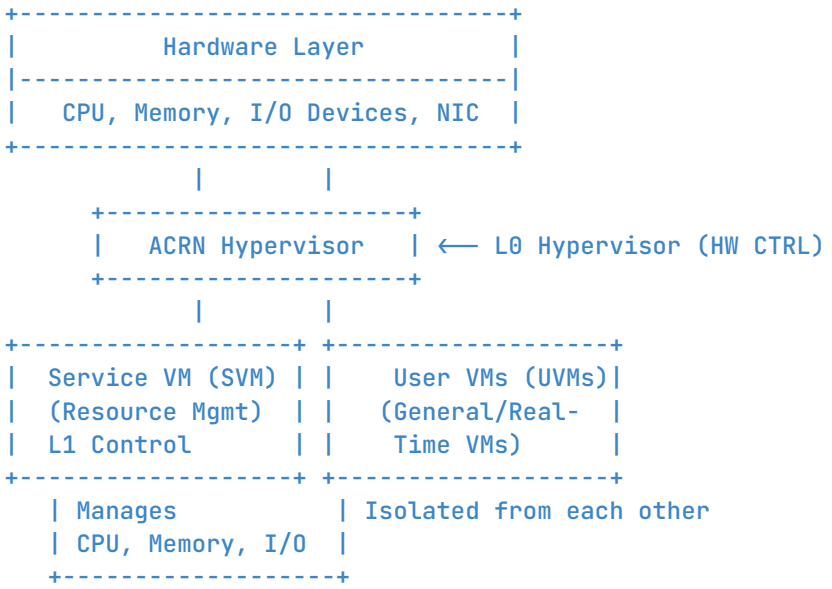


KVM + VMM(s) are sharing the very same kernel, like containers do, breakout can still create DoS on other VMs. Create a void for escapes.

secVM Model for Enhanced Cloud Virtualization

Remove nested virtualization

- **Simplified Architecture:** Flattens the virtualization hierarchy by running all VMs as L1 guests, bypassing nested layers. This reduces the complexity traditionally associated with multi-layered models.
- **Performance & Security Gains:** Lowers virtualization overhead, resulting in improved speed and responsiveness. The streamlined design also strengthens isolation, enhancing security in the cloud.
- **Reduced Resource Usage:** Minimizes resource demands by eliminating redundant layers, leading to cost savings and improved efficiency.

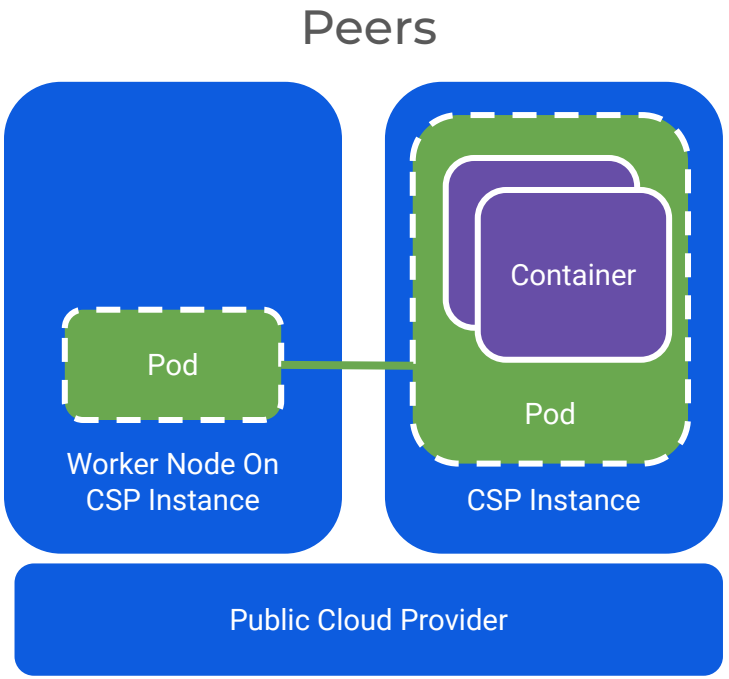


Free the turtles is a reference to the Turtles Project that introduced the concept of nested virtualization in the open-source virtualization technology - Kernel-based Virtual Machine

Confidential Containers: Peer Pods

Remote Hypervisor:

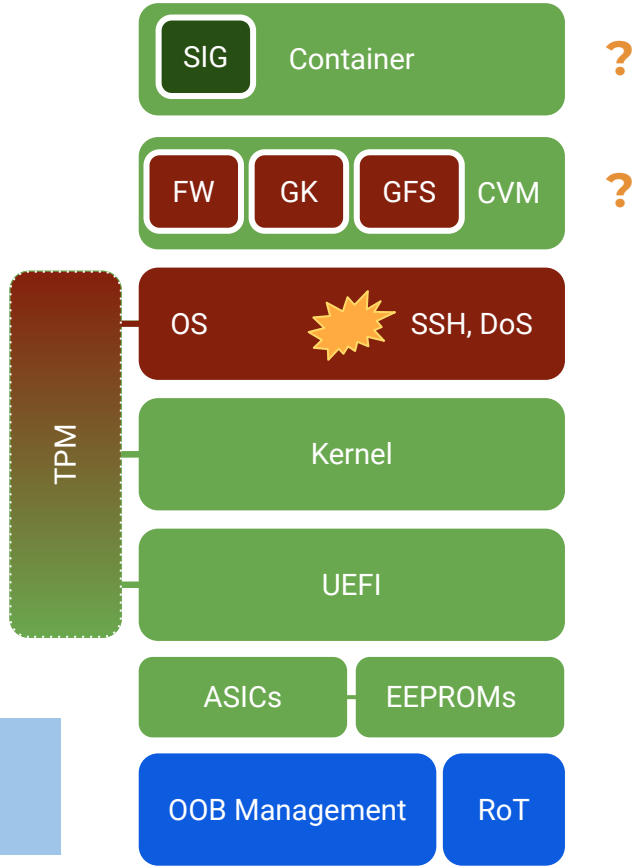
- **Confidential Containers:** Run without requiring bare-metal servers or nested virtualization
- **Protection:** Host and workload are protected by CSP, we can run privileged workloads
- **Transparent:** Pod is behaving the very same way as a traditional Pod
- **Hybrid Cloud:** Run a Pod on-prem via local VM and burst out to a CSP with Peer-Pods, lift-and-shift
- **Control-plane:** Clear separation from control-plane and workers



Threat Model – Do not trust the Guest

Guest Runtime integrity

- dm-verity, fs-verity
  - Merkle-tree
- IMA/EVM
  - Signed files and metadata
- Immutable OS
  - RHCOS, Ubuntu Core, etc.
- How to protect the runtime measurements?

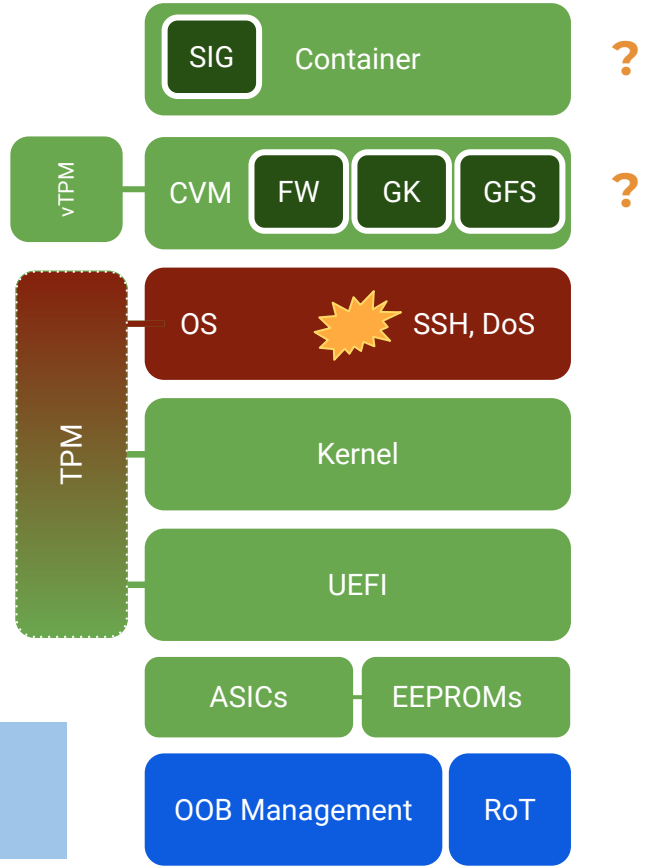


Runtime measurements can be protected via attestation report fields, TDX has RTMR, vTPMs?

Threat Model – Do not trust the Guest

We cannot pass-through a vTPM

- **Coconut-svsm, OpenHCL:** High privileged FW running in a special VM level
- **vTPM Stateless:** No state is saved or loaded
- **vTPM Manufacturing:** EK is bound to attestation of the CPU and linked to the cert-chain of the vendor
- **Storage, Ephemeral Keys**: All other keys can now be derived from the Endorsement Key

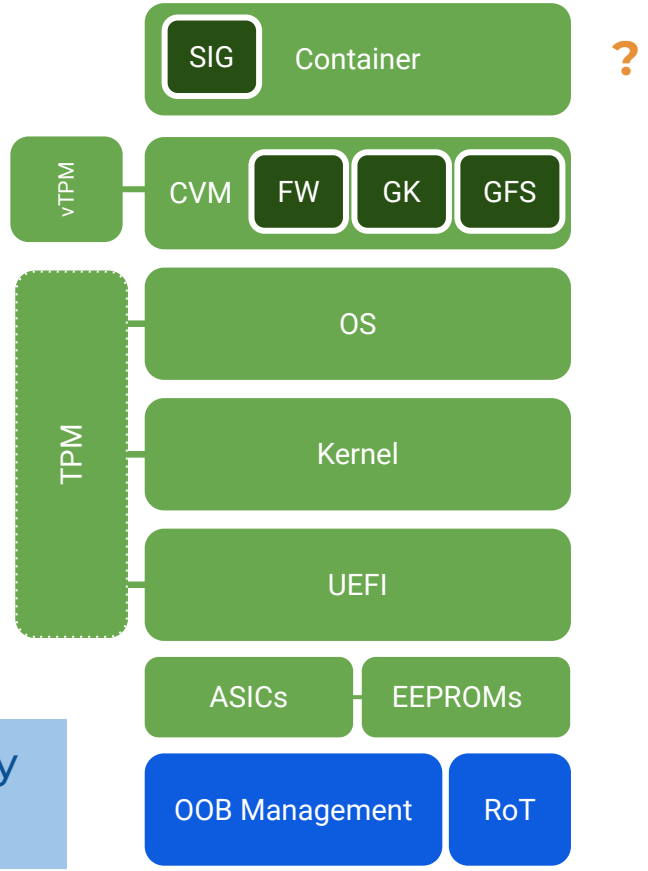


Is the TMP the right interface to protect runtime measurements? Upstream disagrees

Threat Model – Do not trust the Guest

Guest-VM just the deployment vehicle:

- **Guest-kernel hardening:** We do not need a full blown Kernel, only the parts for running the kata-agent
- **Distroless guest fs:** Package only the needed libs to run the kata-agent
- **Firmware hardening:** Disable not need parts
- **Container payload:** Protect the guest filesystem from executing container payloads via SELinux or AppArmor



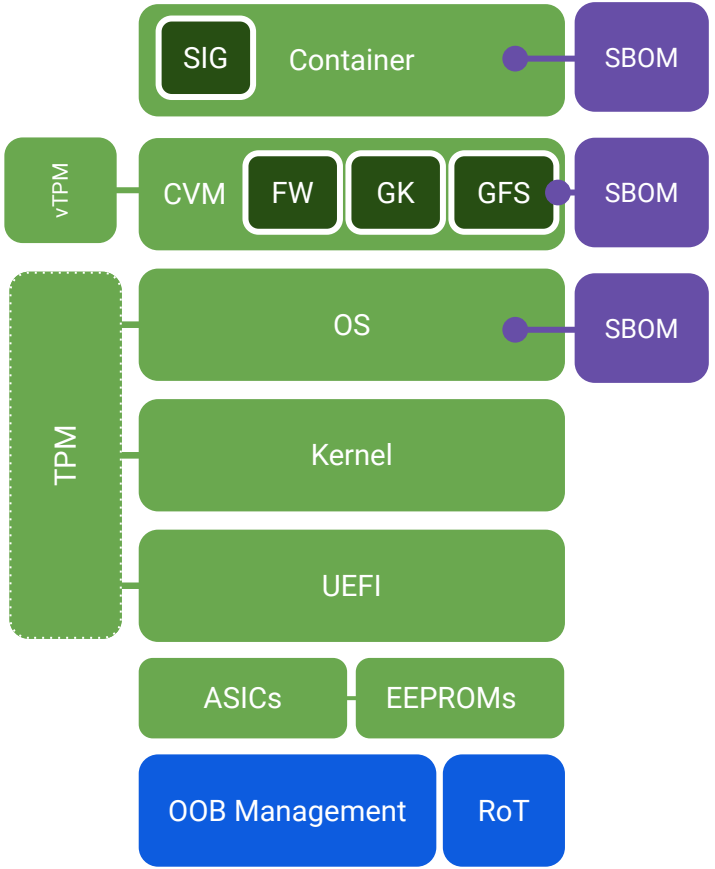
Create a void for the attacker without the possibility to enumerate anything or lateral movement

Supply Chain – SBOMs

What is running exactly in my artifacts:

- **SBOM:** React on CVEs, bugs or needed updates in the artifacts
- **File based SBOMs:** For all things distroless, we do not have a package manager or db.
- **Attestation:** Verify all signatures, SBOMs, certificates via remote attestation

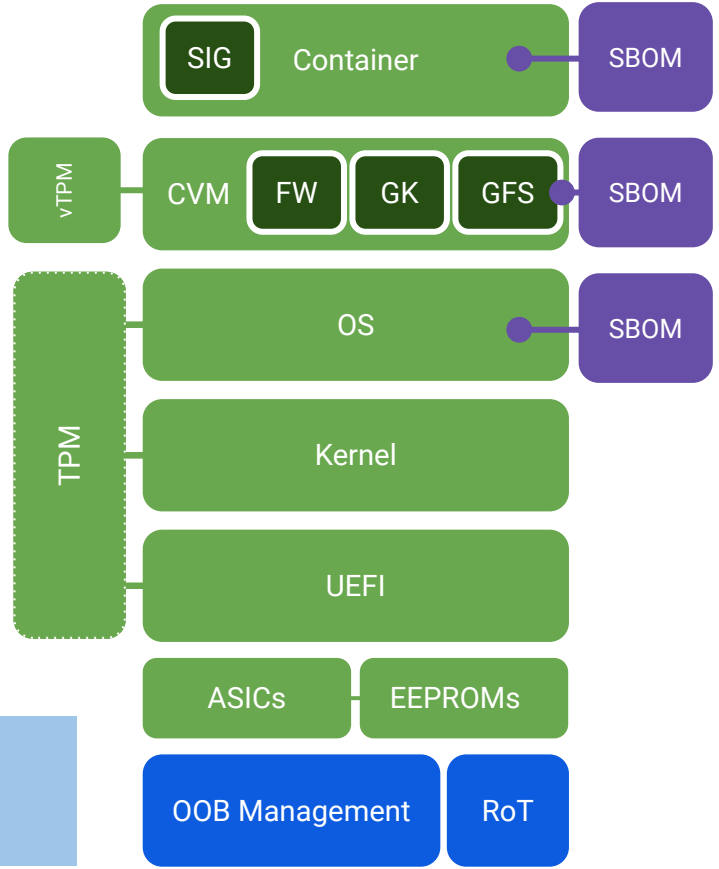
We need to build a chain of trust starting with RoT and going up to the Containers SBOM.



Remote Attestation – Trustee

Sign/Measure everything:

- **CSP Attestation:** They are attesting themselves, which maps back to the problem of the circular dependency
- **Trustee:** Run it on trusted tenant infrastructure, if needed broker to vendor RAS
- Attest everything



Measure and Attest Every Layer – Trust Only What You Verify

Cloud Functions – Serverless

Cloud Functions run on shared, abstracted infrastructure where the underlying layers are hidden but crucial to security.

- **Opaque Layers:** The underlying hardware, hypervisor, and OS are managed by the provider and shared across tenants.
- **Shared Resources:** Potential for data leakage and malicious interference across functions without visibility into lower layers.
- **Hidden Threats:** Vulnerabilities in hardware or firmware could impact your function without your knowledge.
- **Compromised Execution:** If runtimes or OS are unmeasured, they could be tampered with, impacting function integrity and data security.
- **Hidden Threats:** Vulnerabilities in hardware or firmware could impact your function without your knowledge.
- **Compromised Execution:** If runtimes or OS are unmeasured, they could be tampered with, impacting function integrity and data security.


KubeCon
CloudNativeCon
North America 2024
Q&A