Confidential Containers

Sensitive Data and Privacy in Cloud Native Environments

2024-06-19, stackconf 2024 Berlin

stackconf

Hi!

Magnus Kulke

SWE @Microsoft, Azure Core Linux

magnuskulke@microsoft.com

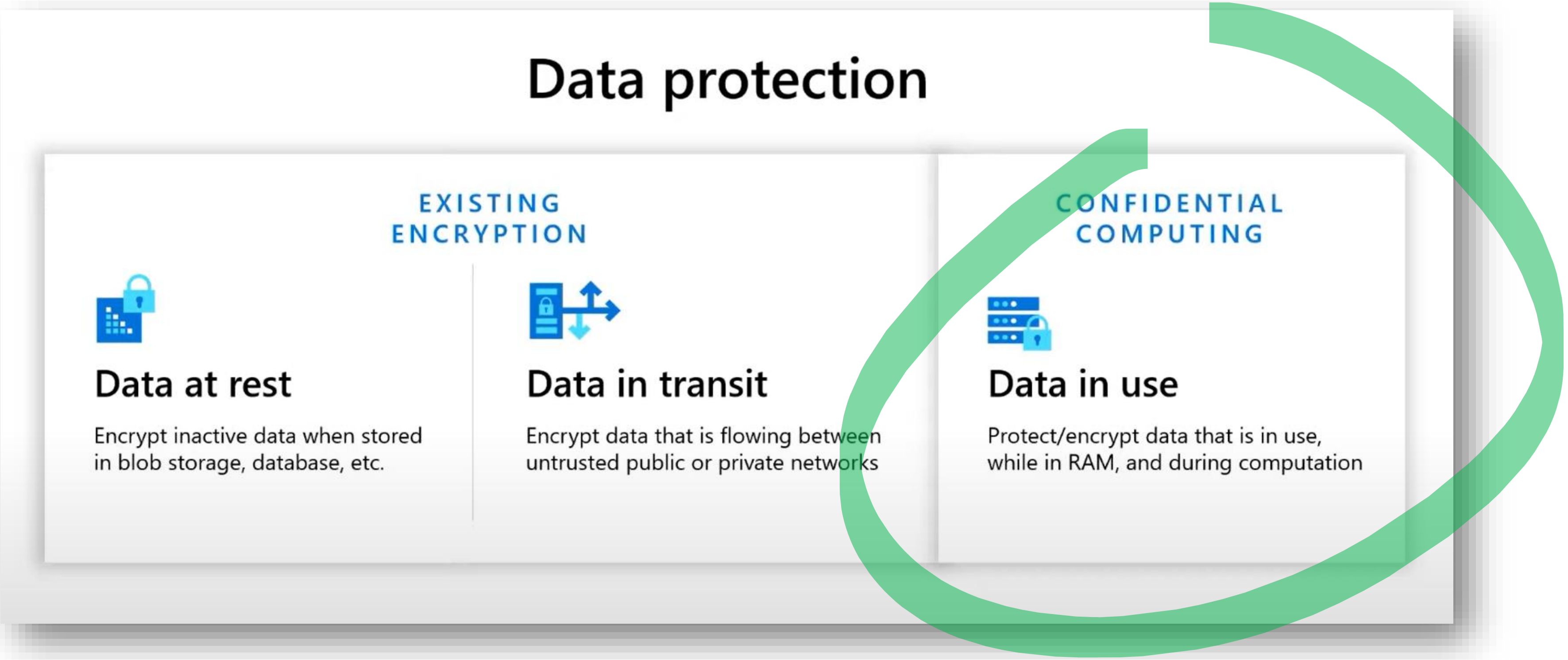
github.com/mkulke

Microsoft Azure

Itinerary 📋

- Why Confidential Containers?
- Key concepts and technologies
- Demo

Sales Pitch! 👔



Data protection

EXISTING ENCRYPTION

Data at rest

Encrypt inactive data when stored in blob storage, database, etc.

Data in transit

Encrypt data that is flowing between untrusted public or private networks

CONFIDENTIAL COMPUTING

Data in use

Protect/encrypt data that is in use, while in RAM, and during computation

[Journey towards the Confidential Cloud]

Confidential Computing: Definition

> Confidential Computing is the protection of data in use by performing computation in a hardware-based, attested Trusted Execution Environment.

Confidential Computing Consortium

Why be concerned about „Data in Use“?

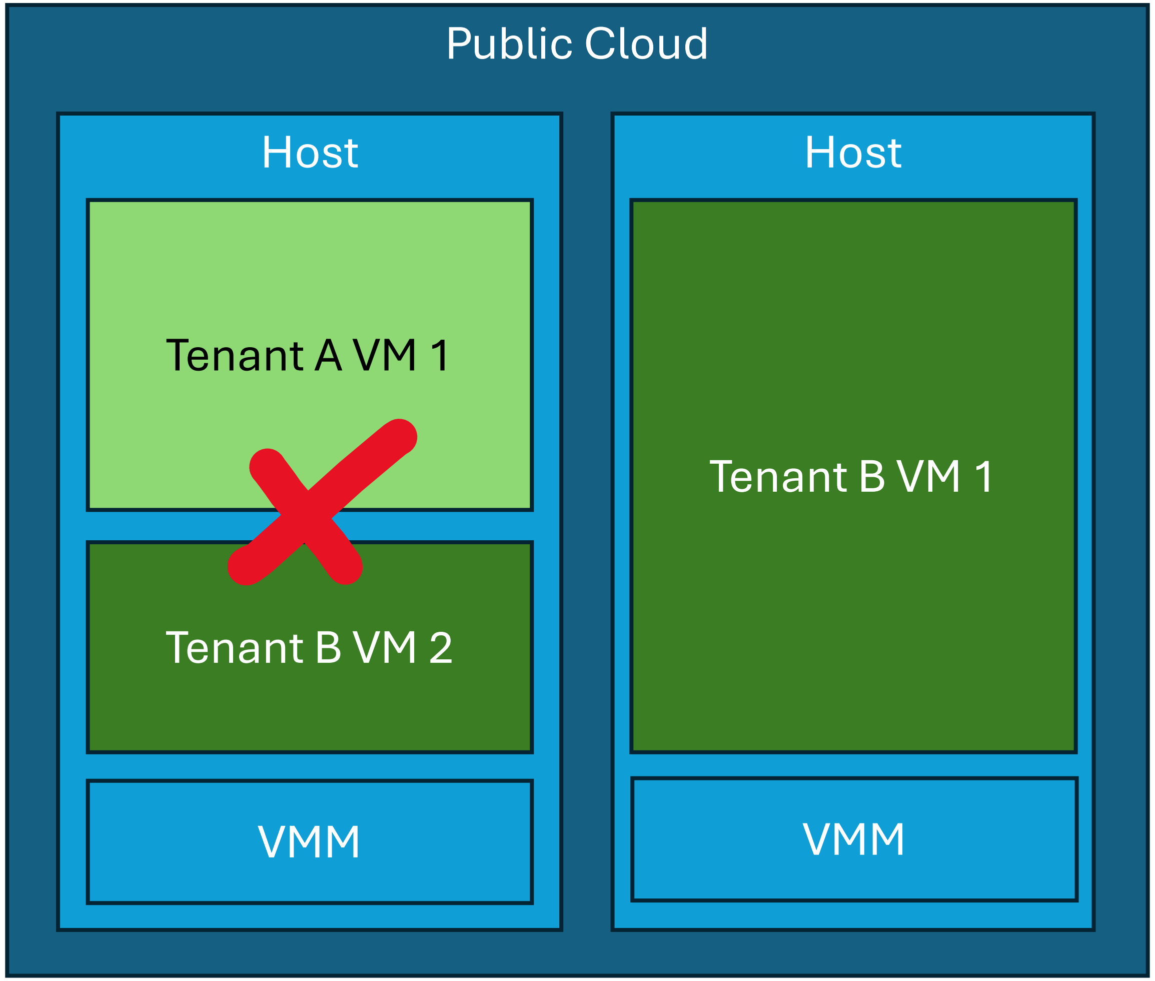
- Remote compute landscape
  - Edge
  - Cloud
- Processing sensitive data with strong privacy requirements
  - LLM agents
- Multi-party computations
  - Train models for fraud detection
  - Health data collaboration
- Security
  - Reduce fallout of compromises, data leaks

Reminder: VM state is transparent to the Host

State of a Virtual Machine ~

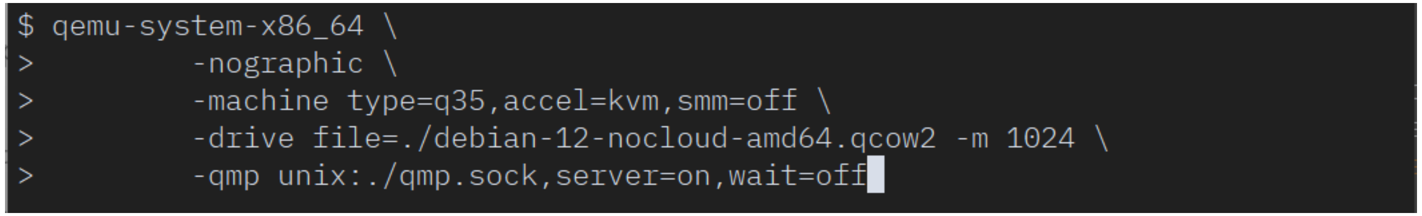
- RAM
- CPU-Register
- Caches

- …



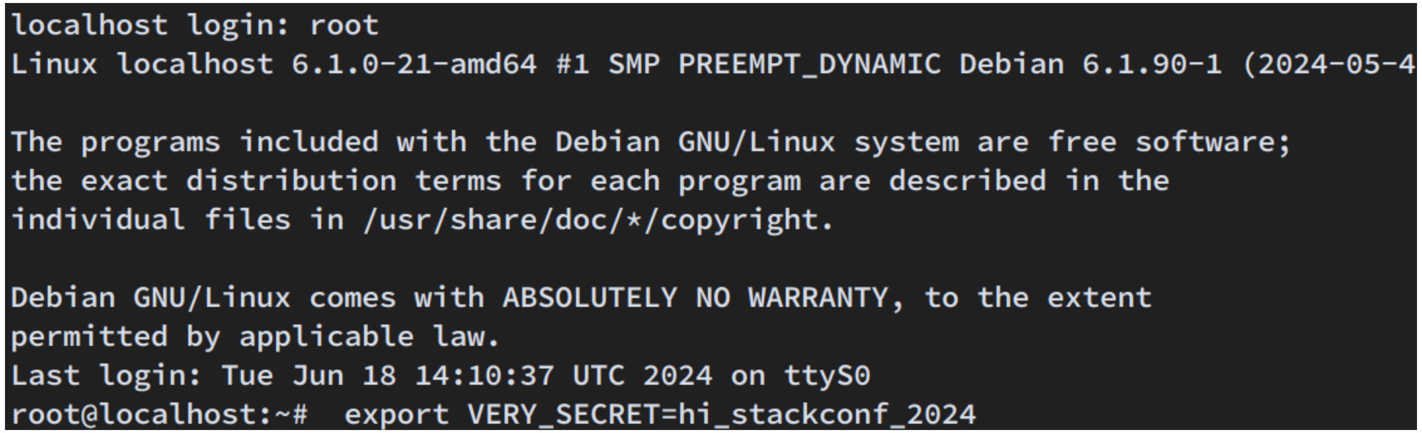
Detour: inspect memory of a VM

Start QEMU VM w/ management socket enabled



```
$ qemu-system-x86_64 \
>         -nographic \
>         -machine type=q35,accel=kvm,smm=off \
>         -drive file=./debian-12-nocloud-amd64.qcow2 -m 1024 \
>         -qmp unix:./qmp.sock,server=on,wait=off
```

Set a secret in the VM shell (w/o persisting it)



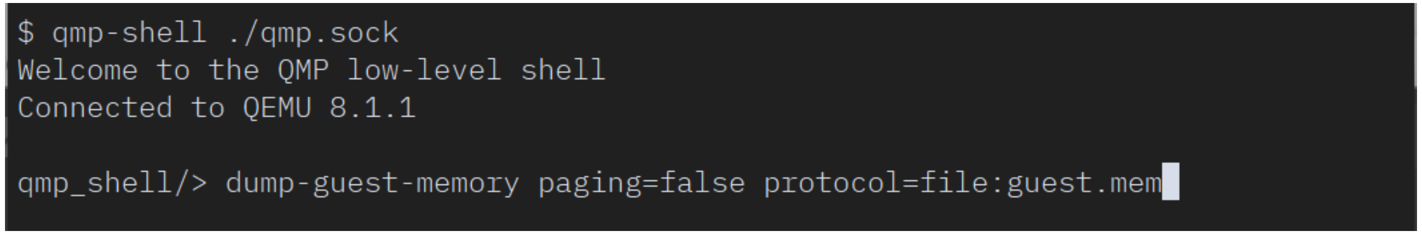
```
localhost login: root
Linux localhost 6.1.0-21-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.90-1 (2024-05-4

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jun 18 14:10:37 UTC 2024 on ttyS0
root@localhost:~#  export VERY_SECRET=hi_stackconf_2024
```

Detour: Read Memory of a VM

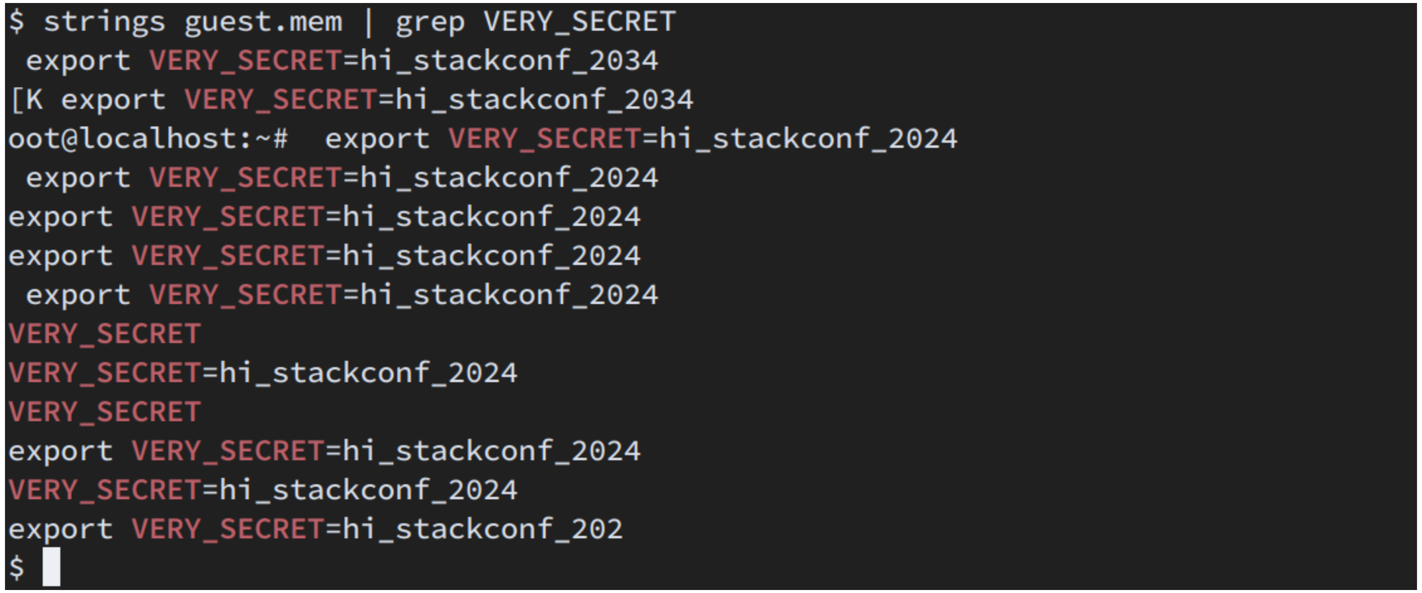
Create memory dump in the QEMU management console



```
$ qmp-shell ./qmp.sock
Welcome to the QMP low-level shell
Connected to QEMU 8.1.1

qmp_shell/> dump-guest-memory paging=false protocol=file:guest.mem
```

Search for secret in the dump file



```
$ strings guest.mem | grep VERY_SECRET
 export VERY_SECRET=hi_stackconf_2034
[K export VERY_SECRET=hi_stackconf_2034
oot@localhost:~#  export VERY_SECRET=hi_stackconf_2024
 export VERY_SECRET=hi_stackconf_2024
export VERY_SECRET=hi_stackconf_2024
export VERY_SECRET=hi_stackconf_2024
 export VERY_SECRET=hi_stackconf_2024
VERY_SECRET
VERY_SECRET=hi_stackconf_2024
VERY_SECRET
export VERY_SECRET=hi_stackconf_2024
VERY_SECRET=hi_stackconf_2024
export VERY_SECRET=hi_stackconf_202
$ 
```

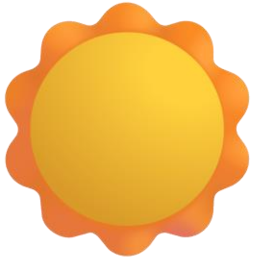
Why Confidential *Containers* then? 🏗️

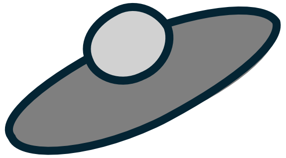
- *Confidential Computing* (CC) is not trivial to implement and deploy
- *Cloud Native* is a popular platform/interface for applications
- Rationale: CC will become accessible and popular once it blends into the container ecosystem
- Pledge: Application don’t require costly modifications: **Lift and Shift** them into a TEE

Key concepts

Trust 🤝 | Integrity 👼 | Attestation 📜

Trust in cloud computing





Trust no one?

Trust in cloud computing

Not quite. There is implicit trust in:

- Our code
- Deployments
- Operations teams
- Cloud service providers
- CPU vendors
- …

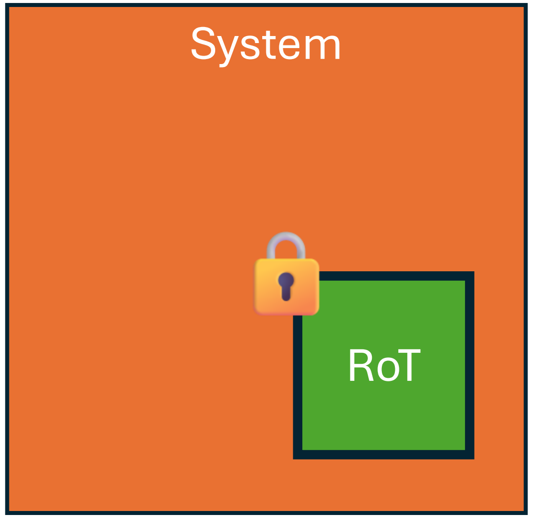
Trust in cloud computing

CC wants to shrink the trust boundary:

- Our code
- ~~Deployments~~
- ~~Operations teams~~
- ~~Cloud service providers~~
- CPU vendors (GPU, too)

What/who to trust then?

- Assumption: An environment is not trustworthy by itself
- Hence, we can only trust an entity, which isn’t part of that environment
- A hardware **Root of Trust** (RoT) is deliberately isolated from the rest of the environment (e.g. HSM or TPM modules)



System

RoT

Thales PCIe HSM

GIGABYTE TPM 2.0 Module

Hardware Root-of-Trust

> The general idea is to add a new chip (...) to your computer that you don't get to run code on.

Educated Guesswork Blog: Do you know what your computer is running?

Integrity: What is our machine running?

- Is the running workload really matching our specs?
- Has our application/OS/environment been compromised?
- **Measurements**: Assess the integrity of a system via cryptographic hash functions.

💾 📏 => 1234abcd

Example: Hash as checksum for binaries

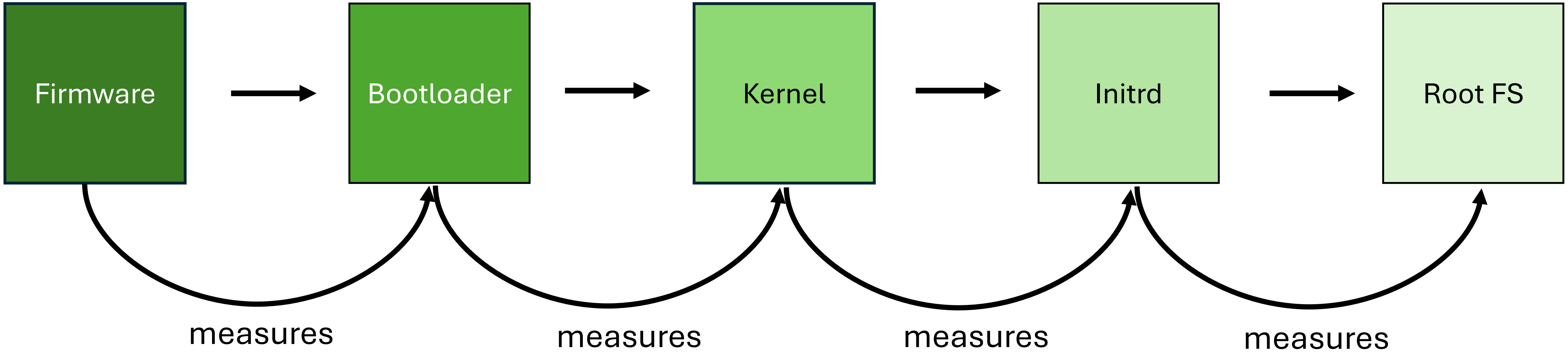
```
$ curl -sLO https://dl.k8s.io/release/v1.29.2/bin/linux/amd64/kubectl
$ curl -sLO https://dl.k8s.io/release/v1.29.2/bin/linux/amd64/kubectl.sha256
$ echo "$(cat kubectl.sha256) kubectl" | sha256sum -check
kubectl: OK
```

Hash extension

```
>>> m = hashlib.sha256()
>>> m.update(b'value1')
>>> m.update(m.digest() + b'value2')
>>> m.hexdigest()
'5b848becb4a8d7b1515dbd43472cd3d66e7d027d83fa004d1bb158cf1a248802‘
```

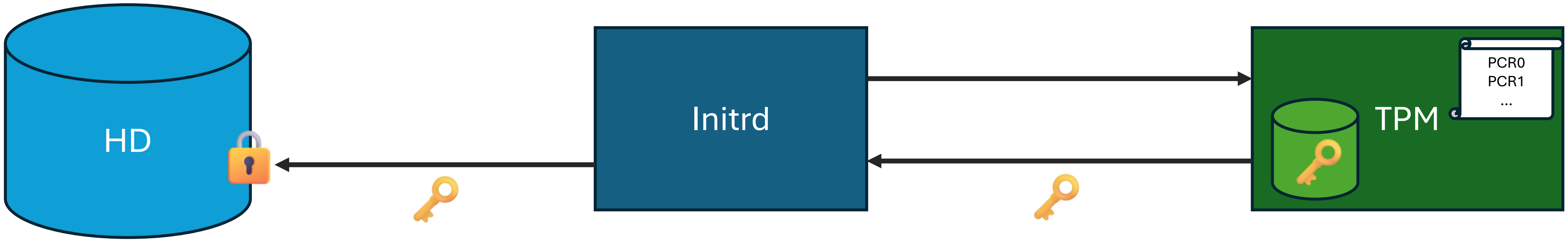
Hash extension as record of events

- Record sequence of events, e.g., Linux measured boot
  - One boot stage measures the next one
  - Deterministic, predictable, replayable
- Related concepts: Git, Blockchains



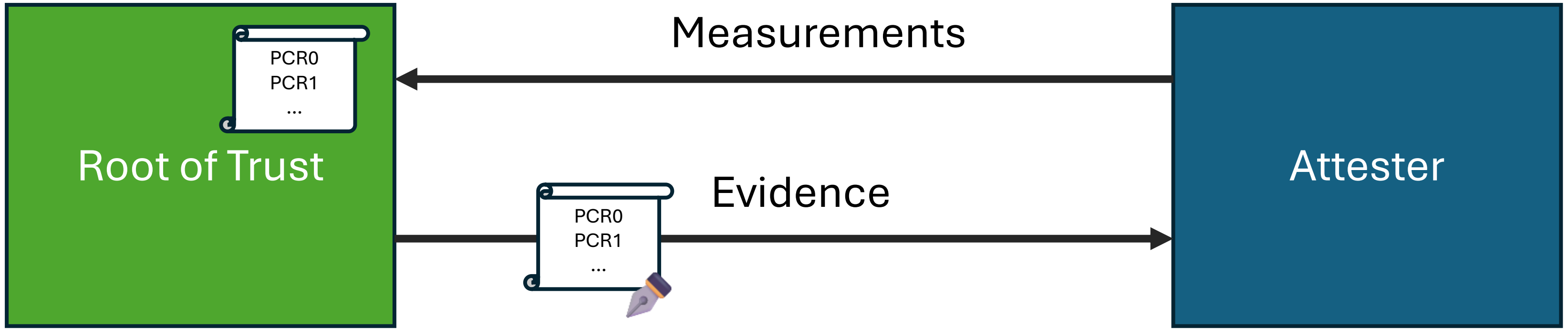
Example: Disk unlock with TPM

- TPMs have extend-only hash registers (PCRs) and can seal keys
- Boot process is measured into PCRs
- TPM will unseal the disk decryption key only for a given set of PCR reference values (local **attestation**)
- A compromised system (FW, kernel, etc.) would not boot



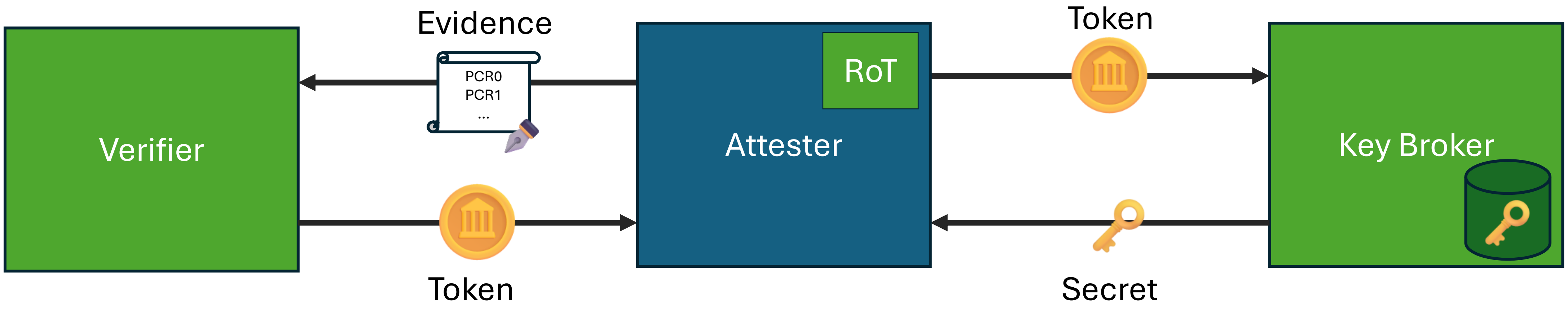
Remote Attestation

- Verification and Key Storage/Release is being performed on a remote system (Verifier, Relying Party)
- RoT hardware will gather measurements about the state of a system (TPM Quote, Launch Measurement)
- RoT signs measurements with a secret + hw-unique asymmetric key (Evidence, Attestation Report)



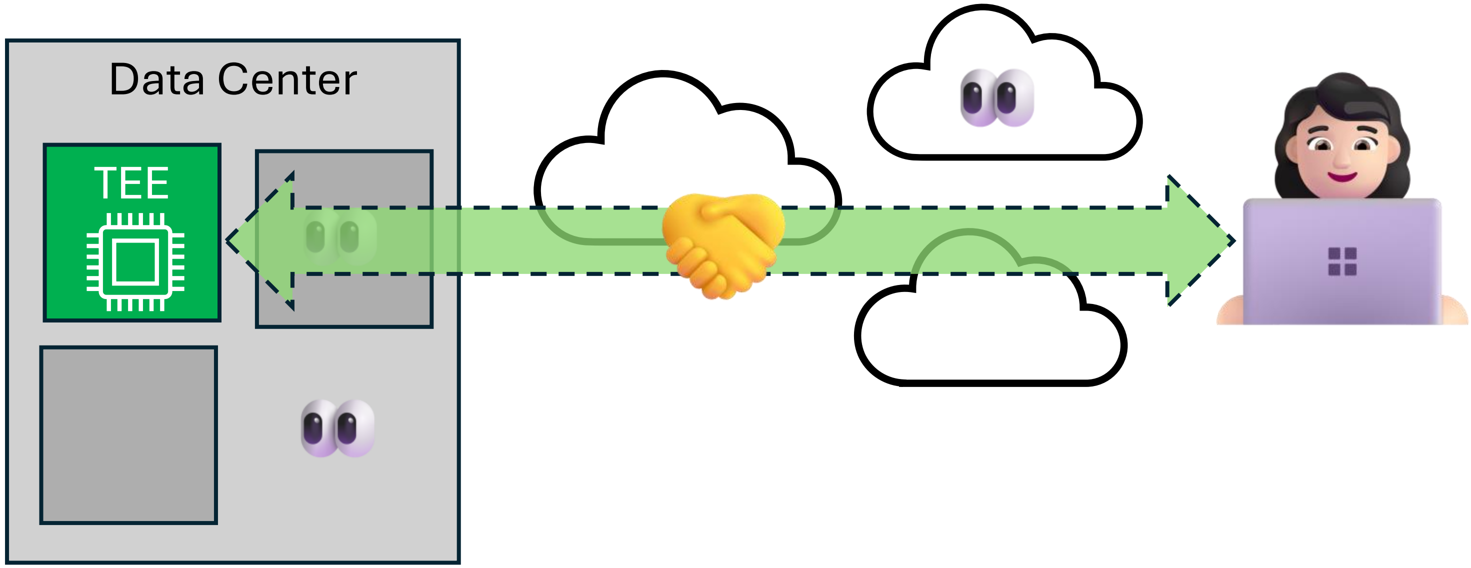
Remote attestation flow (Passport model)

- Evidence can be validated by a trusted, remote Verifier instance
- Verification process asserts:
  - Evidence has been generated & signed by an authentic HW RoT
  - Measurements match expected **Reference Values**
- Verifier will yield tokens to retrieve secrets from Key Broker



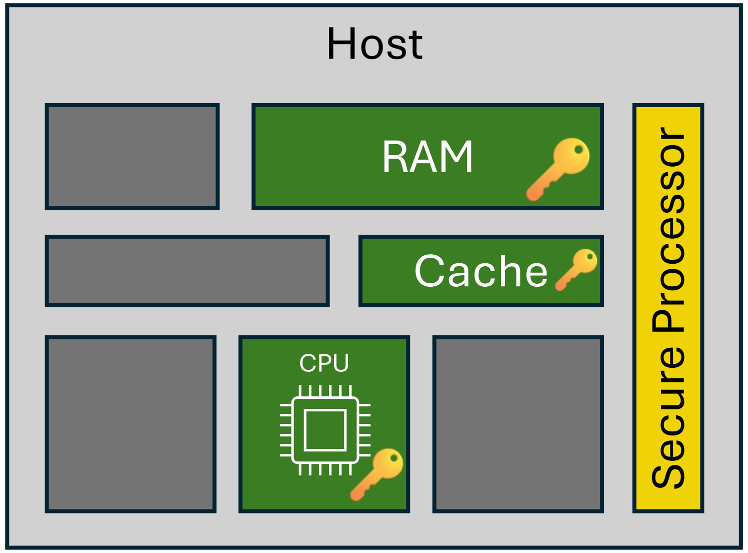
Confidentiality = Integrity + Privacy

CC is a CPU (+ GPU) technology that provides privacy guarantees for the user in a **Trusted Execution Environment** (TEE).



Confidential VM (CVM)

- VM is isolated from Host using memory encryption and integrity protection
- Secure Processor: Hardware RoT
- HW encryption is a key property of a CC TEE



Examples: AMD SEV-SNP, Intel TDX, ARM CCA, IBM SE

Attesting CVMs

- CVM is initialized w/ fixed CPU + RAM State, measured into the TEE HW as **Launch Measurement.**
- Guest can request an **Attestation Report** (AR) from TEE HW, evidence signed with a sealed key
- AR contains facts about TEE
  - Active encryption features
  - Launch measurement
  - Secure processor firmware revision
- Verifier inspects the AR
  - Valid signature? (using certificate chain from HW vendor)
  - Encryption features enabled?
  - Launch measurement matches reference?

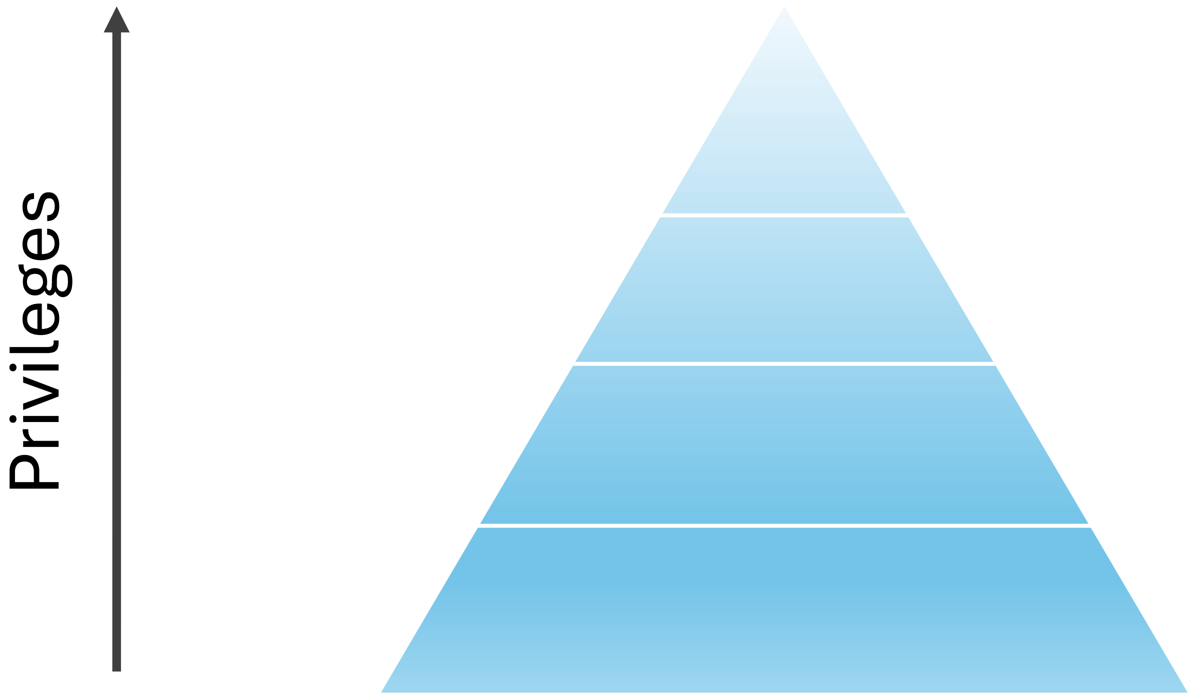
Confidential Containers?

- Integrity ✅
- Trust ✅
- Attestation ✅
- Confidentiality ✅
- Containers ✅

=> Confidential Containers

?

(Simplified) privilege model for Kubernetes



Users

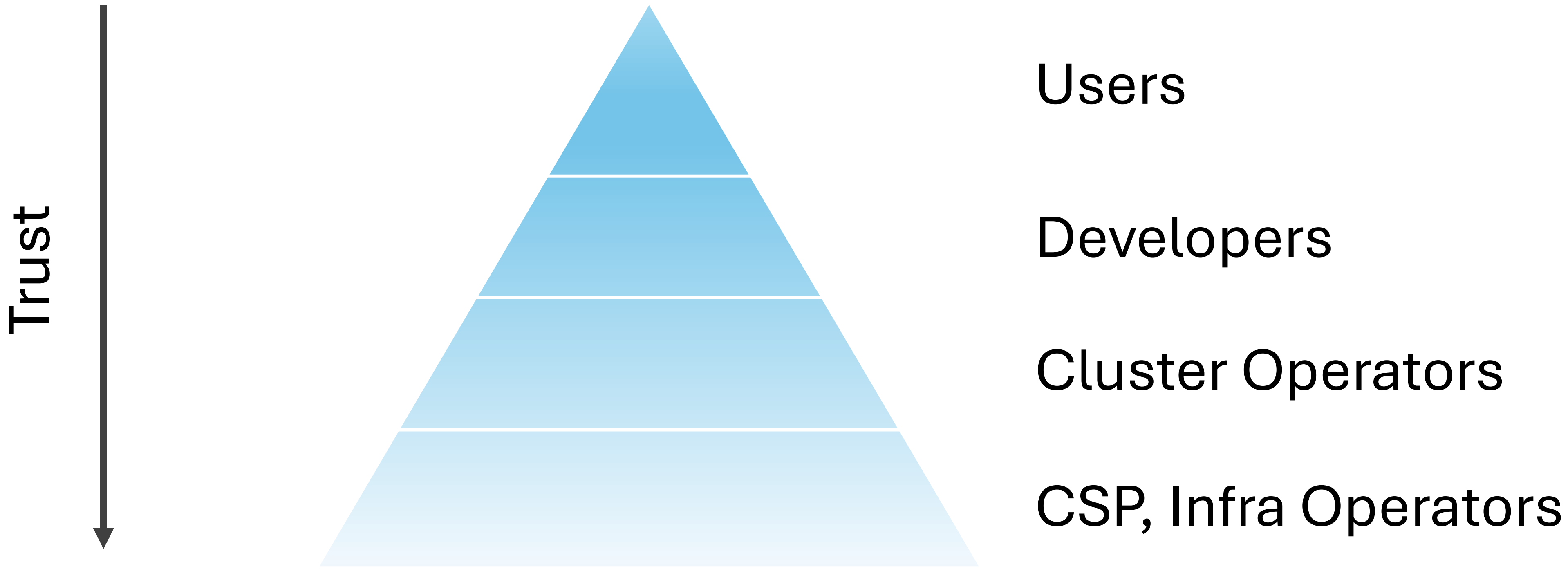
Developers

Cluster Operators

CSP, Infra Operators

Rationale: protect cluster, infra, siblings from malicious users

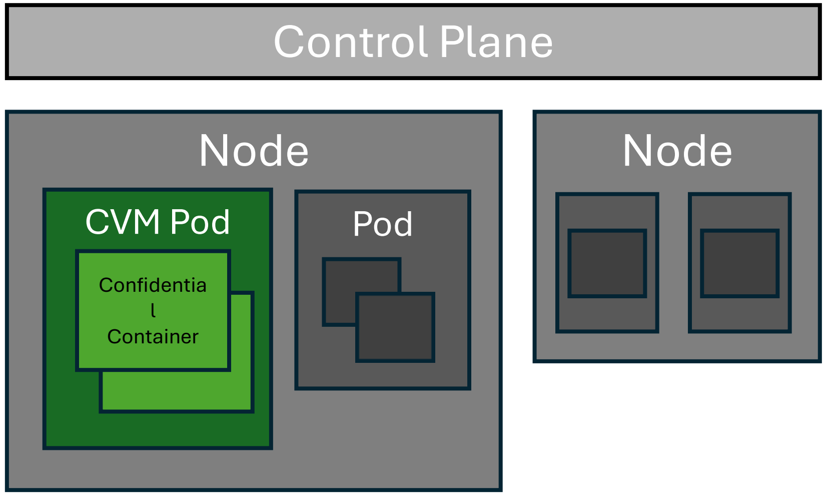
Model of Trust for Confidential Containers



Rationale: Protect the **Trusted Domain**, de-privilege the operators

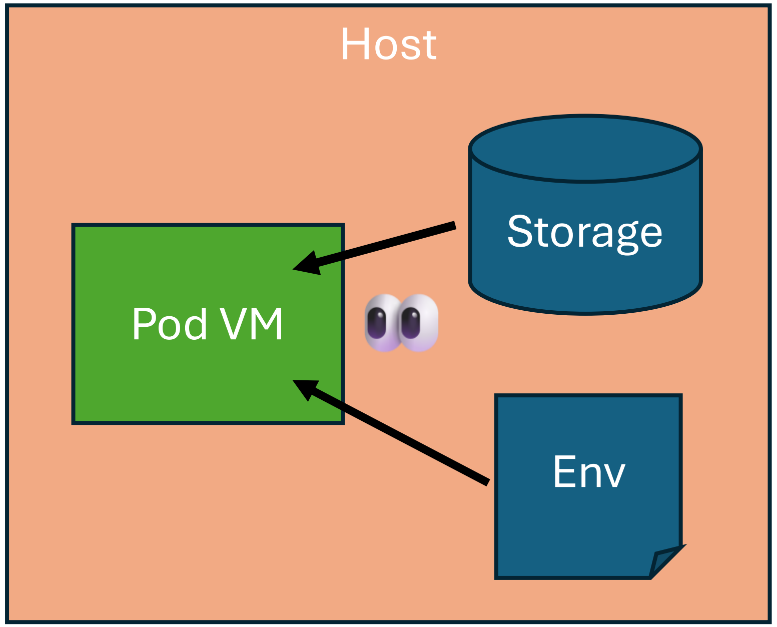
CVMs + Confidential Containers

- Multiple options w/ various tradeoffs: Confidential Cluster/Node/Containers
- Sweet spot: **Confidential Pods** as micro VMs alongside unencrypted Pods
- Prior art for Pod VMs: Kata



Reconciling CC’s and Kubernetes’ paradigms

We do not trust the Node in CoCo. Need to measure Pod spec! All host-injected resources are untrusted!



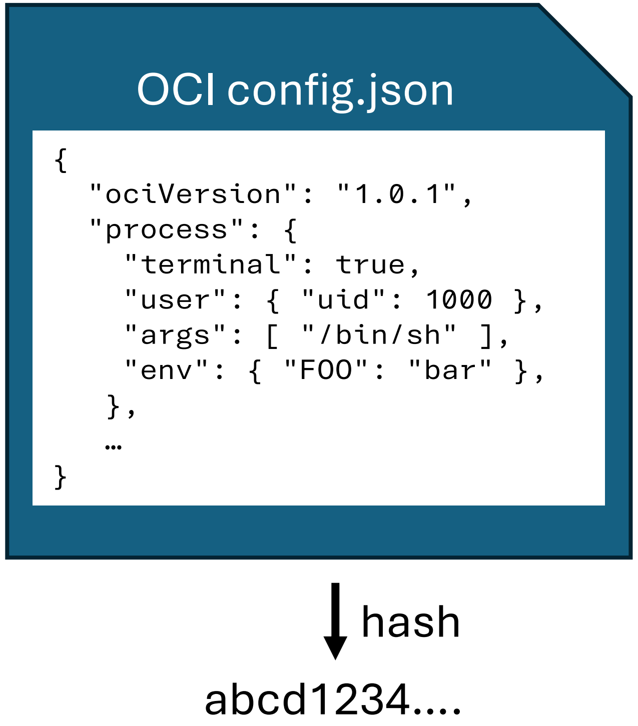
Examples:

- Environment vars (REDIS_HOST)
- Storage (EmptyDir)
- Config maps (/etc/nginx)
- Image Layer/Metadata caches
- Runtime APIs (exec, cp)

Covering inherent dynamisms in Pod specs

Example: *_SERVICE_HOST/PORT env

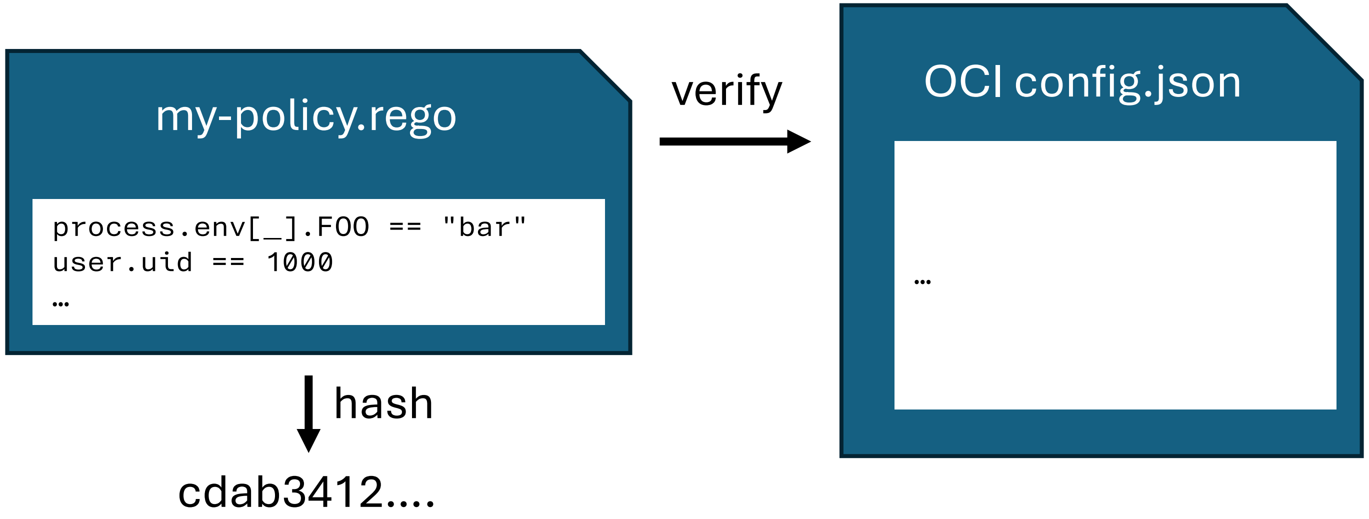
Not something that we can reasonably predict in many clusters



```
{
  "ociVersion": "1.0.1",
  "process": {
    "terminal": true,
    "user": { "uid": 1000 },
    "args": [ "/bin/sh" ],
    "env": { "FOO": "bar" },
   },
   …
}
```

Solution: Express dynamic properties in policies. Validate at launch.

Downside: requires additional tooling/manual work



```
process.env[_].FOO == "bar"
user.uid == 1000
…
```

Confidential Containers, as of today

Asciinema demo

Takeaways

- Cloud Native is an attractive platform for Confidential Computing
- Trust + Integrity + Remote Attestation are key concepts that we need to pick up to leverage CC
- Things are hairy. Confidential Containers have specific, non-trivial challenges

thx! 🙏

Links

- Confidential Containers (github.com)
- confidentialcontainers.org
- Kata Containers (github.com)
- TPM-backed Full Disk Encryption is coming to Ubuntu
- RFC 9334: Remote ATtestation procedureS (RATS) Architecture
- AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More
- Intel® Trust Domain Extensions (Intel® TDX)
- IBM Secure Execution for Linux