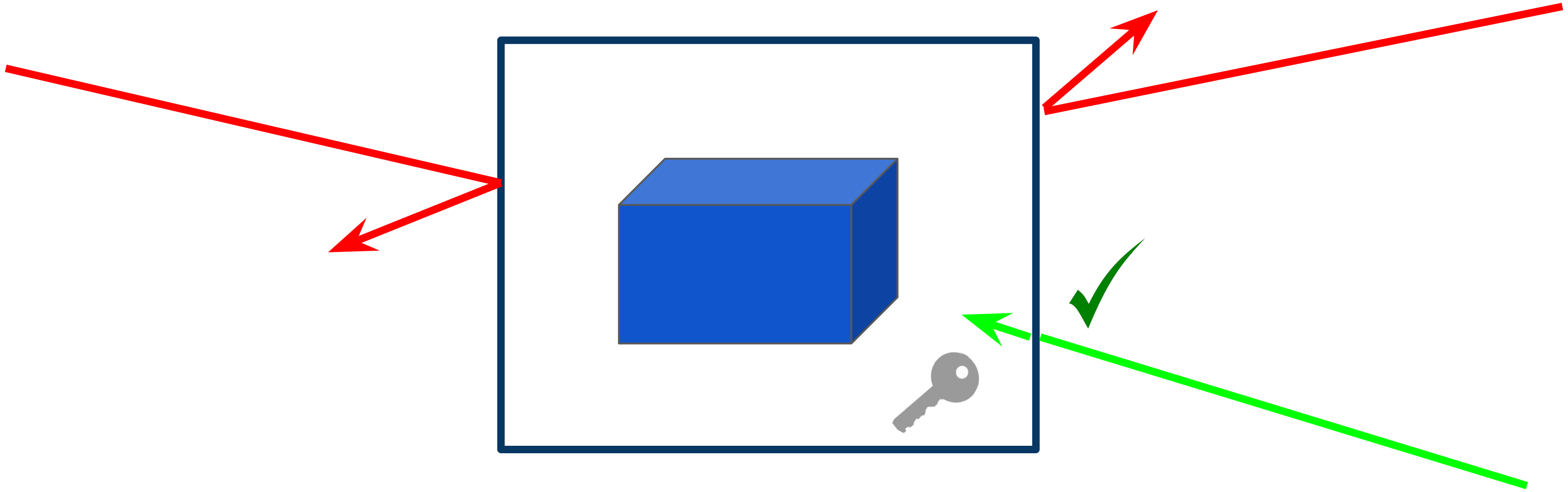
Don’t Peek Into my Container!

Christophe de Dinechin, Red Hat
Alice Frosi, Red Hat
Sergio López Pascual, Red Hat

Today's Topics

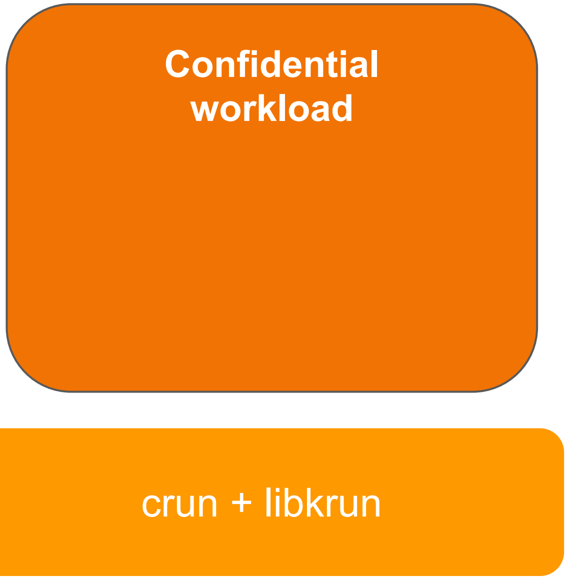
- Confidential computing
- Confidential workloads with k8s and libkrun
- SEV-enabled libkrun
- From Kata Containers to Confidential Containers

“Confidential Computing is the protection of data in use by performing computation in a hardware-based Trusted Execution Environment”

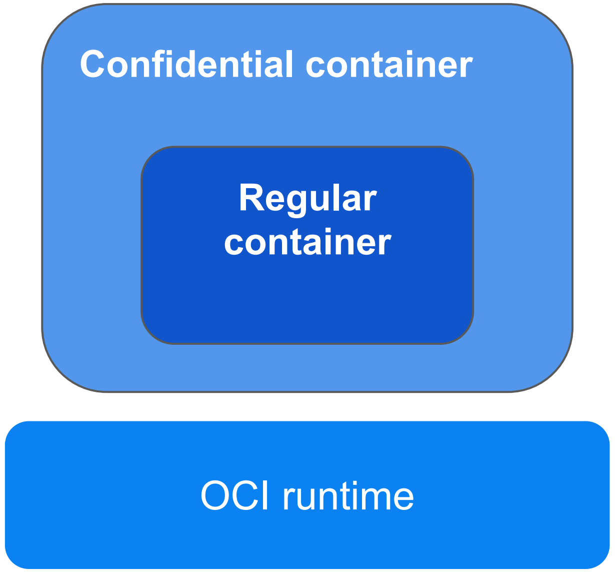


https://confidentialcomputing.io/whitepaper-02-latest/

Confidential workloads are transformed containerized workloads into a special form that can be deployed with libkrun and confidential computing technologies



Confidential containers are the deployment of a regular containers with an OCI runtime (e.g Kata Containers) and confidential computing technologies



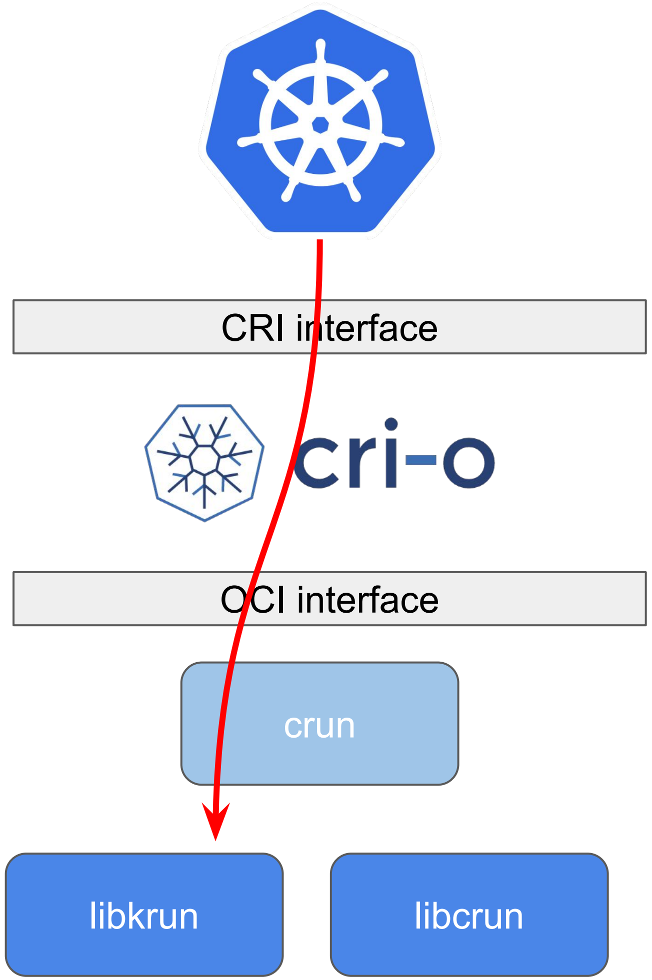
Confidential workloads

- Confidentiality at container level
- Single container per encrypted VM
- Deploy a special form of container image with a single layer
- Simpler architecture and reusing the existing k8s infrastructure

Confidential containers

- Confidentiality at pod level
- Multiple containers per encrypted VM
- Use encrypted layered container images
- Part of the infrastructure is moved inside the trusted environment (e.g image offloading)

Confidential workloads with k8s and libkrun



Receive the pod information

Schedule the workload on a node

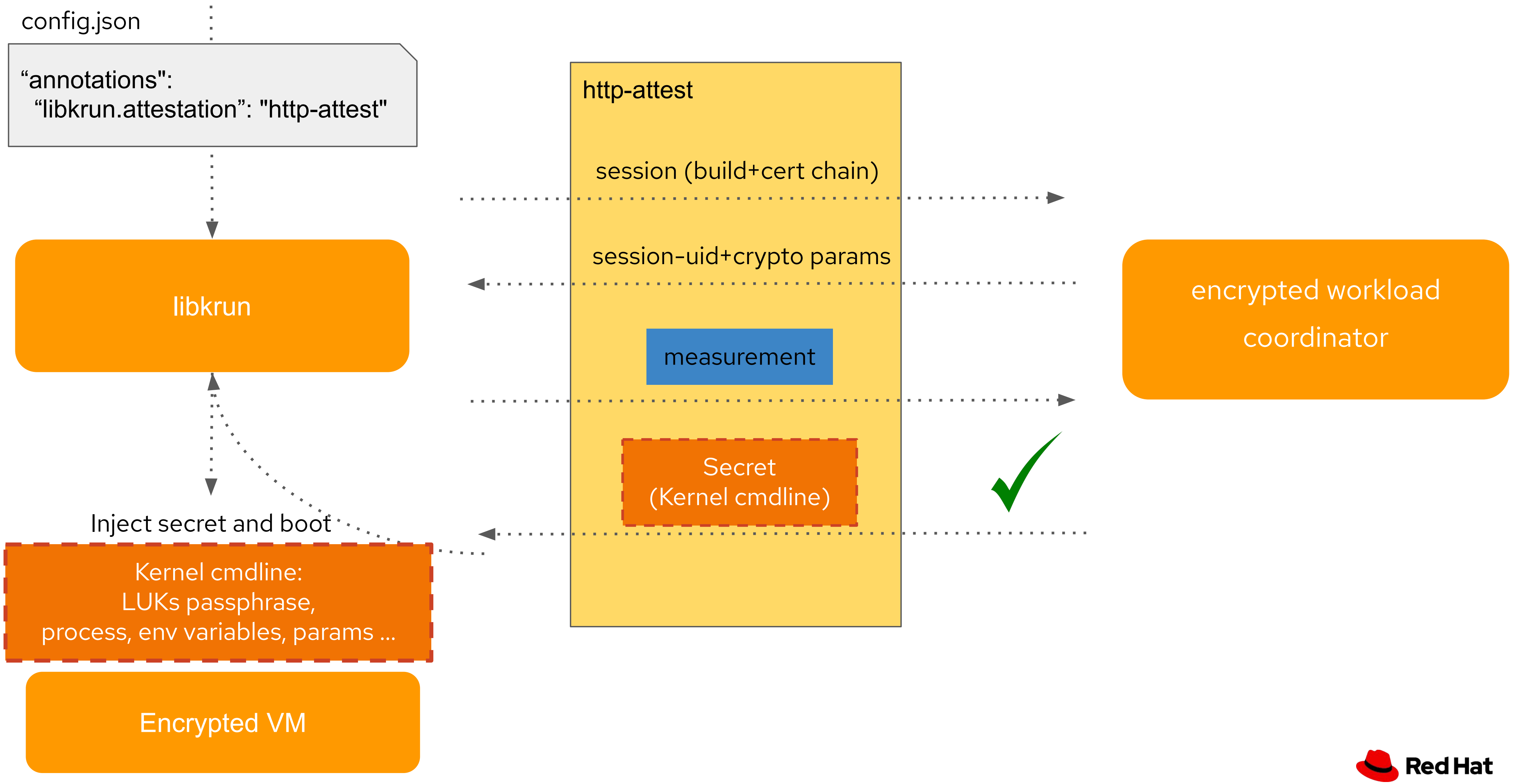
Pull the container image on the node

Prepare the bundle with the rootfs of the container

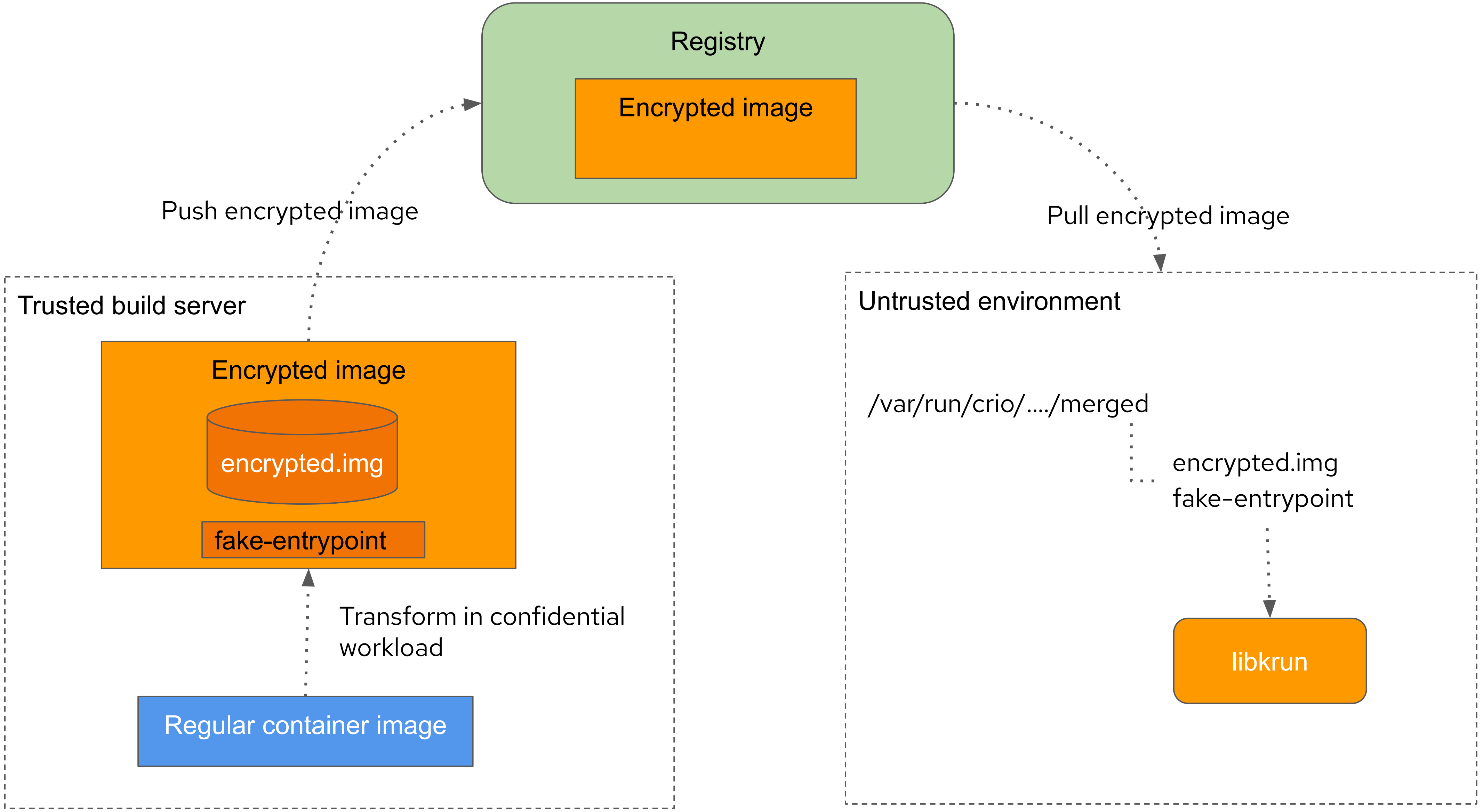
Create the config.json with the container information

Create and run the container

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-krun-sev
  annotations:
    libkrun.attestation: "http-attest"
    run.oci.handler: "krun-sev"
spec:
  containers:
  - image: encrypted/nginx-tls
    name: krun-sev
    command: [“/fake-entrypoint”]
    ports:
    - containerPort: 443
  nodeSelector:
    sev: "true"
```


config.json
“annotations":
“libkrun.attestation”: "http-attest"
libkrun
Inject secret and boot
Kernel cmdline:
LUKs passphrase,
process, env variables, params …
Encrypted VM
http-attest
session (build+cert chain)
session-uid+crypto params
measurement
Secret
(Kernel cmdline)
encrypted workload
coordinator

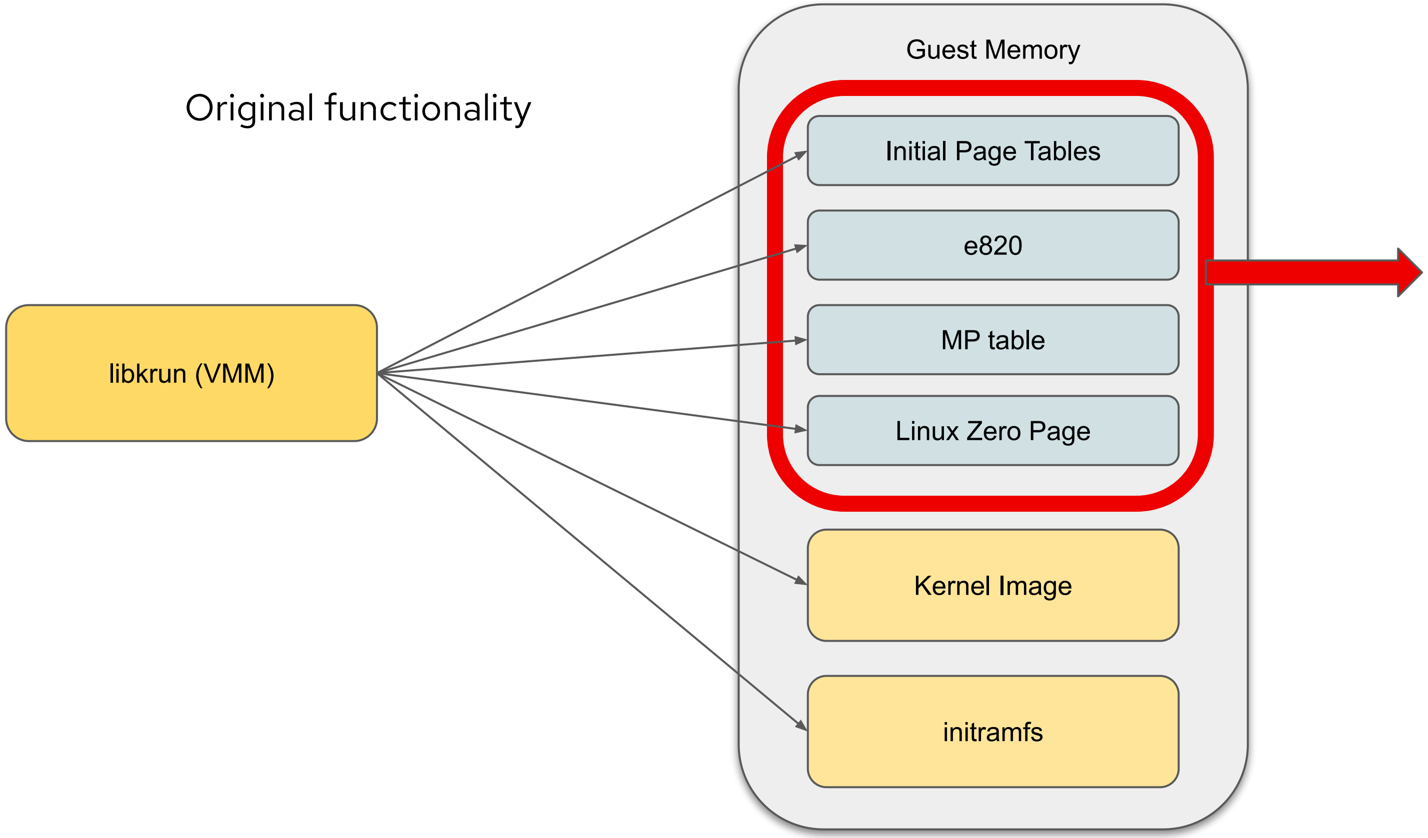
Red Hat


Registry
Encrypted image
Push encrypted image
Pull encrypted image
Trusted build server
Encrypted image
encrypted.img
fake-entrypoint
Transform in confidential workload
Regular container image
Untrusted environment
/var/run/crio/..../merged
encrypted.img
fake-entrypoint
libkrun

SEV-enabled libkrun for Confidential Workloads

The need for a Minimal Firmware (I)

Original functionality

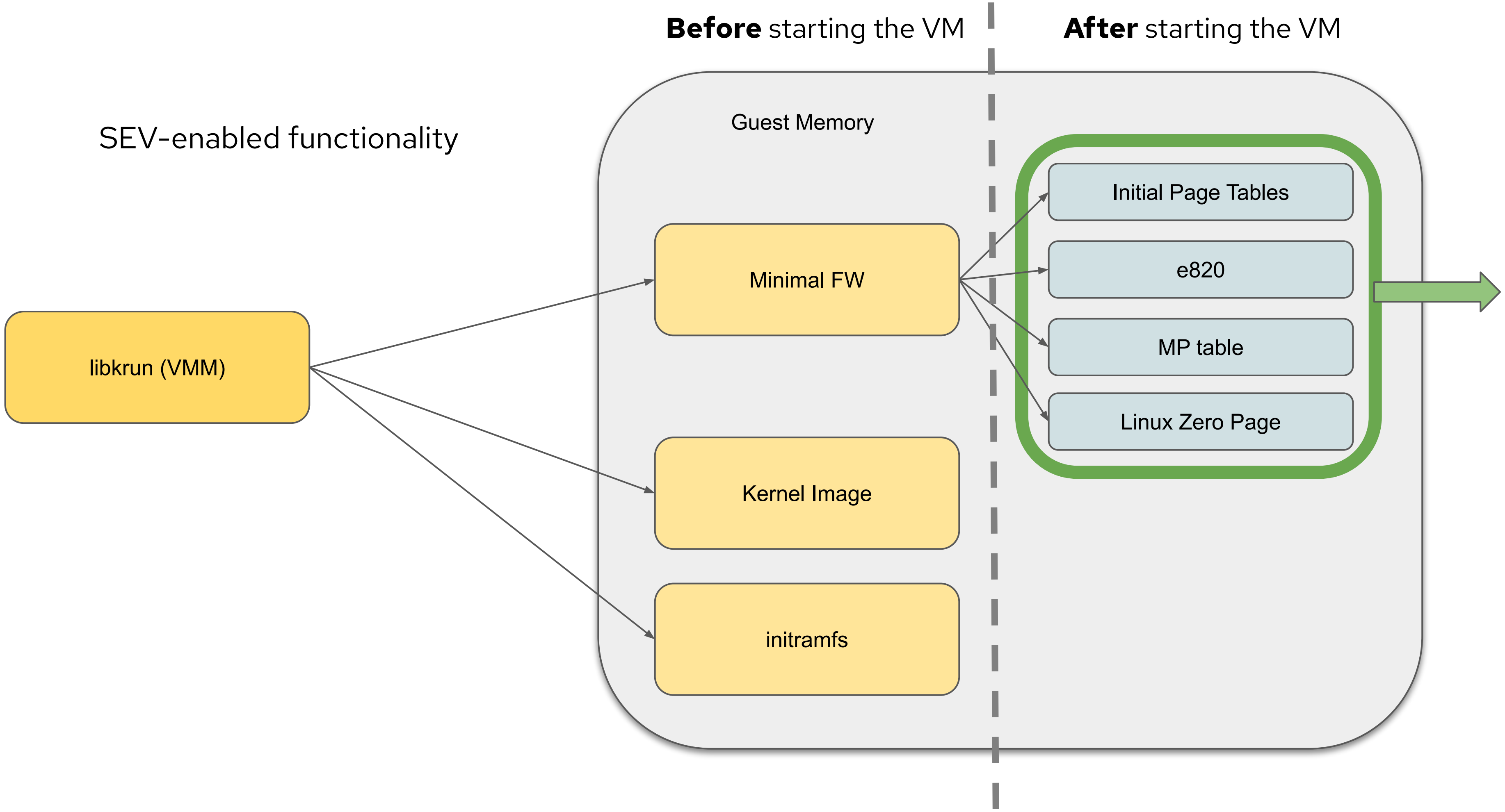


These tables are written by the VMM and become part of the launch measurement!

Red Hat

The need for a Minimal Firmware (II)

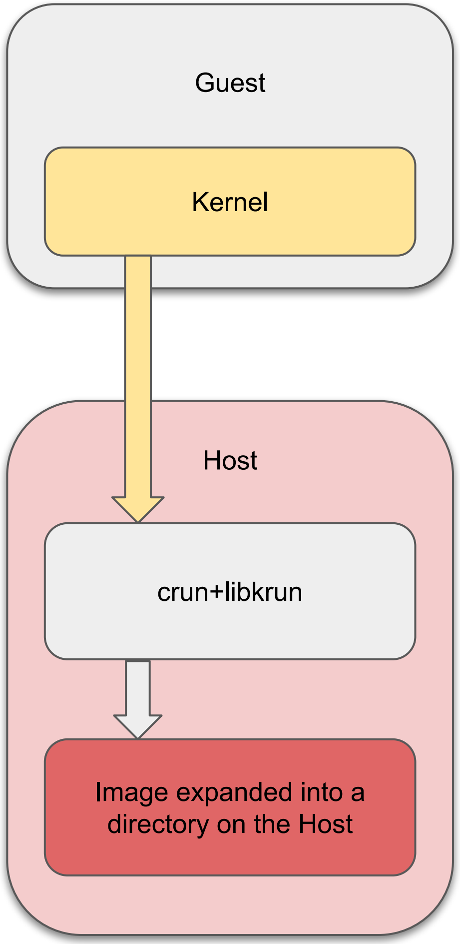
SEV-enabled functionality



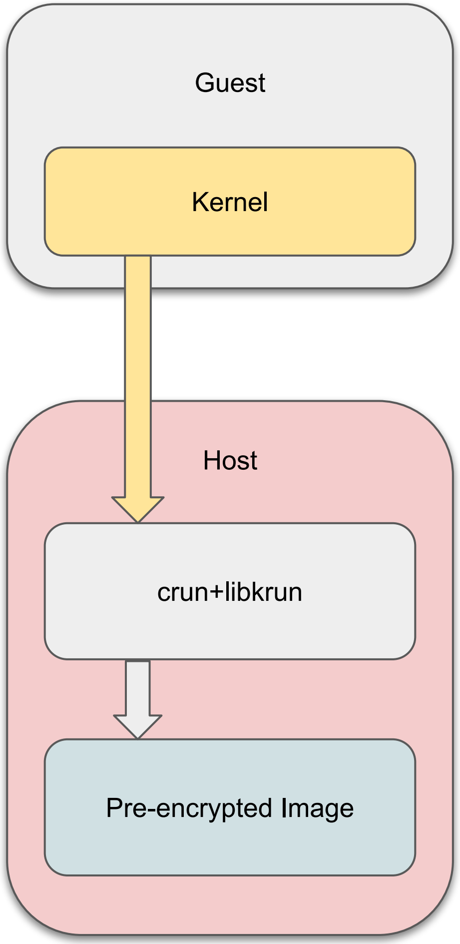
These tables are written by the Firmware and are **NOT** part of the launch measurement!

Replacing virtio-fs with virtio-blk (I)

Regular libkrun with virtio-fs



SEV-enabled libkrun with virtio-blk



Replacing virtio-fs with virtio-blk (II)

- libkrun uses virtio-fs because fits nicely with the container isolation use case.
- But, for the Confidential Workloads use case, virtio-fs is not the best solution.
  - Even if we could find an acceptable filesystem-level encryption mechanism, the implementation will leak too much information.
  - The implementation is quite large and complex (is, by far, the largest component in libkrun) compared with virtio-blk, and requires a large number of syscalls, which implies a more permissive seccomp filter.
    - Lines of Code: virtio-fs = 7444, virtio-blk = 1325

Replacing virtio-fs with virtio-blk (III)

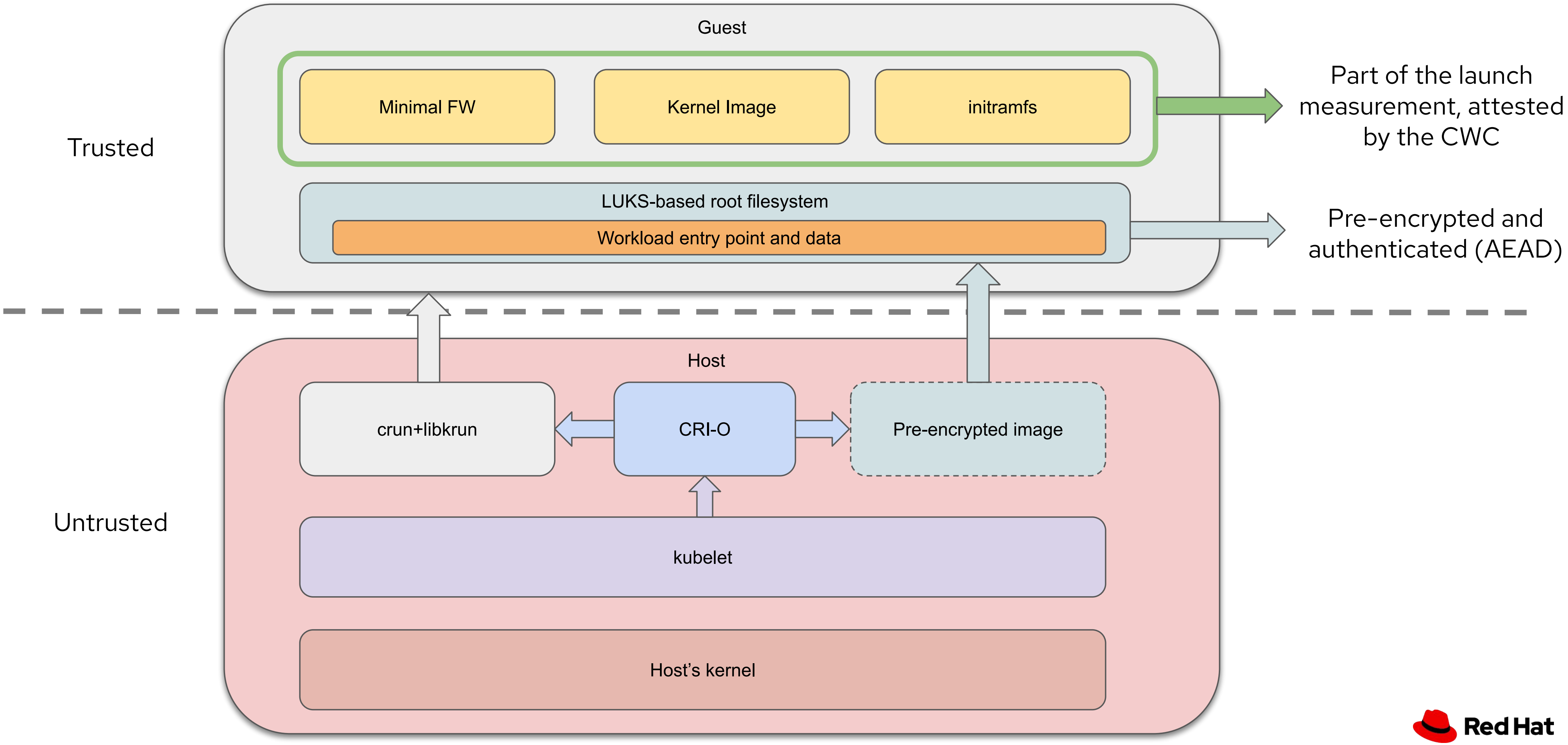
- Using virtio-blk allows us to easily rely on LUKS.
  - LUSK2 has the ability to combine dm-crypt and dm-integrity.
  - Provides both confidentiality and integrity protection (Authenticated Encryption with Additional Data, AEAD).
  - Protects against all known attacks, except data replay, which would require specialized hardware storage.
  - Reference: Practical Cryptographic Data Integrity Protection with Full Disk Encryption Extended Version by Milan Brož, Mikuláš Patočka and Vashek Matyáš.

The need for an initramfs

- The binary we use in libkrun to set up the environment inside the guest is bundled in the integrated virtio-fs server.
  - Without virtio-fs, we needed an alternative.
- We've incorporated a simple initramfs.
  - Includes a variant of the binary to set up the environment, a static version of cryptsetup, and some support directories and device nodes.
  - For SEV-SNP and TDX cases, it'll likely include a small attestation client.
  - Opens the LUKS device (using the injected secret) and continues doing the usual environment adjustments before executing the workload entry point.

Red Hat

The Big Picture

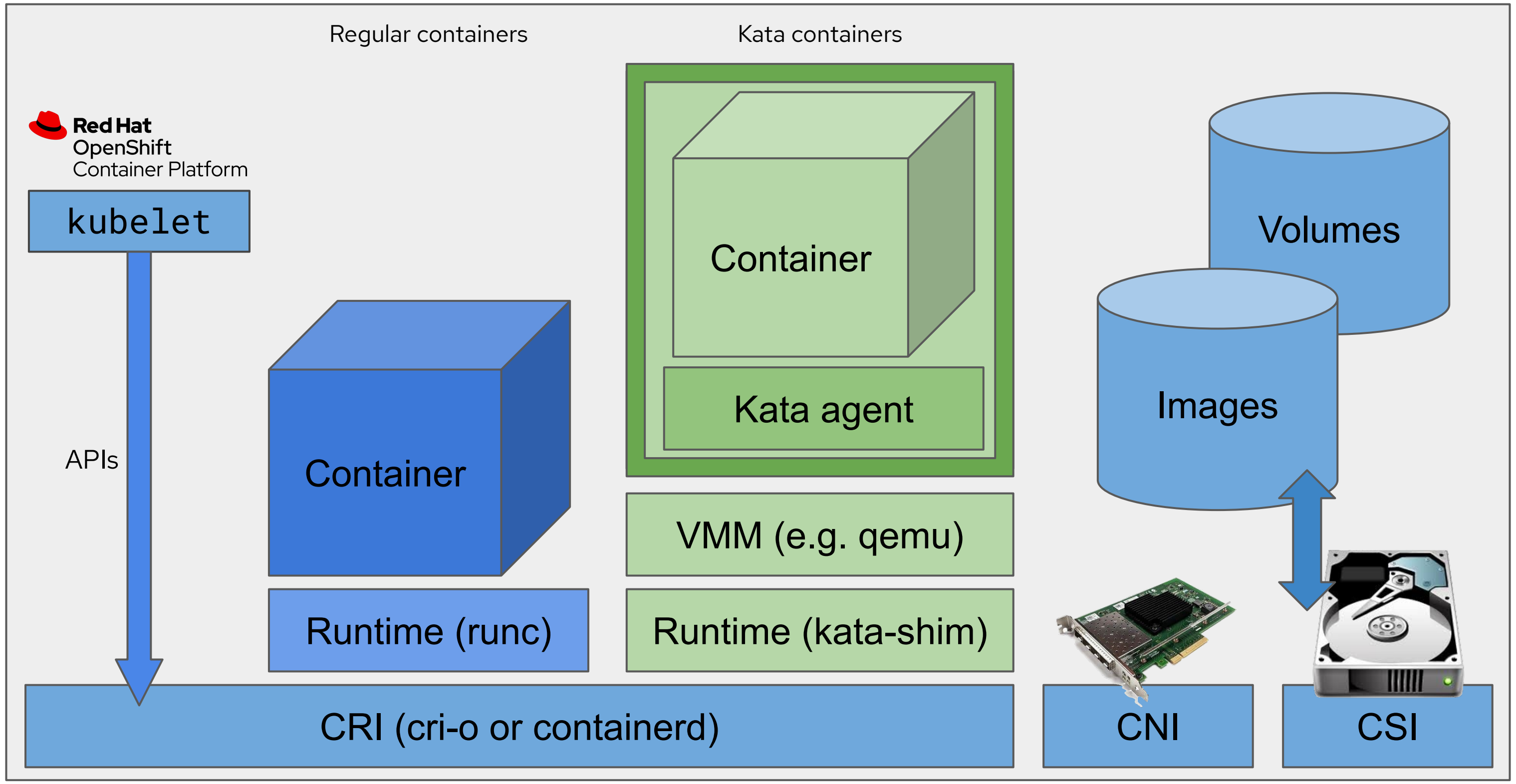


From Kata containers to Confidential containers

Red Hat

Kata Containers overview

- **Run Containers** described the usual way (e.g. same yaml file, images, storage, networking...)
- **... in Virtual Machines** with their own independent kernel and very little user space



The ecosystem of **containers**
The sandboxing of **virtualization**

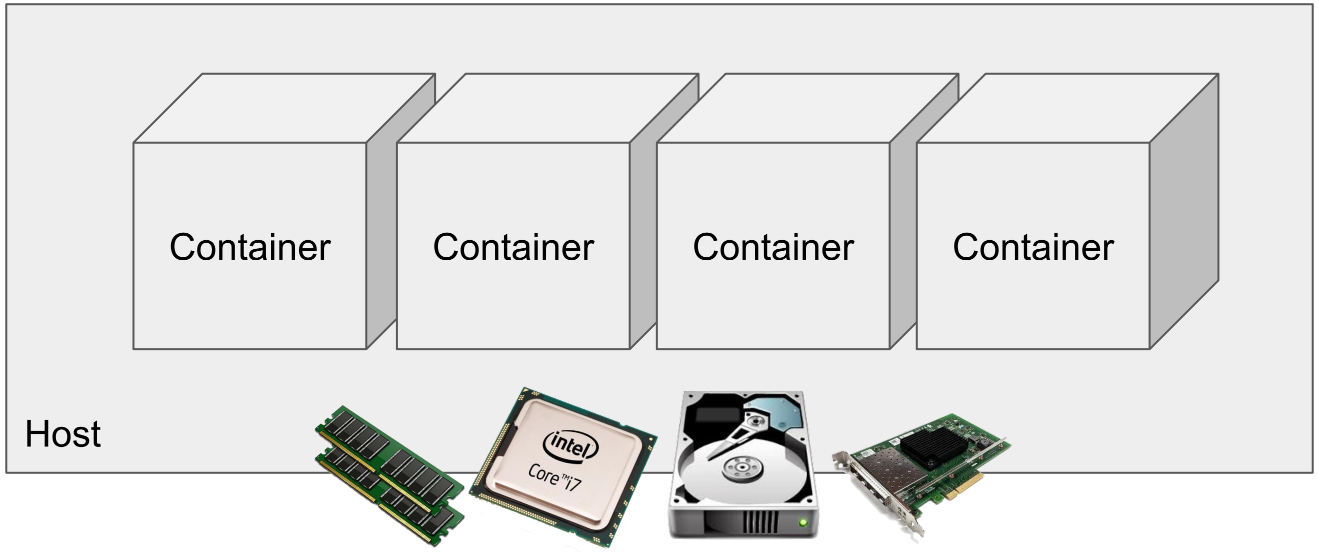
CRI: Container *Runtime* Interface
CNI: Container *Networking* Interface
CSI: Container *Storage* Interface

Problem statement:
Can we trust the host?

Problem Statement: Can we trust your host?

- **Containers** run on a host, often managed by a third party, like a cloud provider
- **Sandboxing** goes only one way, protecting the host from containers, not the other way round
- **Resources** belong to the host, which owns them and has free access

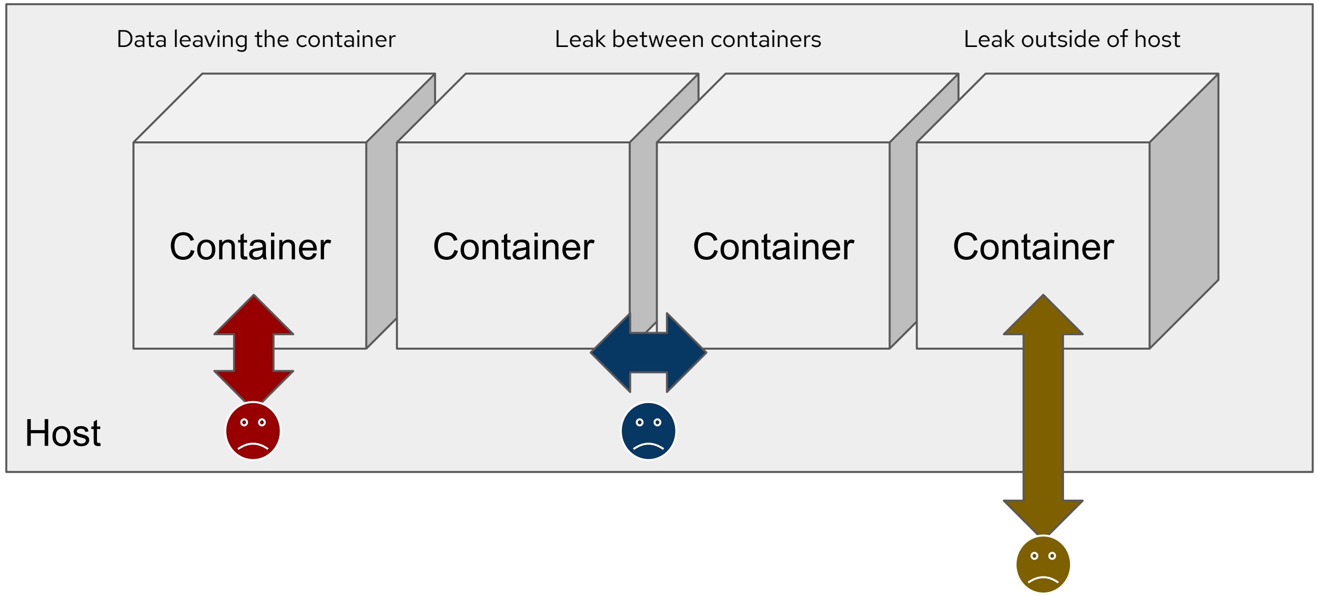
Containers are *carved out* of the host resources



What do you need to do if you start considering the host as **hostile**?

- **Data exposure** of information held in the container is possible
- **Multiple tenants** may not want to share the same host because of confidentiality risk
- **Legal concerns** may preclude the use of containers if you cannot guarantee confidentiality

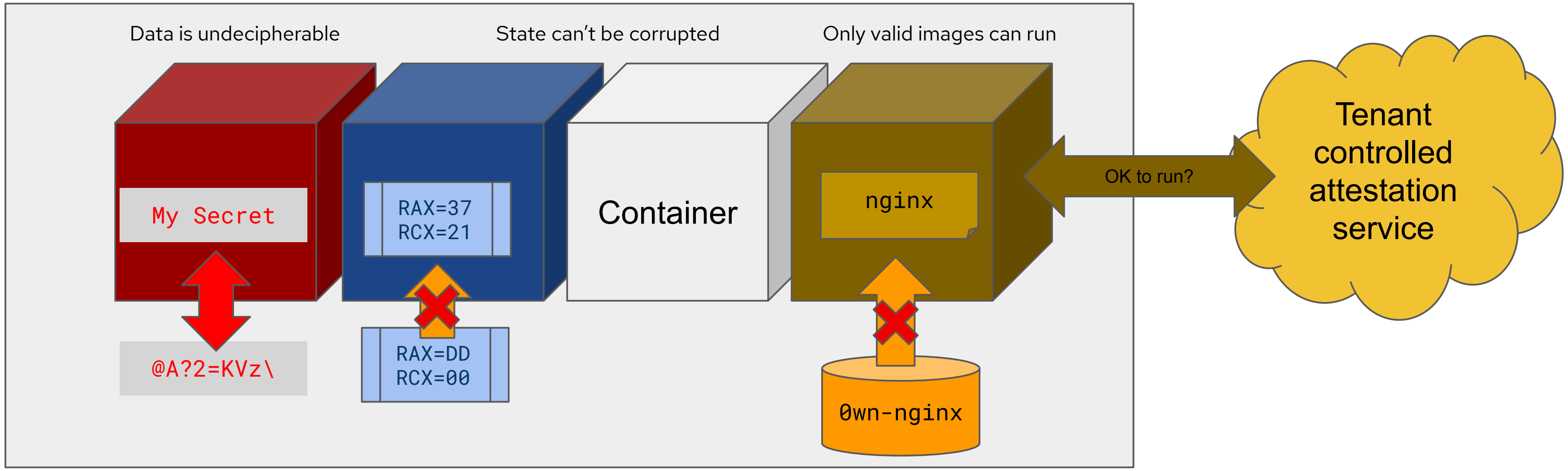
There is a potential for unwanted data leaks



Enabling technology:
Confidential Computing

Confidential Computing: more than encryption...

- **Memory encryption** prevents the host from getting data out of guest memory
- **Integrity protection** offers guarantees about guest state corruption
- **Attestation** lets the guest owner (tenant) validate what runs in the guest



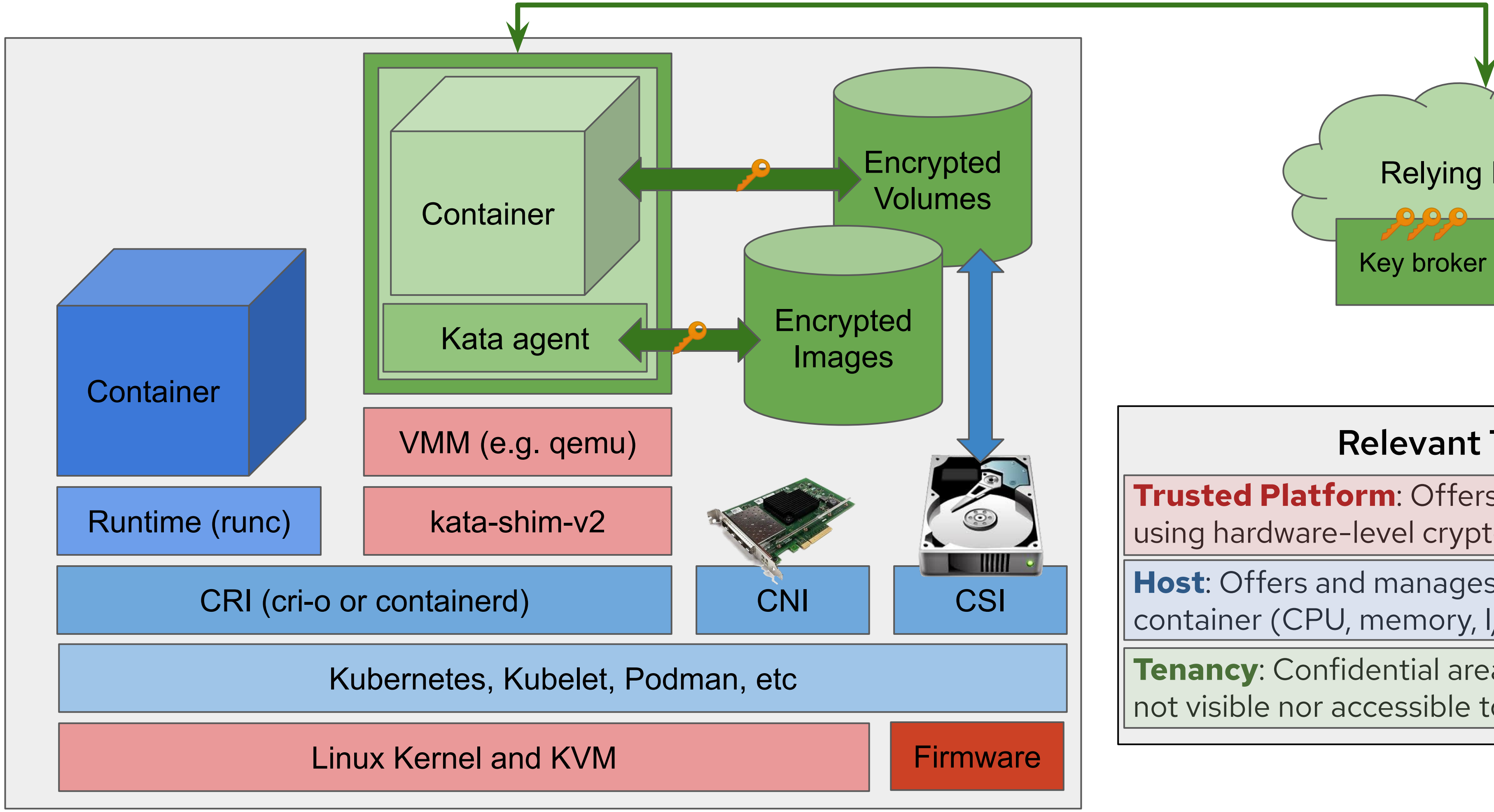
Many vendor-specific technologies

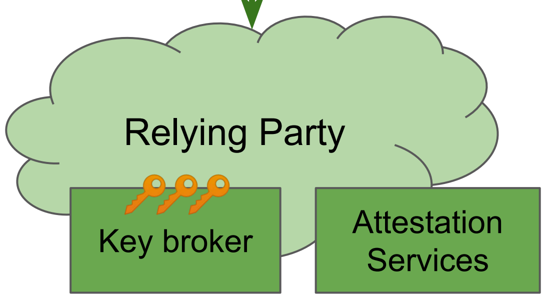
- **AMD** offers Secure Encrypted Virtualization (SEV)
  - SEV-ES adds Encrypted State (e.g. CPU register file)
  - SEV-SNP adds Secure Nested Pages (integrity protection for memory and more)
- **Intel** offers Trusted Domain Extensions (TDX)
- **IBM S390** offers Secure Execution (SE)
- **Power** offers Protected Execution Facility (PEF)
- **Arm** announced Confidential Computing Architecture (CCA)

- All these technologies are based on **virtualization**
- Each of these technologies works in a slightly[1] different way. There be zombies 🧟‍♂️

[1] For a slightly understated definition of “slightly”

Separate Trust Realms: Platform, Tenant and Host





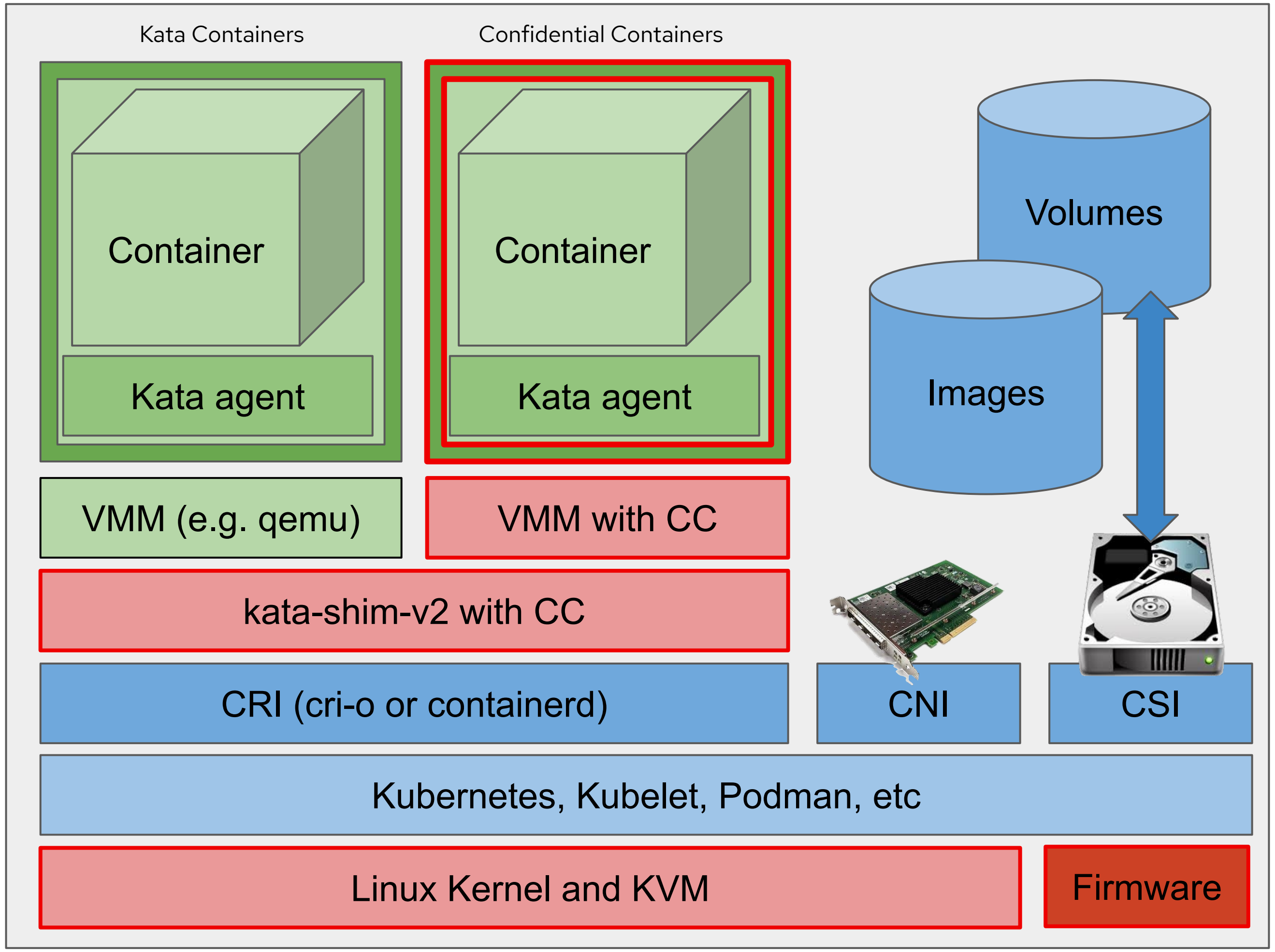
Relevant Trust Realms

Trusted Platform: Offers confidentiality guarantees using hardware-level cryptographic enforcement.

Host: Offers and manages the resources used to run the container (CPU, memory, I/O, etc)

Tenancy: Confidential area carved out of the host, but not visible nor accessible to it.

Enabling Confidential Computing for Kata Pods



Impacted components:

Kata runtime: Pass right options to VMM

VMM: Enable encryption, etc, when setting up VM

Kernel: Low-level hardware support, e.g. SEV, TDX

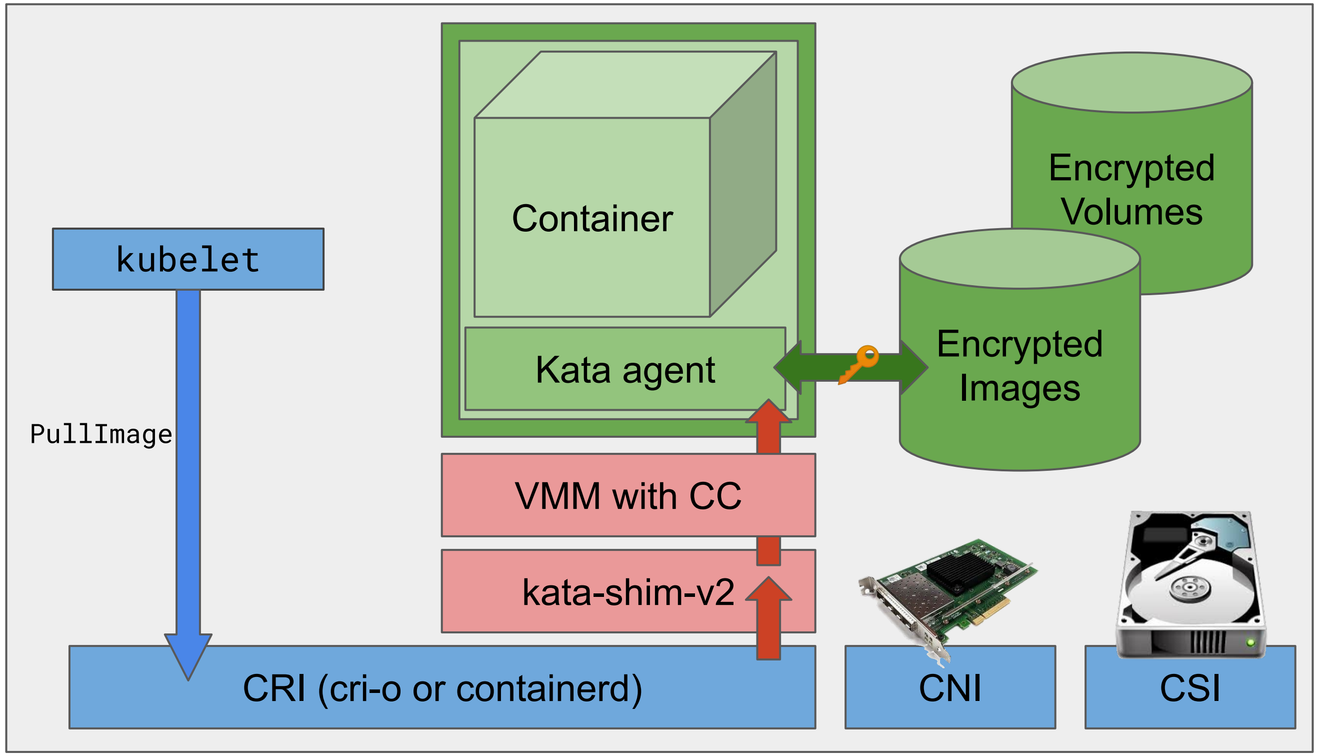
Firmware: Special services, e.g. page validation

Hardware: Encryption in memory controller

The Kata development is done for most platforms

Securing Image Download

- **Pull Image** from *inside* the guest instead of pulling it from the host
- **Store Images** on an encrypted volume, where only guest has decryption keys



Pass `PullImage` API to guest

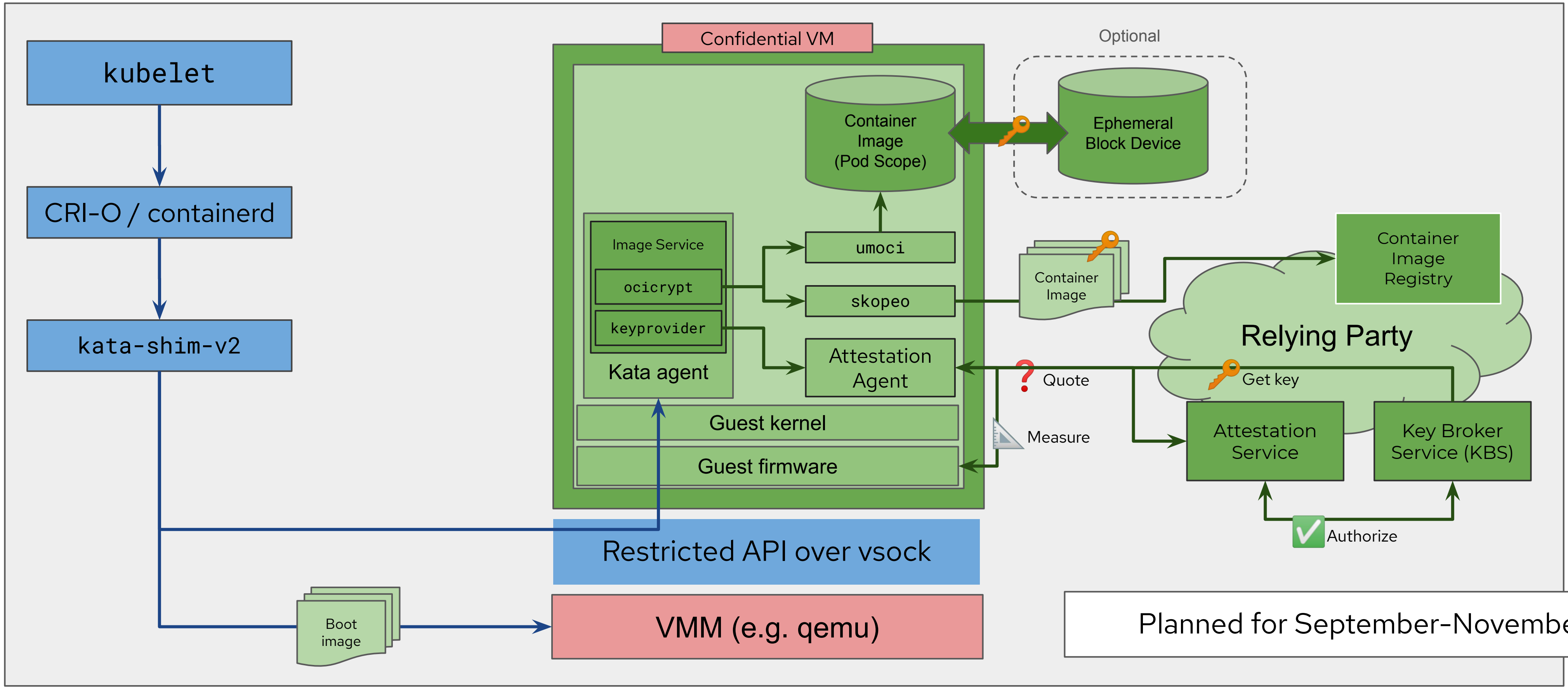
The Kubelet delegates the `PullImage` operation to the `ImageService` in the CRI.

Today, that API does not exist between CRI and `kata-shim-v2`, since container images are currently pulled on the host.

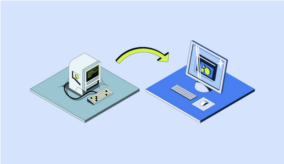
This API situation is relatively typical of the sort of issues we run into for this project.

Also, for initial prototyping, the key has to be pulled out of some magic hat.

Attestation process



Planned for September-November 2021

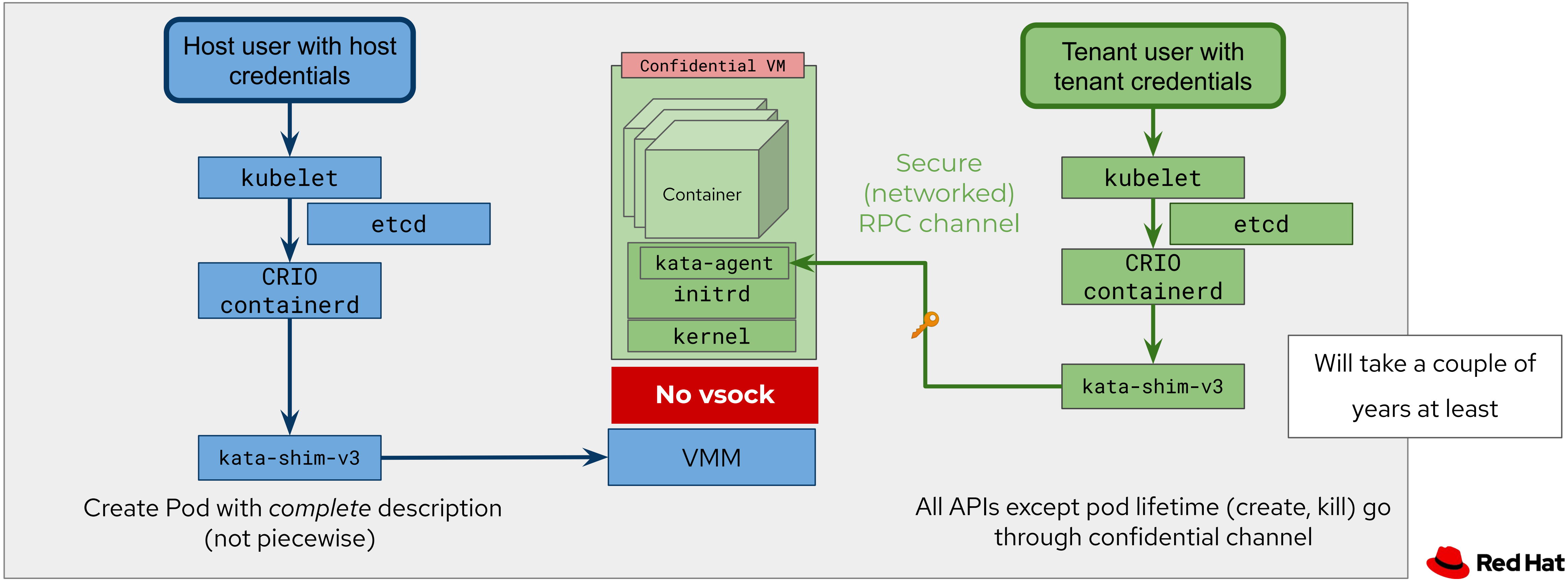


How do you configure your virtual machine?

- **Hot Plugging** is currently used to add memory, CPU or devices to the pod
  - The Pod APIs do not give us the information about container sizes
  - Resources are dynamically added at container creation time
  - This adds a lot of complexity to the runtime, and inefficiency (e.g. fat page tables)
- **Integrity** is hard to guarantee if you can change the configuration at runtime
  - Memory hot-plugging or ballooning mechanisms conflict with encryption / validation
  - Devices, notably pass-through PCI devices with DMA, are also problematic
- **Immutable Pods** are fully defined ahead of time, before booting the virtual machine
  - This requires many changes in the existing Kubernetes APIs
  - Existing APIs may put things “in the wrong place”, e.g. send logs to the *host*.
  - This will simplify and optimize the non-confidential case, e.g. remove hot-plugging

The need for a shadow control plane

- **Tenants** need their own *isolated* administrative realm (logs, container metrics, …)
- **Hosts** manage physical resources (pod creation/destruction, raw disks, H/W metrics…)



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat