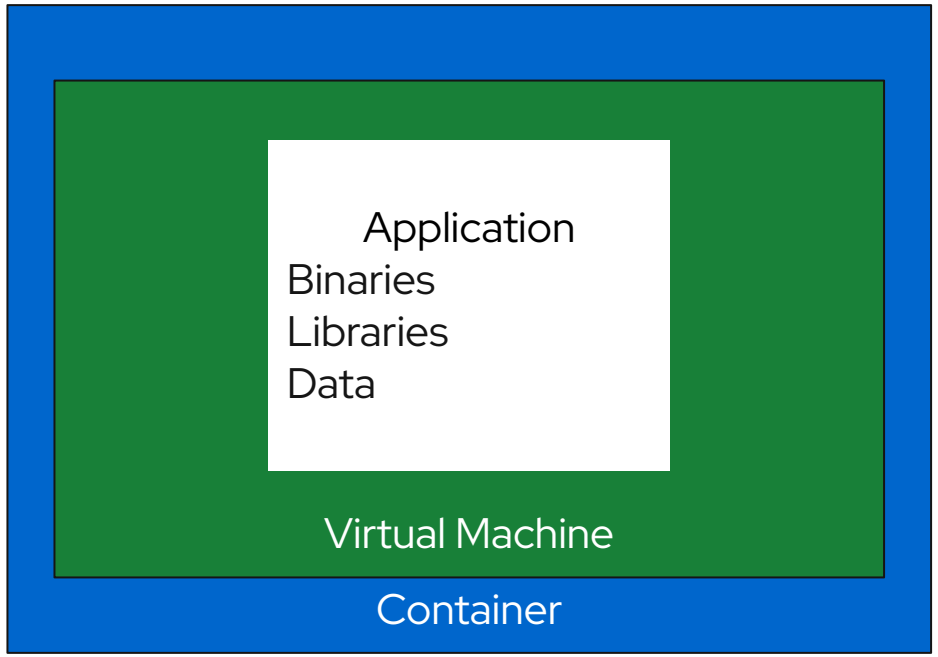
Confidential Containers
with the krun Runtime

Tyler Fanelli, Red Hat

Red Hat

What is `krun`?

- Extension of `crun` runtime with added virtualization capabilities.
- Allows containers to run inside micro-VMs, a machine type optimized for boot time and footprint.
  - Much smaller footprint than standard VMs, as they are only providing a “thin layer of virtualization” wrapping a container’s workload, with a minimal amount of devices.

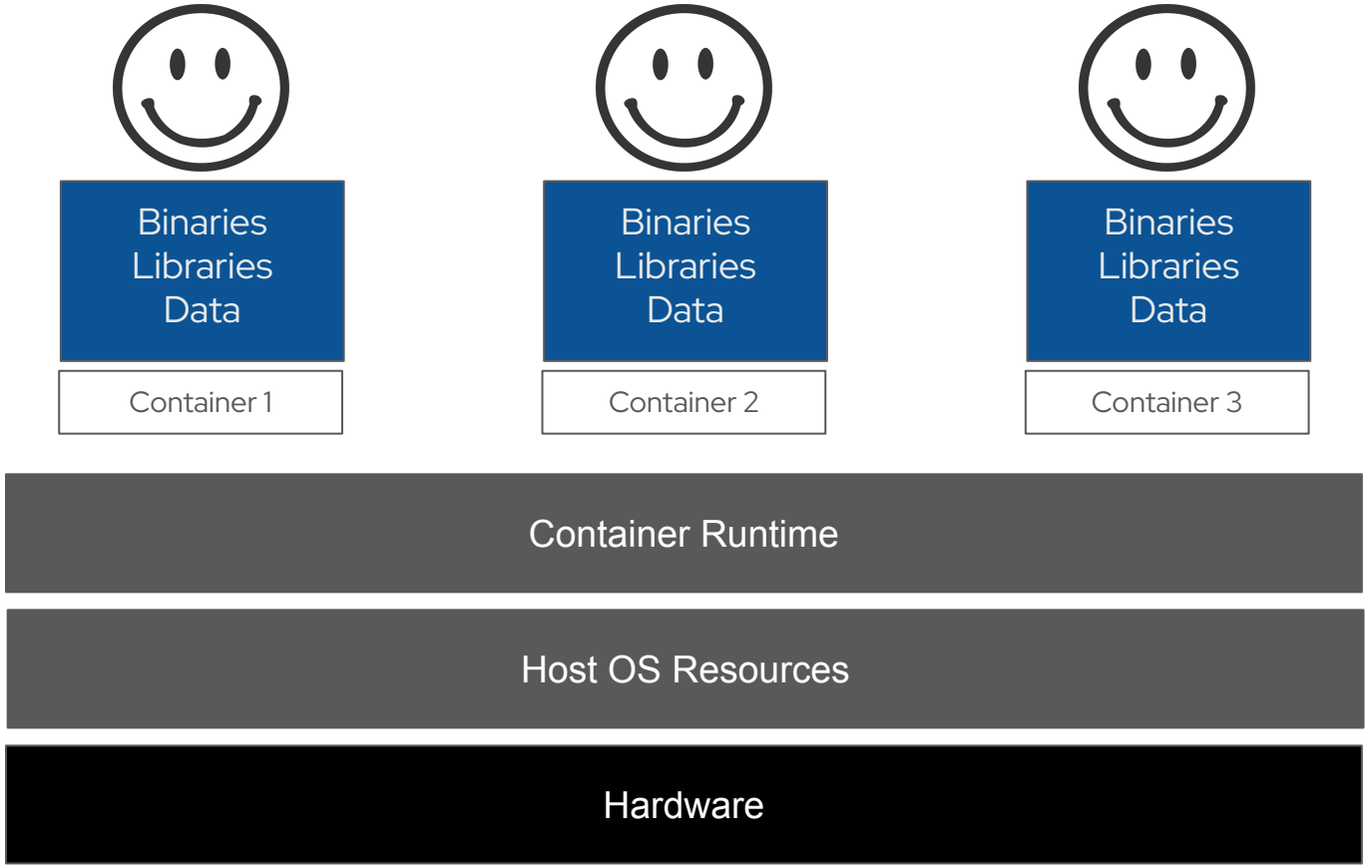


- **Why would we want virtualization for containers?**

A scenario:

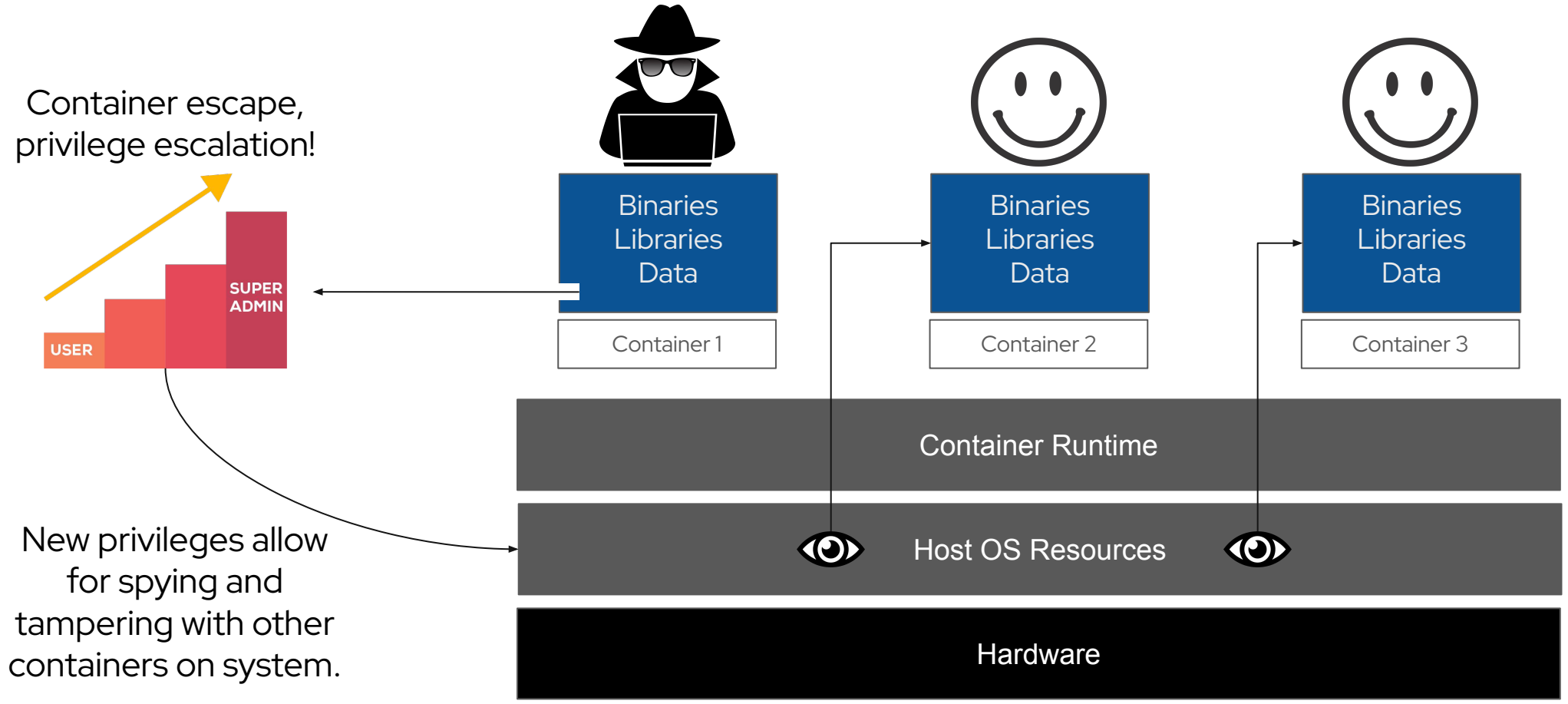
Three containers running on a host, sharing resources and completely unaware of each other.

Typical workloads, nothing out of the ordinary going on.



Red Hat

A scenario:



Red Hat

How can we further isolate containers from each other, and even the host itself?

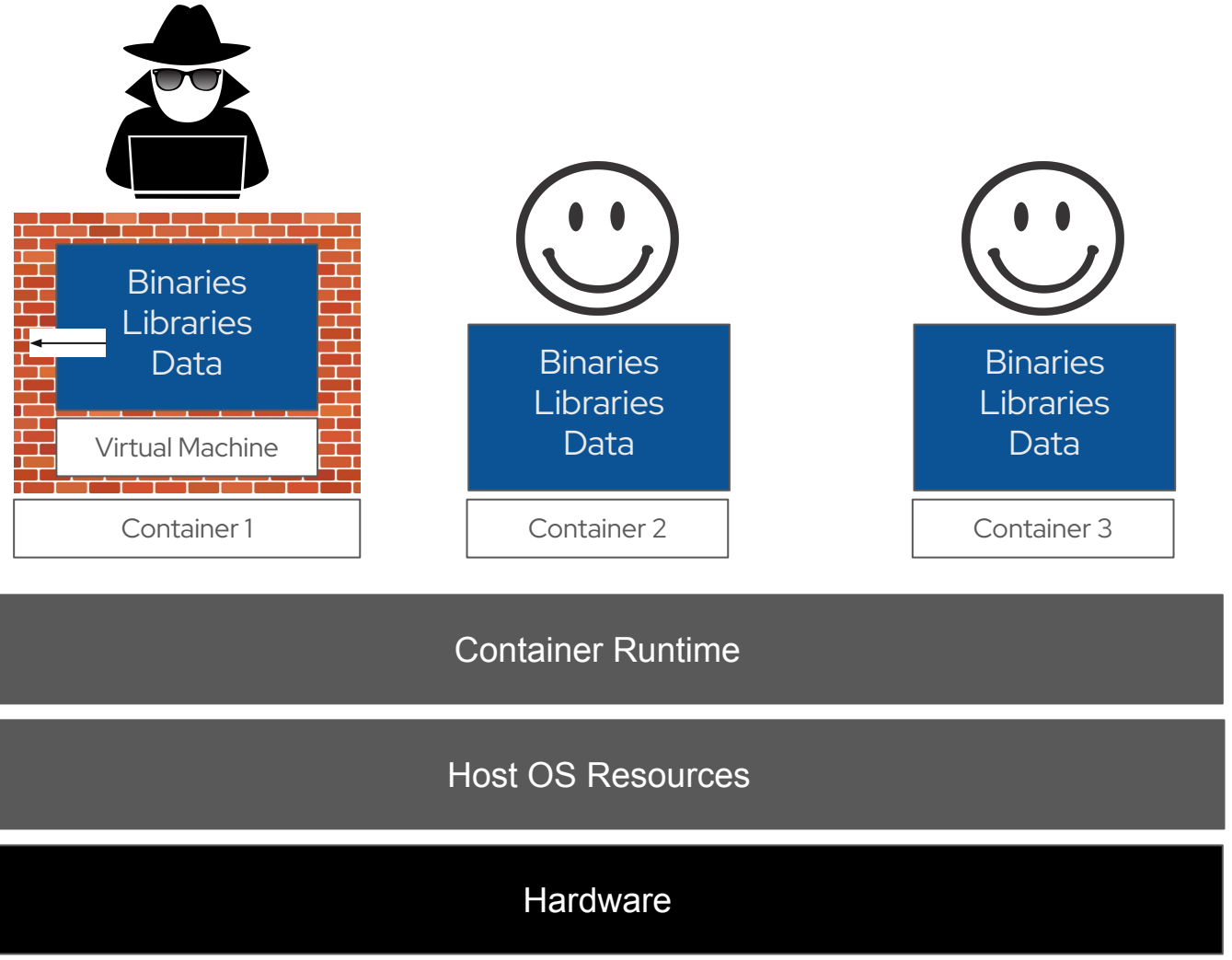
Previous scenario with VM isolation

With VM isolation of containers, common container escape exploits will not affect the host system.

Process may escape from container with escalated privileges, but is still sandboxed in VM.

VM escape possible, but much more difficult.

Isolating applications that are potentially malicious could greatly increase the security of the host system, as well as other containers running on it.



Red Hat

VM isolation of Containers

- Extra layer of security for running containers that are potentially buggy or malicious.
- Attacks such container privilege escalation mitigated completely with extra layer of isolation.
- Escaping VM much more difficult in practice.
- With KVM acceleration, no performance degradation in using VMs.

How can we leverage existing container projects to build VM-isolated containers?

Component: crun

- Fast, low memory footprint OCI runtime.
- Responsible for running the containerized processes on the host.
- **Written in C**.

crun

Component: libkrun

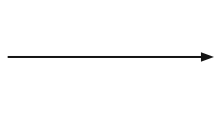
- **C API** to run processes in a KVM virtual machine.
- Integrates a hypervisor solely for the code you are running virtualized. That is, the hypervisor only manages the libkrun VM you create.
- **libkrun VMs are micro-VMs**, they run processes virtualized with the minimal amount of emulated devices for a much lower memory footprint than standard VMs.
- Abstracts most of the complexity that comes from VM management.

```
#include <libkrun.h>

int32_t krun_create_ctx();

. . .

int32_t krun_set_vm_config(uint32_t ctx_id,
     uint8_t num_vcpus,
     uint32_t ram_mib);
. . .

int32_t krun_set_root_disk(uint32_t ctx_id,
                     const char *disk_path);

. . .

int32_t krun_add_virtiofs(uint32_t ctx_id,
                          const char *c_tag,
                          const char *c_path);

. . .

int32_t krun_set_exec(uint32_t ctx_id,
                      const char *exec_path,
                      char *const argv[],
                      char *const envp[]);

. . .

int32_t krun_start_enter(uint32_t ctx_id);
```

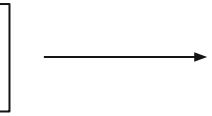
crun + libkrun = krun

- crun runtime, using the libkrun C API to virtualize the container's contents.
- Virtual machine inside of a container.
- Container orchestration software communicates with crun, crun uses libkrun API to manage virtual machine.

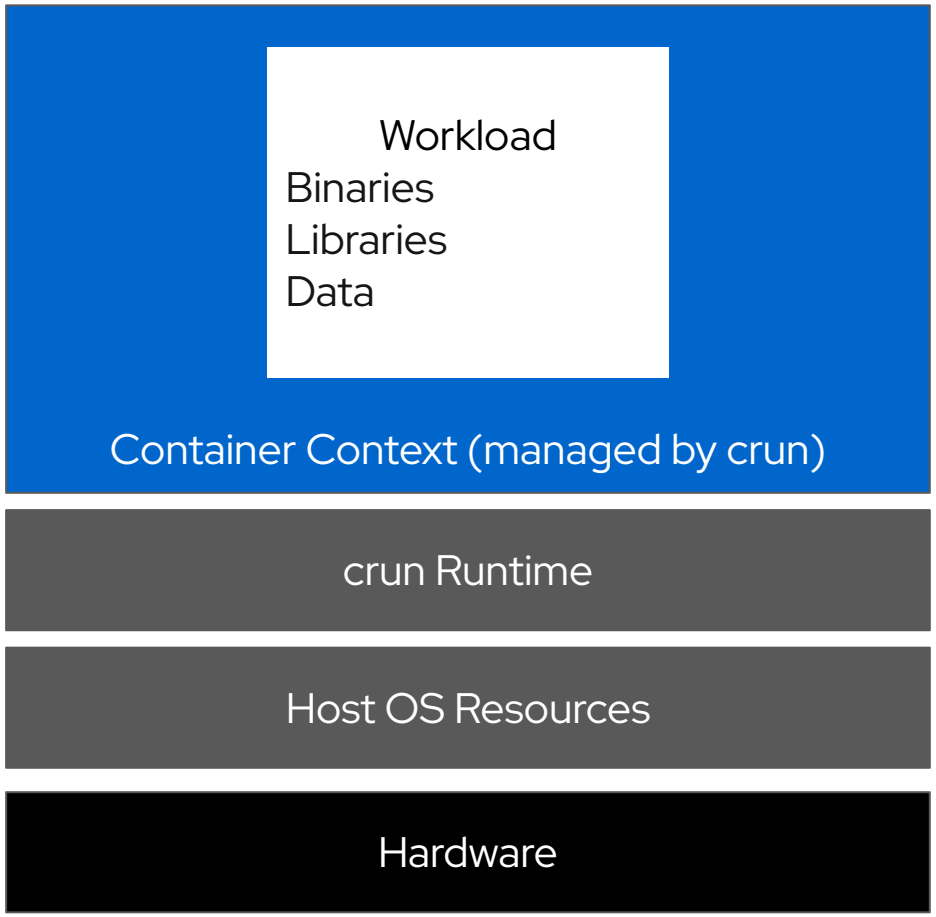
crun



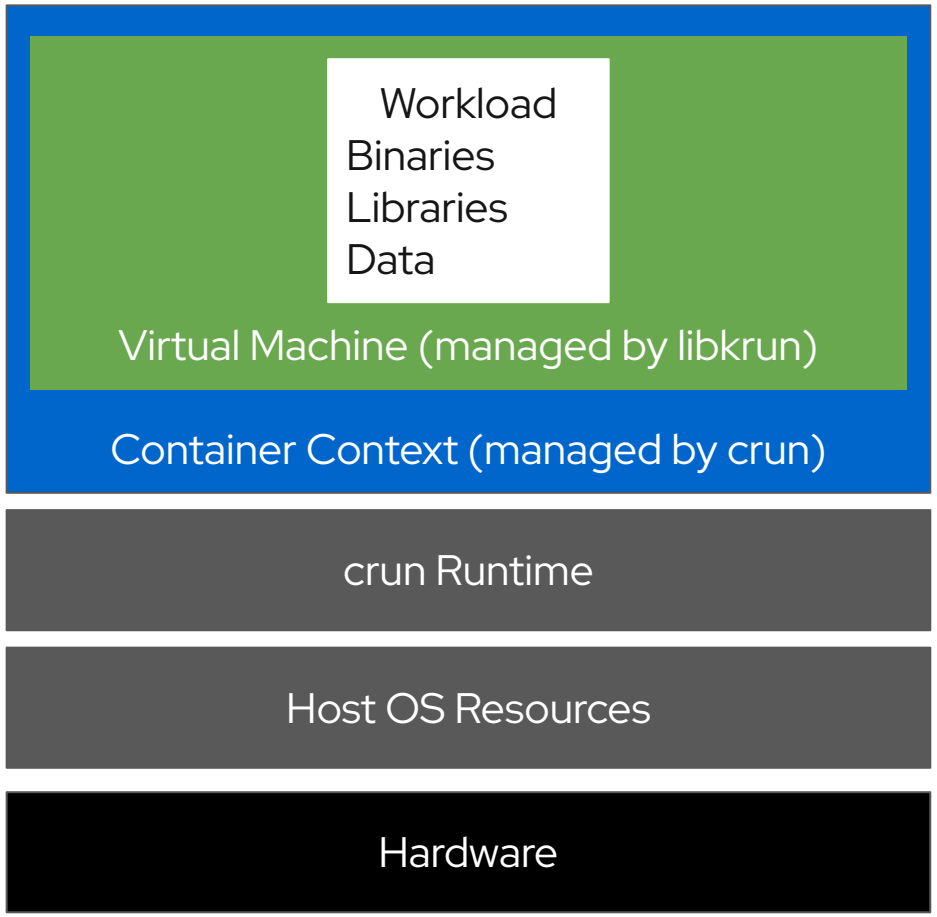
```
#include <libkrun.h>
```



KVM Virtualized Container



crun
Container



krun
Container

Red Hat

krun Tradeoffs

+ VM isolation for potentially malicious/buggy workloads

+ Near-zero performance decrease

+ Ability to take advantage of CPU Trusted Execution Environments for **Confidential Computing**

– Sharing resources with processes and other containers outside of the krun VM is more difficult.

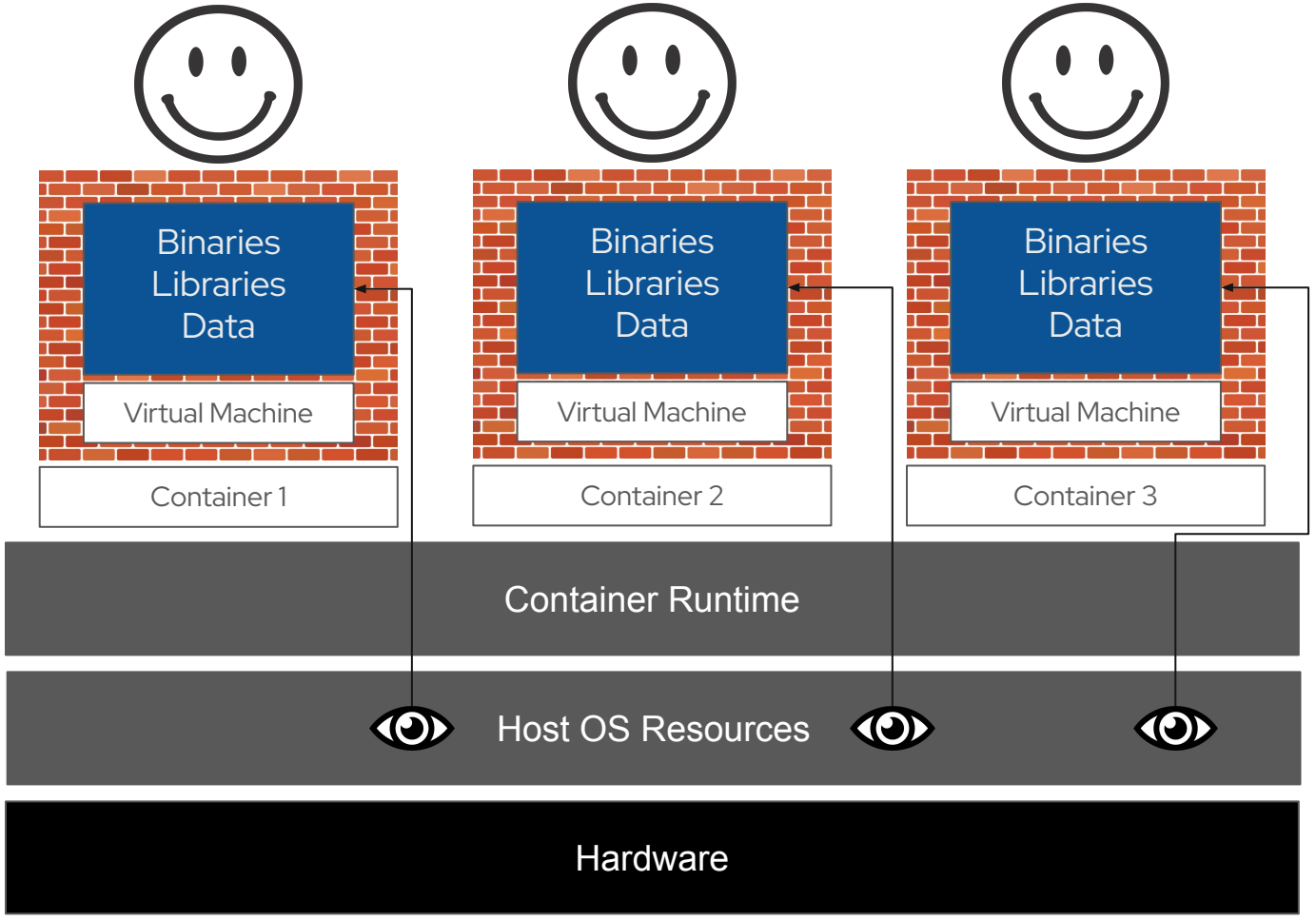
Previous scenario with VM isolation

No matter if running containers natively or with VM boundaries, host system administrator can read or tamper with guest memory.

With these privileges, a rogue administrator has the ability to spy on applications and copy data from them.

With workloads handling sensitive information (bank or medical records, trade secrets, etc...), sophisticated attacks can gain access to this information.

Mitigation of these types of attacks is desired.



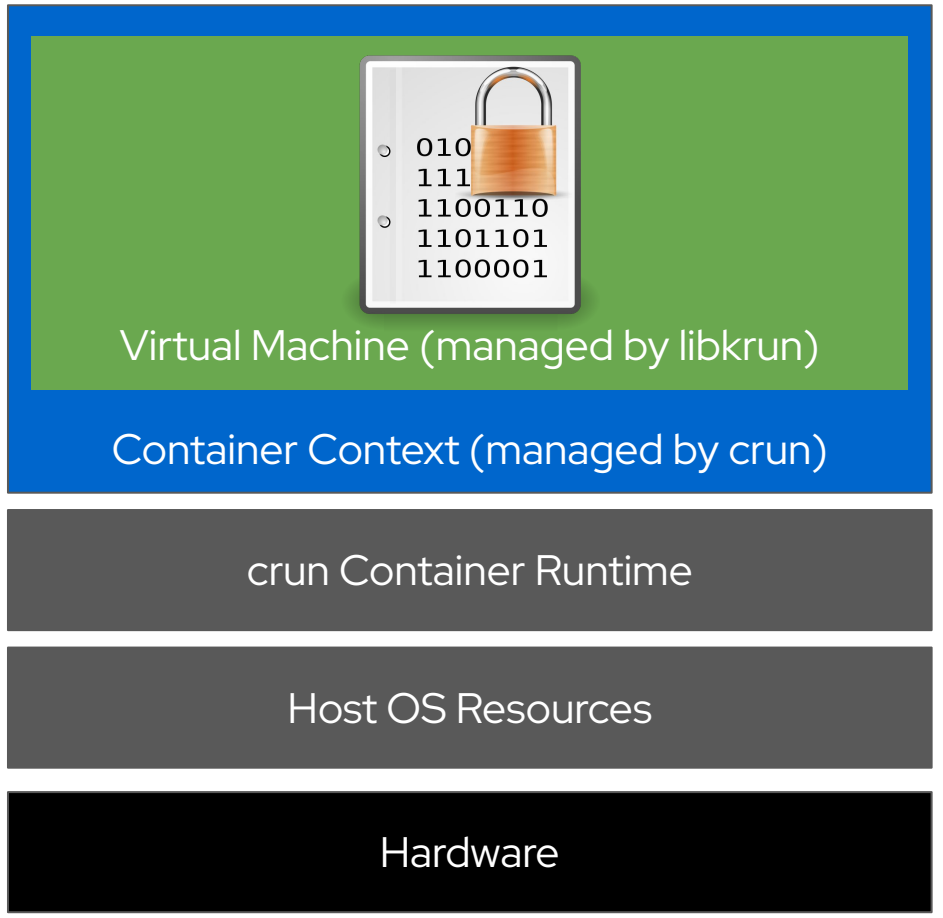
Red Hat

Confidential Computing

- Virtualization technology allowing a VM guest's RAM to be encrypted.
  - VM guest known as a **Trusted Execution Environment** (TEE).
- Not even host hypervisor can view encrypted guest memory.
- Encryption/decryption keys stored on secure processor managed by hardware alone.
- Focused on data in-use (RAM + CPU registers) rather than data at-rest (disk storage).
- Gives cloud/edge users the ability to run their applications in untrustworthy environments with extra protection from data breaches.
  - Protection from other users of the system as well as host system administrators.

krun Confidential Computing

- libkrun VMM packages the kernel, firmware, initrd needed to run confidential workloads.
- Each VM region (including RAM for the application itself) is measured and encrypted to ensure all components of the VM are run confidentially.
- VM unable to immediately run application when beginning to execute.
  - Root disk containing application code and data is encrypted.
  - VM must **attest** itself to unlock the root disk and begin running the application.



krun
Confidential
Container

Red Hat

We don’t want to run our application if we’re not confidential. How can we *prove* that we’re running confidentially?

VM Attestation

- We're told our application is confidentially running on TEE hardware, but how can we be sure?
- Must attest, or cryptographically prove that our application is running confidentially.
- Two main components of attestation:
  - **Hardware**: Verify that we're running on authentic TEE hardware from chip supplier.
    - Prevents the host from "lying" about our application running confidentially.
  - **Software**: Verify that our entire application environment (and ONLY our application environment) is included in secure enclave (the VM)
    - Prevents the host from potentially mapping some extra unencrypted pages in our enclave to snoop on its activity (e.g. a GDB server to spy on a guest).

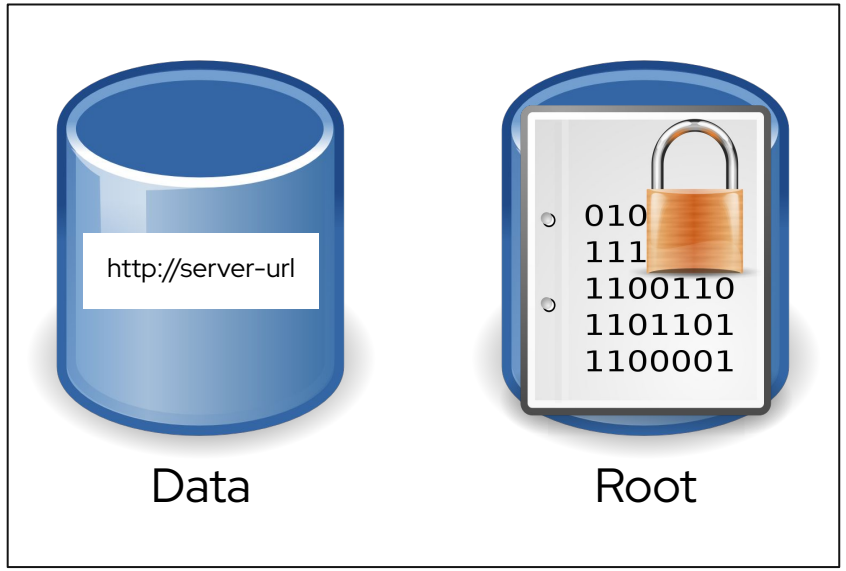
- **How does krun attest its VMs?**

Component: buildah

- Tool to build OCI container images.
- `buildah build`, the main command to build OCI images, recently introduced the `--cw` (confidential workload) flag to build OCI images with krun VM attestation in mind
- `--cw` registers build measurements with an attestation server for eventual runtime attestation from the krun VM.
- Confidential workload OCI images much different than standard OCI images.

buildah --cw Images

- Confidential workload images contain two main disks expected by the VM.
  - Root disk
    - Contains application code/data.
    - LUKS encrypted, with key stored in attestation server (more later).
  - Data disk
    - Contains data needed for attestation, such as the URL of the remote attestation server needing to communicate with.
- Since root disk (containing workload) cannot be ran until unlocked, attestation is required for all workloads.

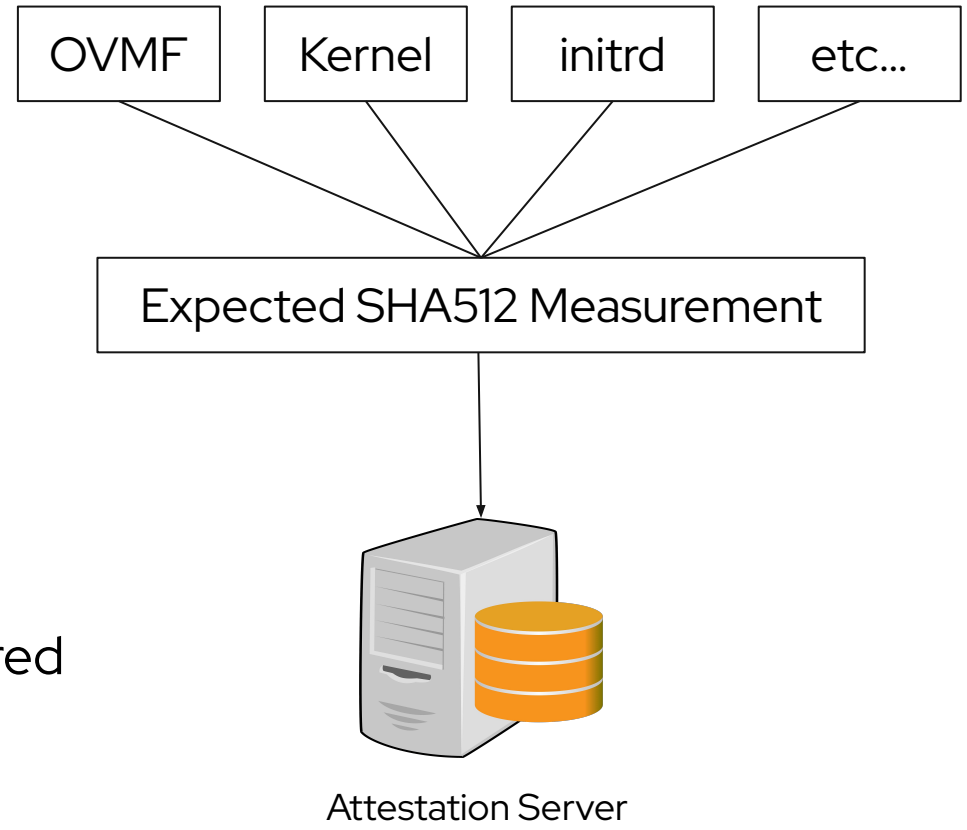


buildah --cw Image

Red Hat

buildah --cw Registration

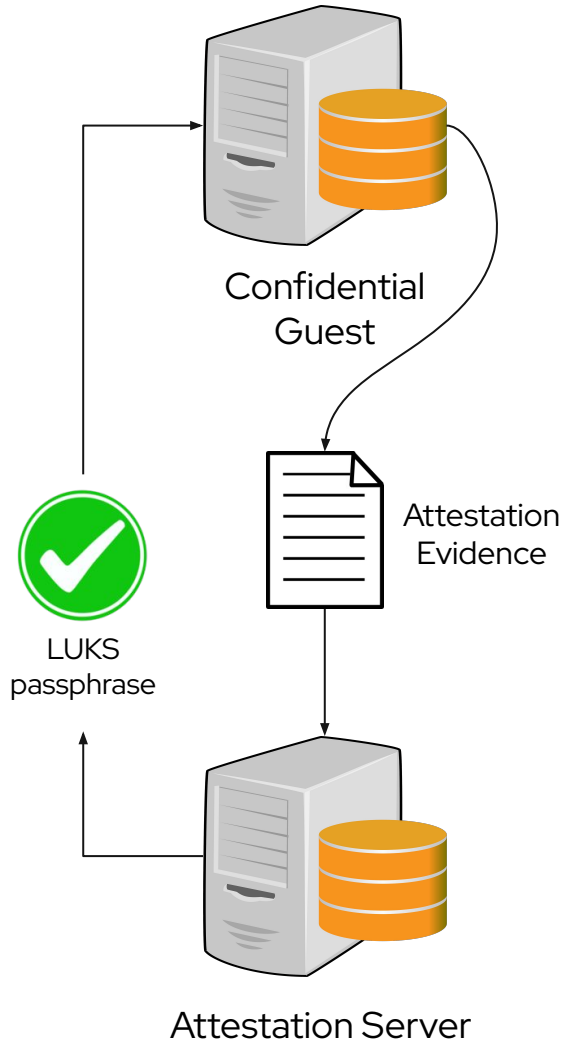
- Buildah uses `libkrunfw` package to fetch each component that will be used to run workload.
  - OVMF, kernel, initrd, etc..
- Measures each section and computes a hash of the contents.
- Hash is registered with the attestation server as the **expected launch measurement**.
- When krun VM eventually attests, the launch measurement from its TEE evidence will be compared with expected measurement (more later).
- URL of server is inserted into data disk.



Red Hat

krun Attestation

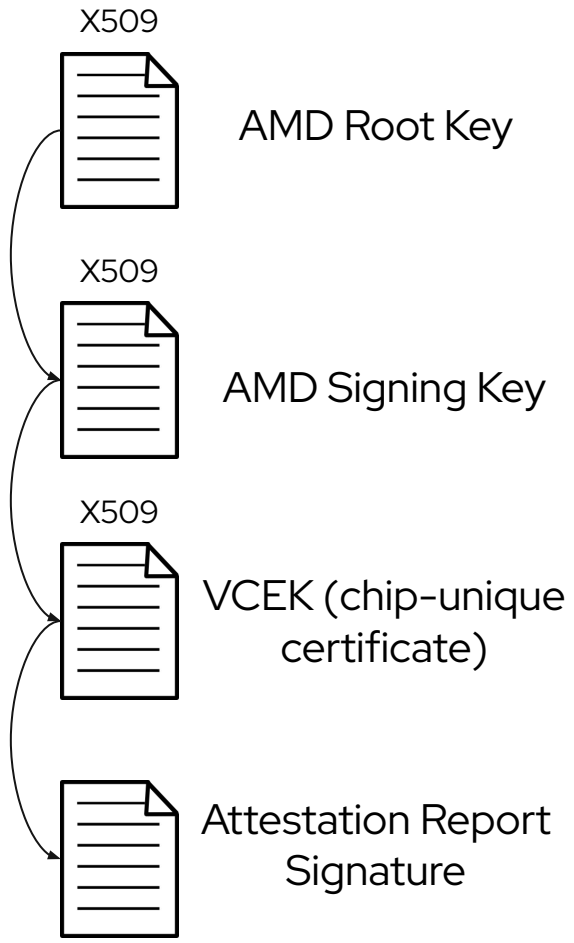
- Each TEE architecture allows guests to request an attestation report from the secure processor.
  - Launch measurements, identification information, etc...
- krun fetches and sends report to the attestation URL (read from data disk).
- Attestation server completes attestation and reports results.
- If successful, server sends krun VM the LUKS passphrase to unlock the root disk and begin running the application.



Red Hat

Attestation Server: Hardware Validation

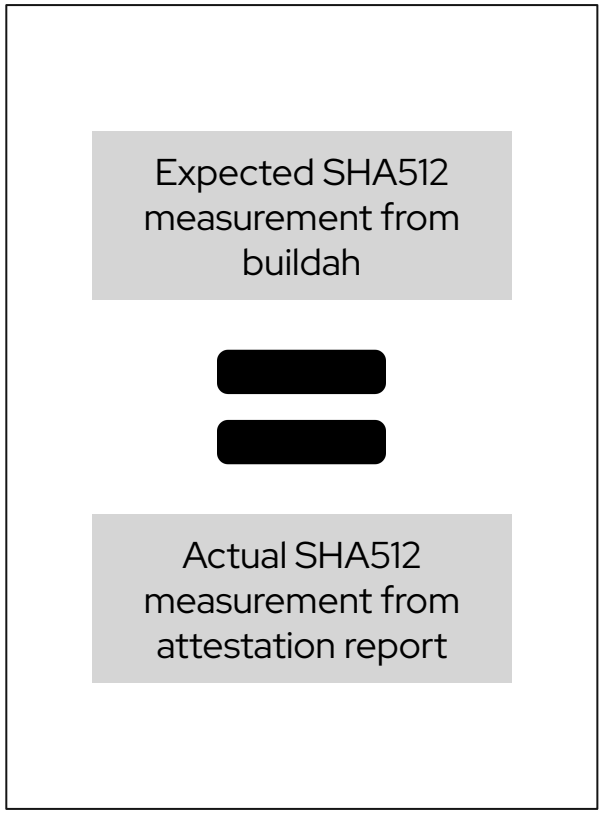
- Verify that we're running on authentic TEE hardware from chip supplier.
- Each attestation report contains a signature that can be traced back to the chip supplier's root of trust.
  - SEV-SNP given as example. Certificate chain traced back to AMD root of trust.
- Cryptographically proves that the attestation report is from an authentic TEE processor.
- Host cannot lie about running confidentially.



Red Hat

Attestation Server: Software Validation

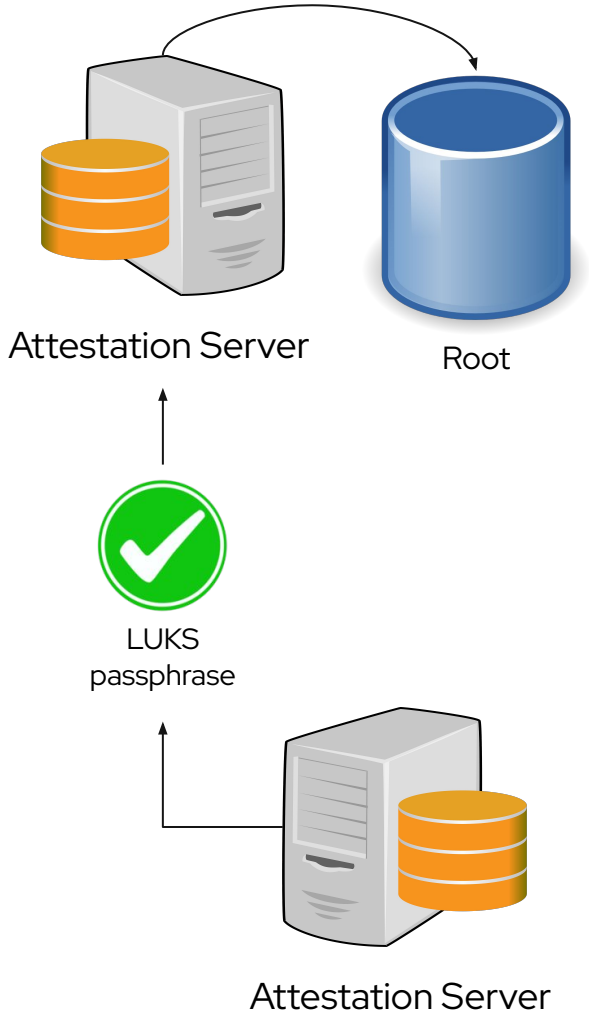
- Recall that `buildah build --cw` previously registered an expected launch measurement with the attestation server.
- The (now authenticated) attestation report contains a hash of all components encrypted by the secure processor (i.e. the actual launch measurement)
- Must compare the expected launch measurement with the actual launch measurement.
- If not equal, either:
  - Not all VM pages were encrypted by secure processor.
  - Extra pages were mapped into VM guest memory.
- Either option may jeopardize the confidentiality of the workload, so attestation fails if expected and actual launch measurements do not match.



Red Hat

krun VM: Successful Attestation

- On successful attestation, attestation server returns LUKS passphrase to VM's root disk.
- VM uses LUKS passphrase to unlock root disk and begin running application.
- Cryptographically proven that our workload (and only our workload) is running confidentially on the host system.



Red Hat

SEV-SNP
Demo!

Status as of this talk

- AMD SEV-SNP fully supported with attested workloads.
- ARM CCA a focus right now, with development underway for Intel TDX.
- Preparing an initial release for `keybroker`, the remote attestation server built with krun VM attestation in mind.
    - Would very much like to present and discuss `keybroker` with the Attestation SIG if given an opportunity.

Questions?

Thanks!