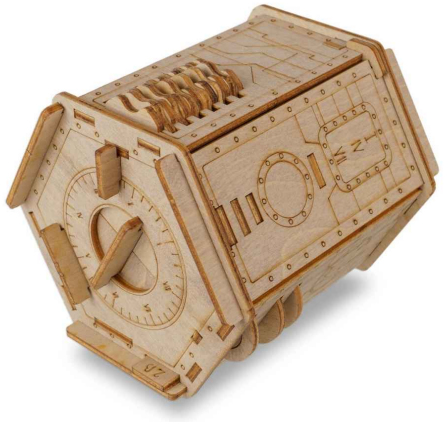
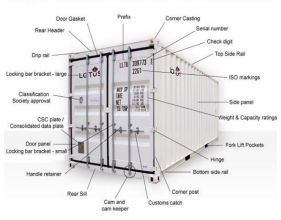
Improving container security by adopting sandboxed runtimes



Iliyan Petkov
Senior Cloud Native Engineer, RX-M, LLC

Objectives

Containers ?



Container anatomy

Runtime security challenges

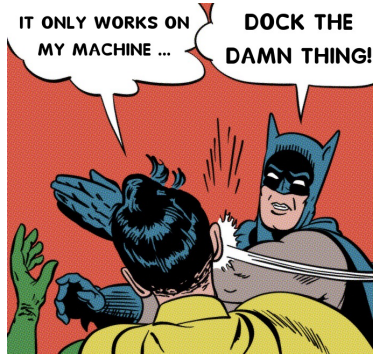
Container runtimes

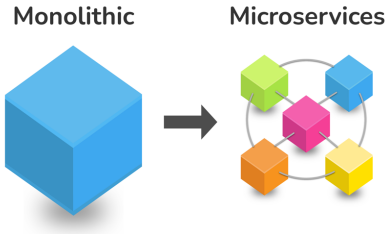
Securing container workloads

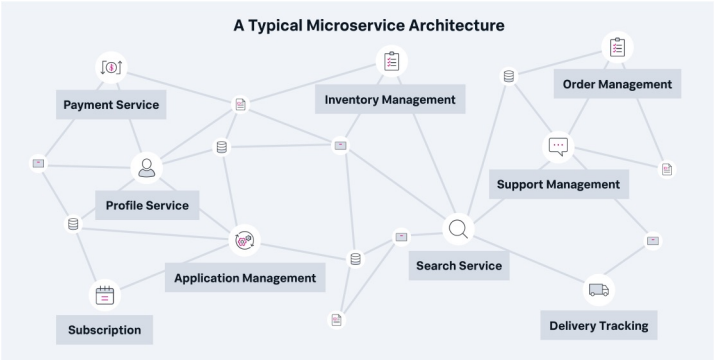
Containers ?

What containers do ?

- Reduce complexity through abstractions
- Well fit for distributed computing
- Good at automation
- "Feels" like VMs (they are not)
- Introduce minimum resource overhead
- Provide some level of isolation

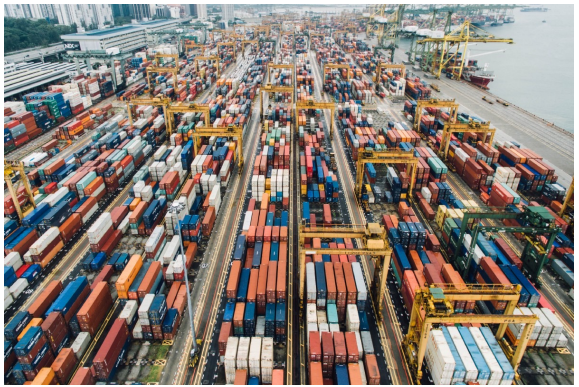






Some concerns

- Not right for all tasks
- Weaker (than VM) isolation
- Prone to sprawl
- Security and governance are platform-specific
- Harder to monitor and troubleshoot distributed applications



Containers are standartized!

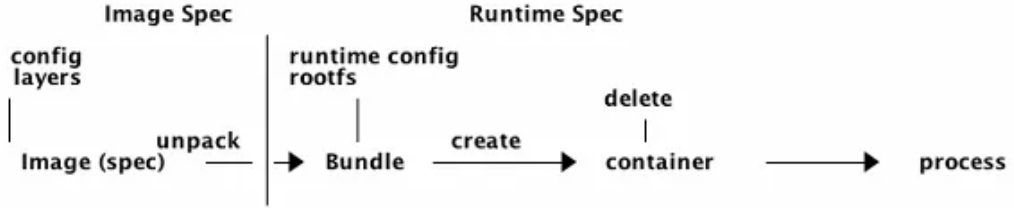
OCI – industry collaborative effort to define open container specification(s)

It consists of three specs:

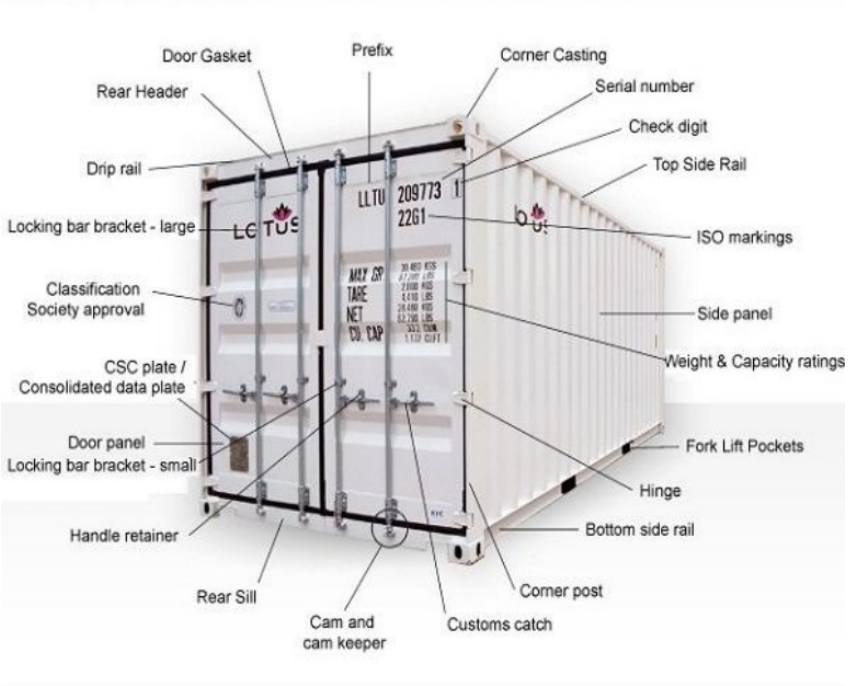
- Image specifications
- Runtime specification
- Distribution specification

OPEN CONTAINER INITIATIVE

```
| - index.json
| - oci-layout
| - blobs
    | - sha256
        | - 3588d025422      (image.manifest)
        | - 4b0bc1c4050      (image.config)
```



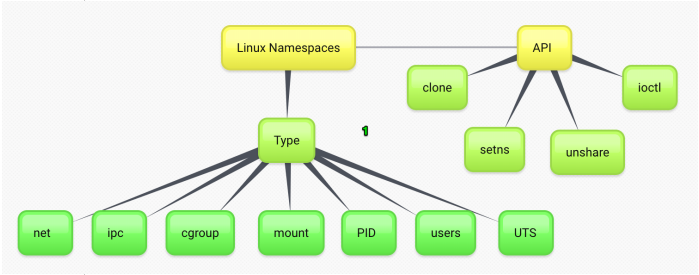
Container anatomy

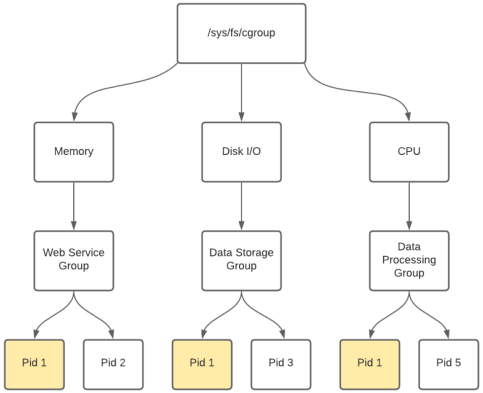


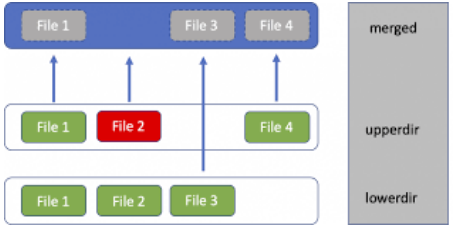

THERE IS NO CONTAINER
IS JUST ANOTHER KERNEL PROCESS

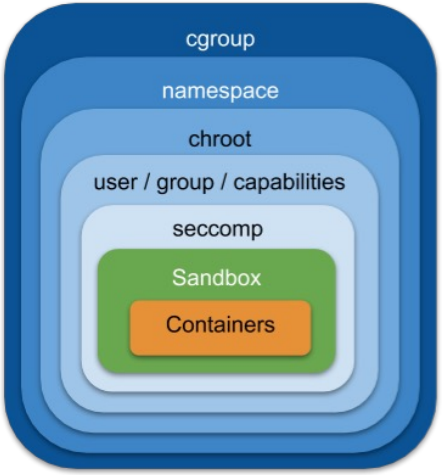
What is actually a container?

- A process that runs on the host system
- Namespaces - limits what a process can see
- Cgroups – metering and limiting on the processes
  Cpu (track user/system cpu time, usage per cpu)
  Memory (accounting, limits…)
- Copy on Write filesystem



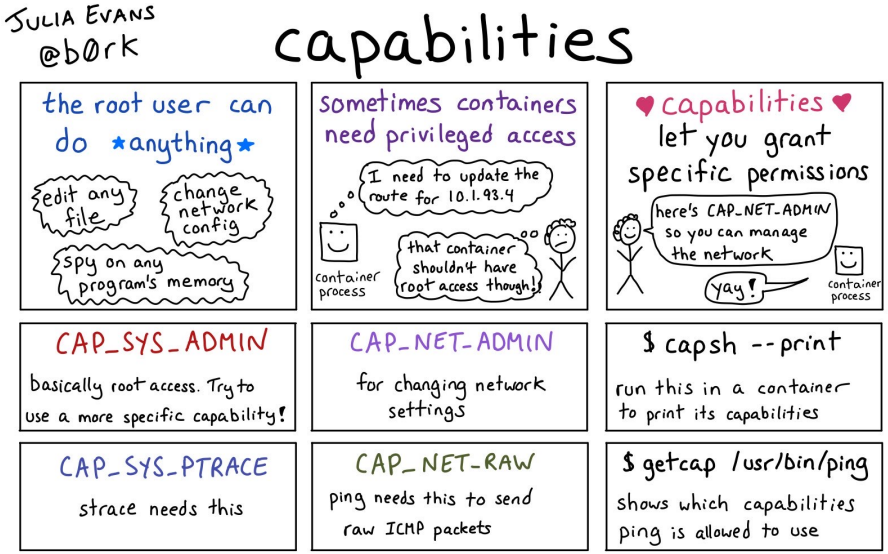






Container privileges

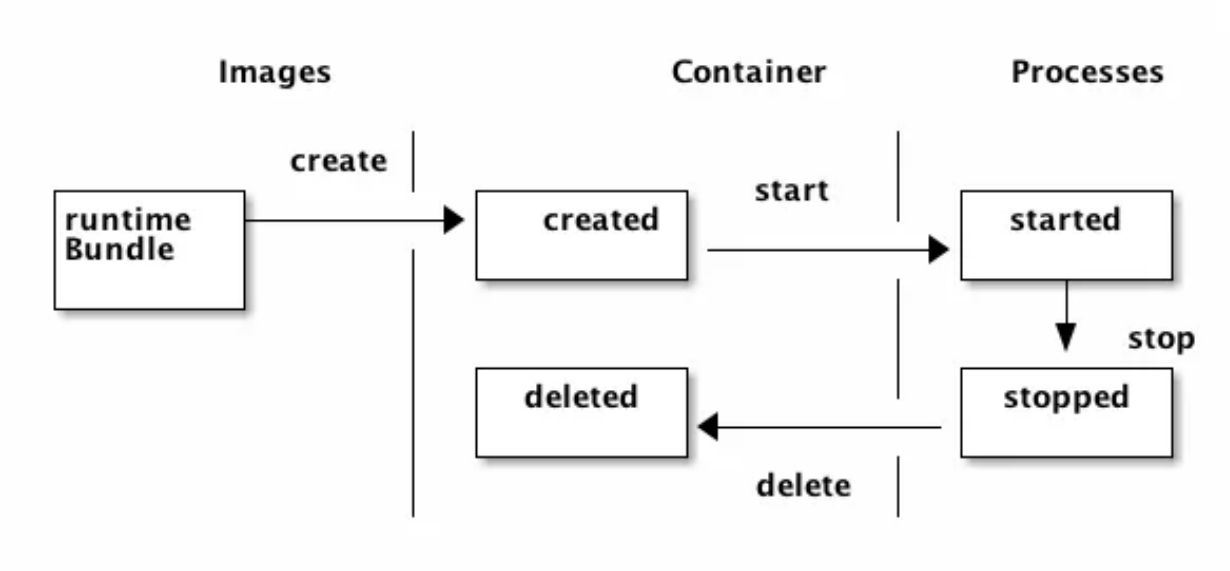
- The container runtime assigns a set of default privileges (capabilities) to the container.
- This default set of privileges is, or can be, harmful.
- The best practice is to drop all privileges and only add the ones you need.



21 lines (20 sloc) | 436 Bytes

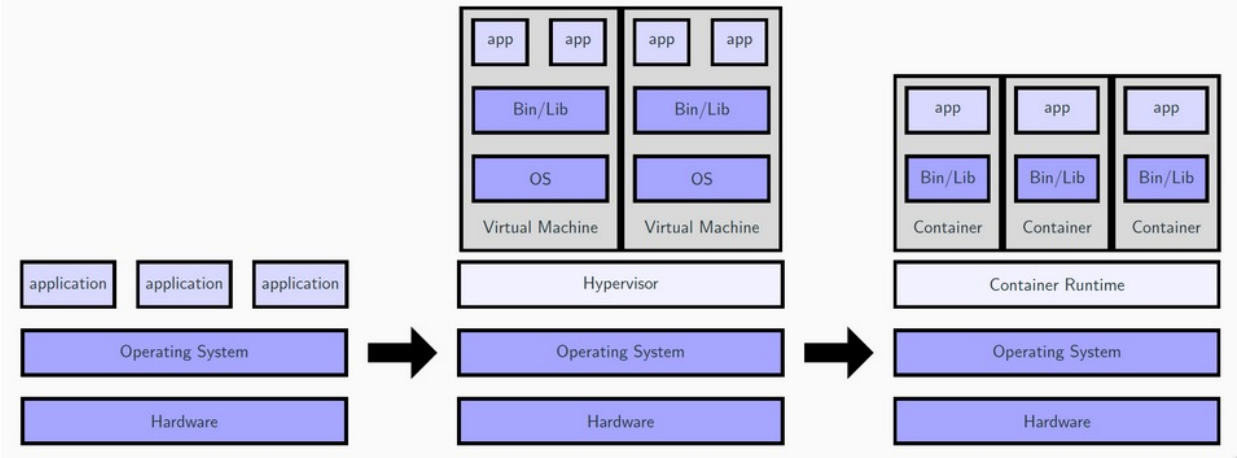
```go
package caps // import "github.com/docker/docker/oci/caps"

// DefaultCapabilities returns a Linux kernel default capabilities
func DefaultCapabilities() []string {
        return []string{
                "CAP_CHOWN",
                "CAP_DAC_OVERRIDE",
                "CAP_FSETID",
                "CAP_FOWNER",
                "CAP_MKNOD",
                "CAP_NET_RAW",
                "CAP_SETGID",
                "CAP_SETUID",
                "CAP_SETFCAP",
                "CAP_SETPCAP",
                "CAP_NET_BIND_SERVICE",
                "CAP_SYS_CHROOT",
                "CAP_KILL",
                "CAP_AUDIT_WRITE",
        }
}
```

Container lifecycle



Containers vs Virtual Machines

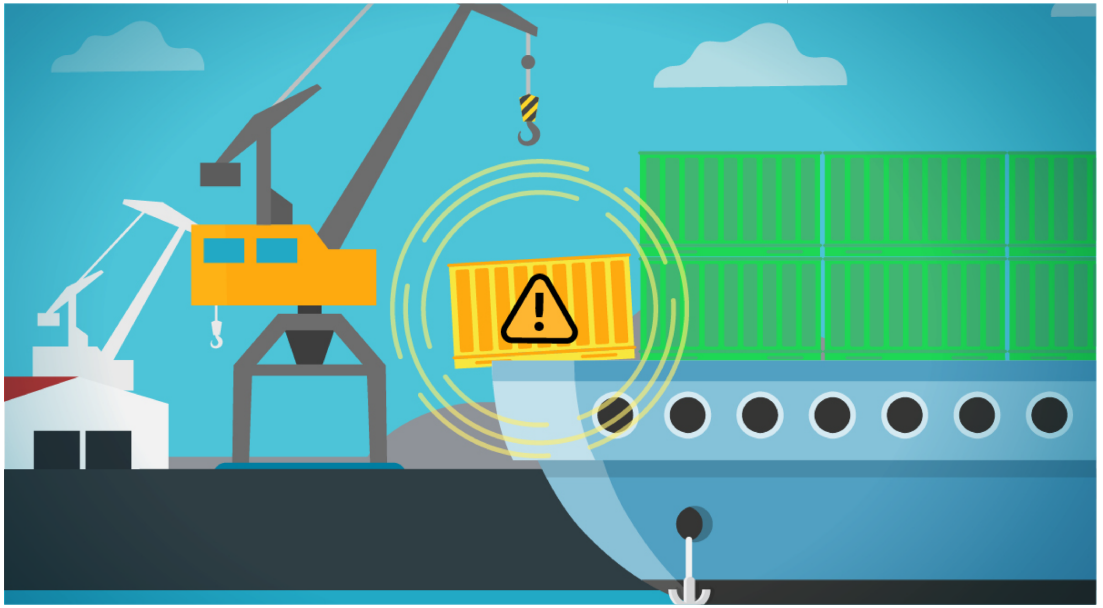
Bare Metal | Virtual Machiens | Containers



- Better isolation
- Virtualization for the hardware

- Quick Start
- Fewer resources
- Security is configuration dependent
- Virtualization for the OS

Runtime security challenges



Threat model

Security objectives:

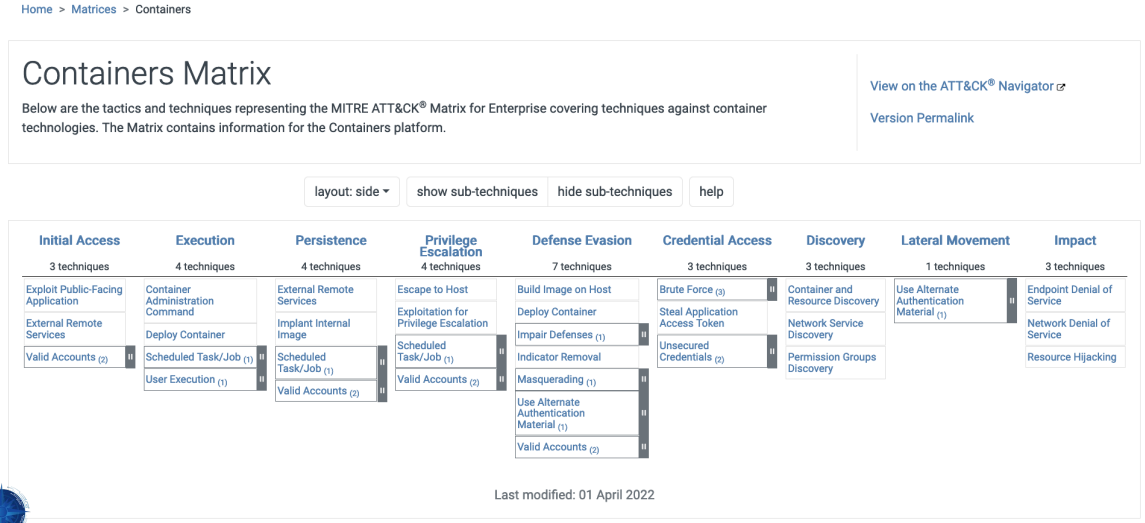
1. **Confidentiality** – Keeping sensitive information private
2. **Integrity** – Consistency of data, networks, and systems
3. **Availability -** "Keeps working in the presence of attack”

CIA Triad

Technically there is a forth one: **non-repudiation** or **accountability**

Threats Modelling

- A process that allows us to identify potential threats (presence of vulnerabilities, lack of safeguards, etc.
- Its purpose is to provide defenders with information about what control or defenses need to be added based on the possible attack vectors, attacker's profiles, etc.



STRIDE – Threat Classification Methodology

Category	Threat Template
Spoofing	Threat action aimed to illegally access and use another user's credentials, such as username and password.
Tampering	Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the Internet.
Repudiation	Threat action aimed to perform illegal operations in a system that lacks the ability to trace the prohibited operations.
Information Disclosure	Threat action to read a file that one was not granted access to, or to read data in transit.
Denial of Service	Threat aimed to deny access to valid users, such as by making a web server temporarily unavailable or unusable.
Elevation of Privilege	Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system.

Attack vectors

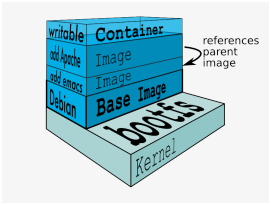
- Attack vectors – interactions between the attacker and the system
- Attack surface – summary of all attack verctors

Common Attack vectors
- Misconfigurations
- System API
  - Opening/Creating combination of files, sockets
  - Passing crafted arguments, structures, packets
  - Creating race conditions with multiple threats
- System ABI
- Side Channels
- Others

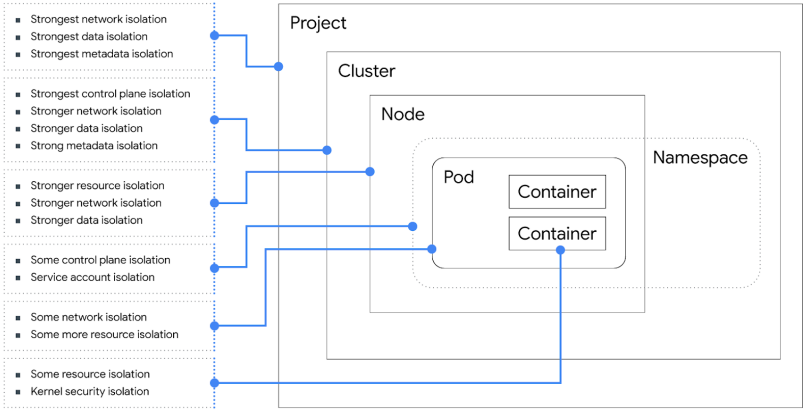
https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

Security landscape

Image registries

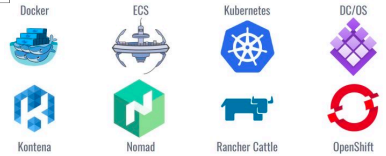


Images



Clusters

Container Runtimes

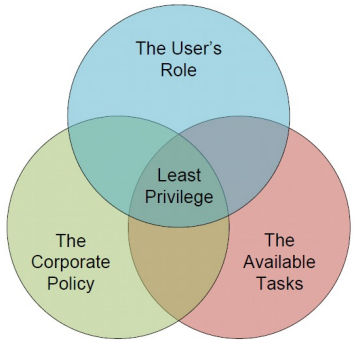


Container orchestrators

Least Priviletes and Zero Trust

Least Priviletes

- Giving the *least* amount of access (privilege) to data, systems, and assets that are needed to get the work done.

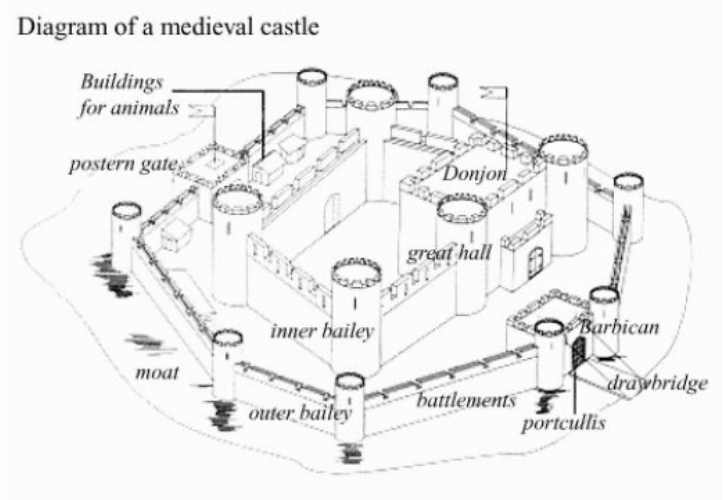


Zero Trust

- No one is trusted by default from inside or outside
- Verification is required prior to accessing resources
- It is far safer to assume that nothing is trustworthy than to assume that preventative security measures have plugged all the holes.

Defence in depth

- Defend a system against any particular attack using several independent methods
- Layering tactic, conceived by the US National Security Agency (NSA) as a comprehensive approach to information and electronic security



Container runtimes

Container runtimes

Container runtime is responsible for:

- Manage container lifecycle – Start, stop, set-up execution environment, configuration, etc.
- Each container runtime has its own strengths

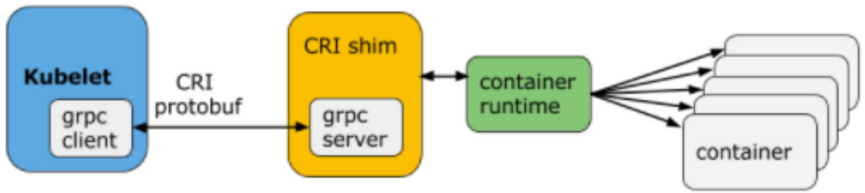
Runtime is not responsible for:

- Running the program itself

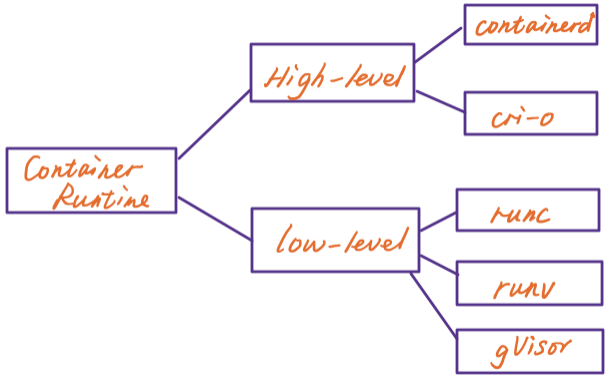
Container Runtime Interface - CRI

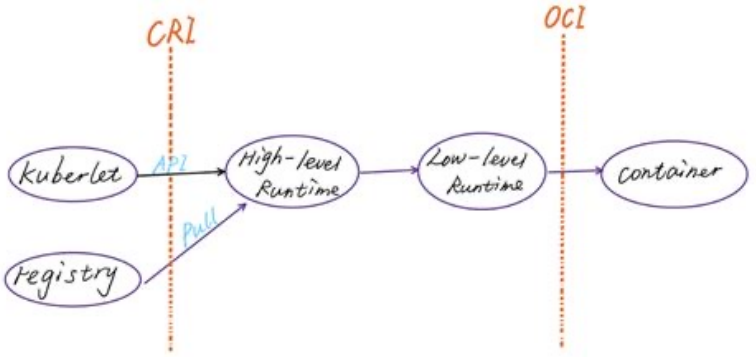
- A Kubernetes plugin interface that enables usage of multiple runtimes
- CRI was introduced with k8s 1.5 in 2016
- Provides a standard API for runtimes to work with K8s

```
service RuntimeService {

    // Sandbox operations.

    rpc RunPodSandbox(RunPodSandboxRequest) returns (RunPodSandboxResponse) {}
    rpc StopPodSandbox(StopPodSandboxRequest) returns (StopPodSandboxResponse) {}
    rpc RemovePodSandbox(RemovePodSandboxRequest) returns (RemovePodSandboxResponse)
    rpc PodSandboxStatus(PodSandboxStatusRequest) returns (PodSandboxStatusResponse)
    rpc ListPodSandbox(ListPodSandboxRequest) returns (ListPodSandboxResponse) {}

    // Container operations.
    rpc CreateContainer(CreateContainerRequest) returns (CreateContainerResponse) {}
    rpc StartContainer(StartContainerRequest) returns (StartContainerResponse) {}
    rpc StopContainer(StopContainerRequest) returns (StopContainerResponse) {}
    rpc RemoveContainer(RemoveContainerRequest) returns (RemoveContainerResponse) {}
    rpc ListContainers(ListContainersRequest) returns (ListContainersResponse) {}
    rpc ContainerStatus(ContainerStatusRequest) returns (ContainerStatusResponse) {}

    ...
}
```



High level vs low level runtimes





Right runtime?

Container Runtime Interface

containerd

cri-o

Native runtimes

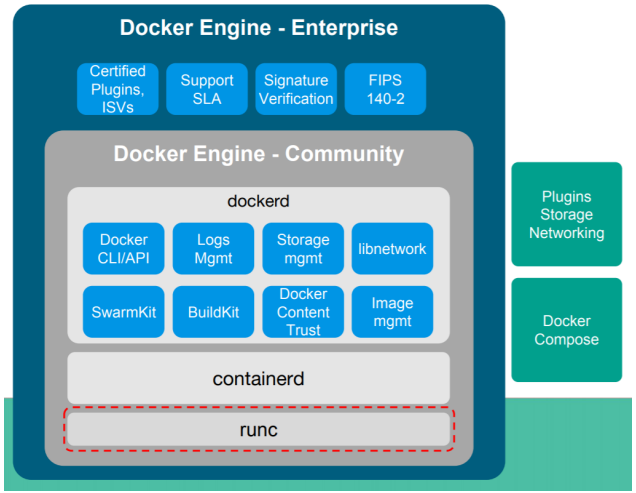
Sandboxed

USER SPACE KERNELS

VIRTUALIZED

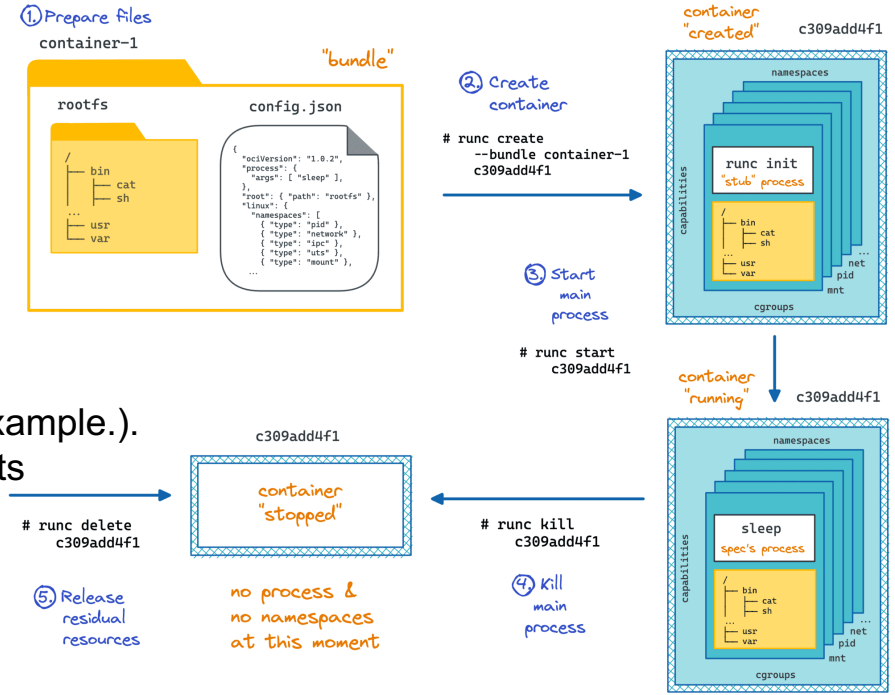
Docker Engine

- It isn’t simply a runtime – it is an entire tech stack
- Deprecated as a runtime in favor of runtimes supporting CRI
- It could be used through a Dockershim
- Not designed to be embedded inside Kubernetes.



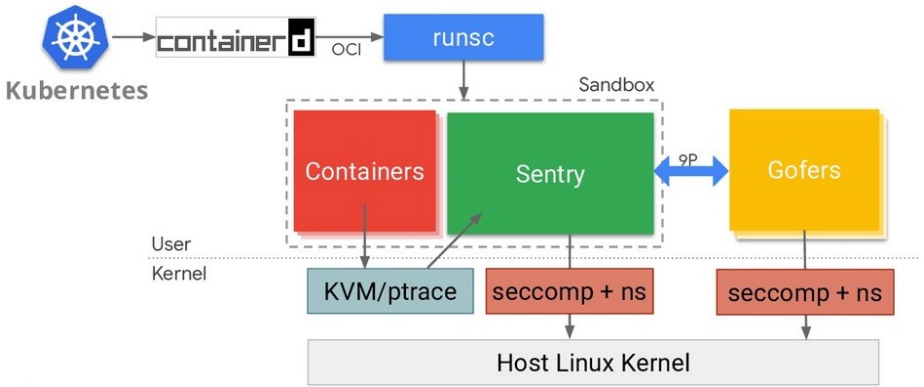
RUNC

- Command line tool for spawning and running containers
- Prepares runtime for container including namespace creation
- Starts (fork/exec) container process and then exits
- improper configuration and usage can lead Into serious security concerns (user could selectively switch off certain NS usage for example.).
- Has native support for security enhancements like seccomp, SELinux and AppArmor

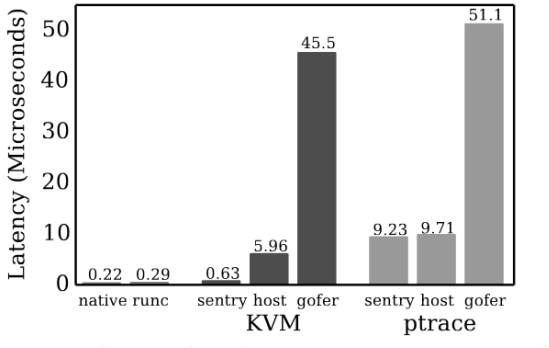


gVisor

- Provide an isolation layer between a container and the underlying kernel through a specialized guest kernel
- gVisor implements a large portion of the Linux system surface
- Implements OCI - compliant runtime called: runsc
- gVisor kernel cannot be trusted, it doesn't have direct access to the file system – a proxy called Gofer is used
- Supports two platforms for intercepting syscalls – ptrace and KVM

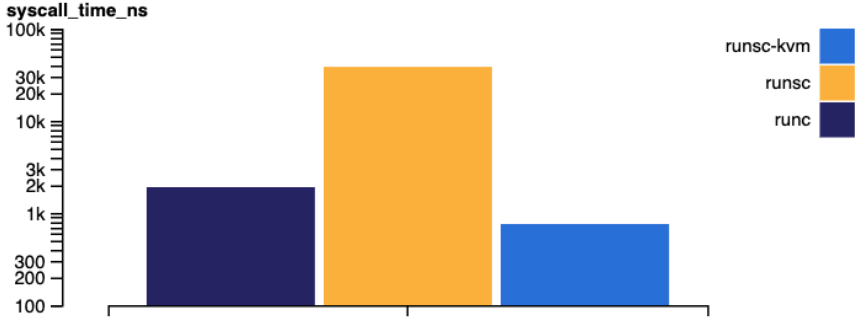


System Call Overhead



The bars show the average latency for `gettimeofday` across 100M executions

Source: The True Cost of Containing: A gVisor Case Study – Ethan G. Young, Pengfei Zhu



The bars show time required for a raw system call on various platforms.
The test is implemented by a custom binary which performs a large number of system calls and calculates the average time required.
1k ns == 1 us

Source: gVisor Performance Guide

gVisor

Pros

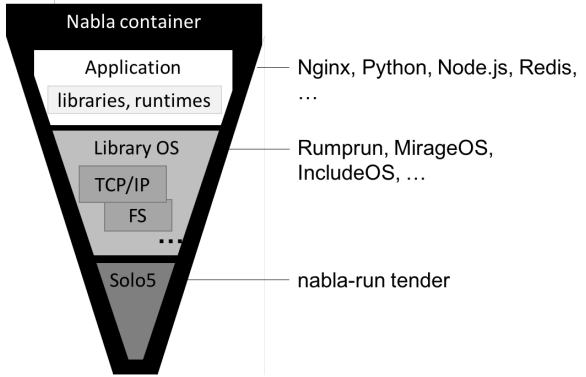
- Even if Sentry is compromised, the workload is still protected by seccomp
- Quick start-up time – about 150ms (collected with /bin/true and /bin/sleep)
- Low resource overhead – about 15 MB memory overhead
- Allow high container density
- Good for running small containers

Cons

- Not suitable for syscall-heavy applications
- The kernel implements not all syscalls
- Not meant to run trusted containers
- Doesn’t allow direct access to the hardware – i.e, passthrough support

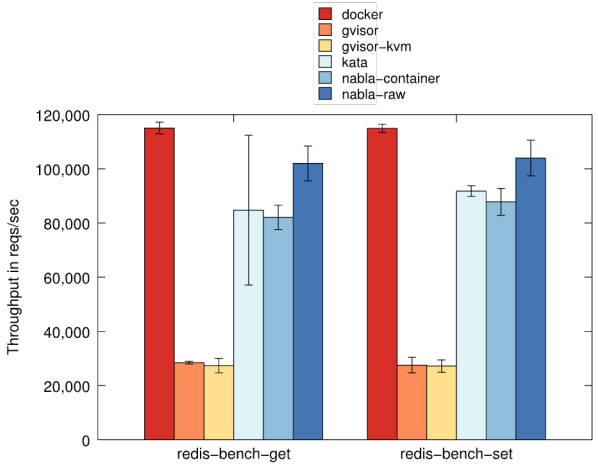
Nabla Containers

- Builds containers on top of Unikernels
- Implements OCI - compliant runtime called: runnc
- Achieve strong isolation by limiting syscalls.
- Use library OS (aka unikernel) techniques
- Library OS implements the SysCall functionality
- Use only 7 system calls - all others are blocked via a Linux seccomp policy.

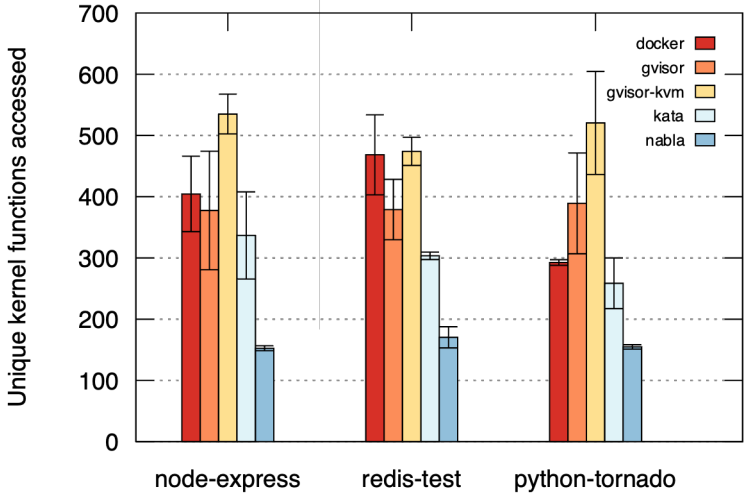


allowed syscalls: read, write, exit_group, clock_gettime, ppoll, pwrite64, and pread64

Nabla Containers



Comparation of network throughput (higher is better)



amount of code traversed in the host kernel captured with ftrace (lower is better)

https://blog.hansenpartnership.com/measuring-the-horizontal-attack-profile-of-nabla-containers/

Pros:

- Lightweight – only what the application uses is compiled and deployed
- Better security – has a small attack surface
- Quick boot time – about 100ms
- Fits many new cloud environments – Serverless, microservices
- Less popular, so less attacked

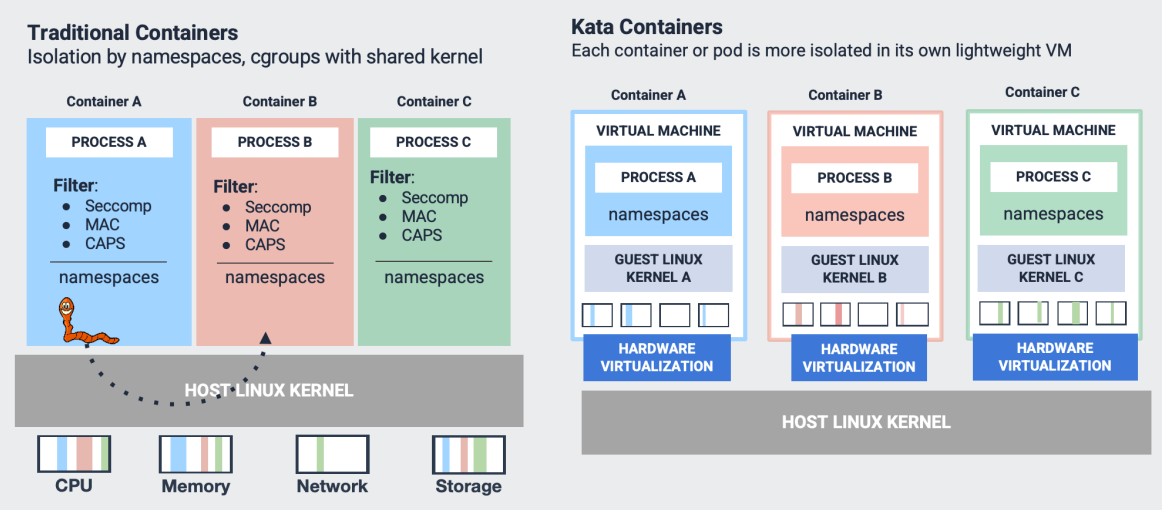
Cons:

- Nabla runtime (runnc) only supports images built for Nabla
- Container runtime limitations – Unable to properly handle /32 IP address assignments.
- A limited set of base images
- Currently, only /tmp is writable.
- Doesn’t allow volume mapping
- Doesn’t support interactive console/tty (dockr run –it)
- Others…

https://github.com/nabla-containers/runnc#limitations

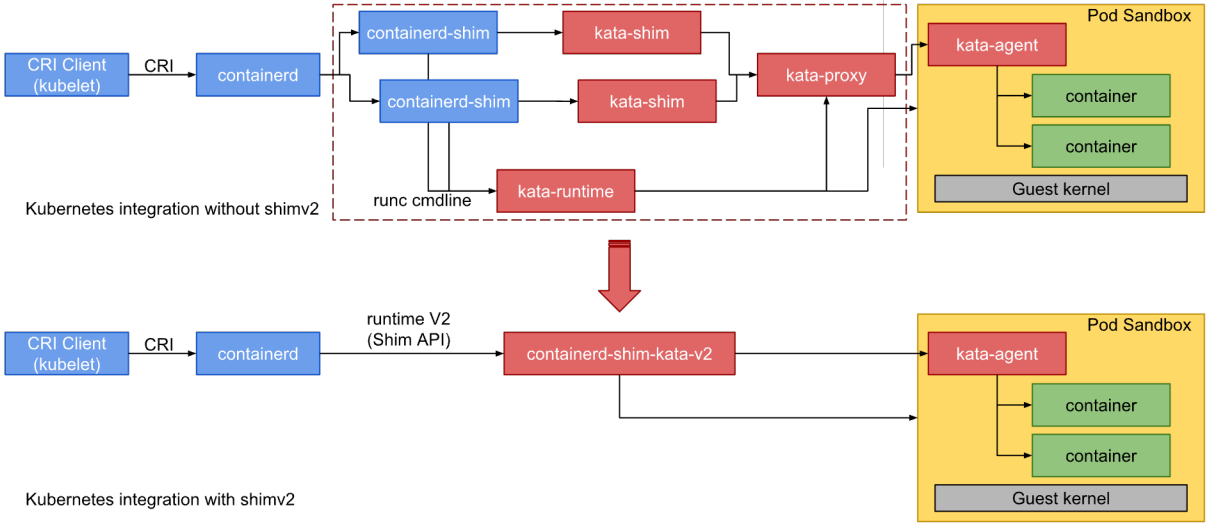
kata containers

- Provide stronger workload isolation (than native runtimes)
- Use lightweight virtual machines that feel and perform like containers
- An OpenInfra Foundation project (previously OpenStack foundation)
- Provides OCI,CRI-O, Containerd compatible runtime: kata-runtime
- Result of a merge between Intel Clear Containers and Hyper.sh RunV



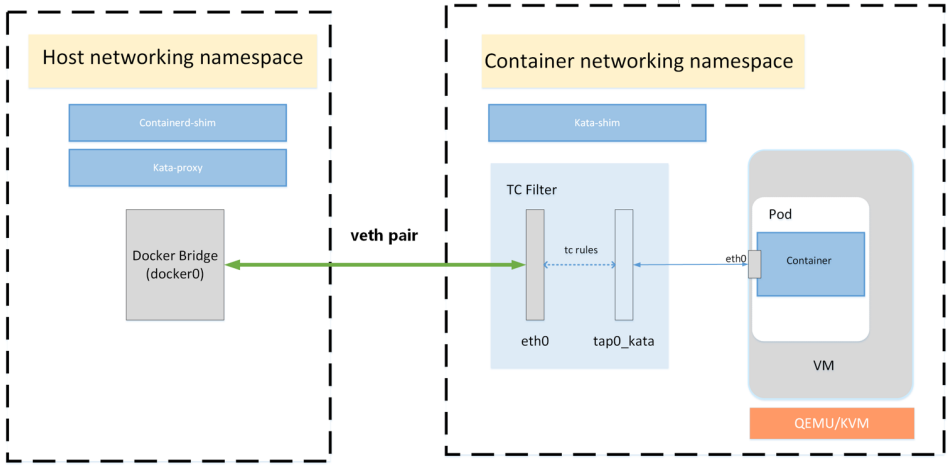
kata containers

- Supports multiple hypervisors – Qemu/KVM, Firecracker, Cloud Hypervisor…



kata containers

- The network between Containers will be shared as a group of namespaces and separated from the network of the host
- There will be a virtual ethernet (veth) connection between them, with one end connected to the Host networking namespace and the other end connected to the container networking namespace.
- The connection between the tap device and the host will be in charge of kata runtime.



kata containers

Pros:

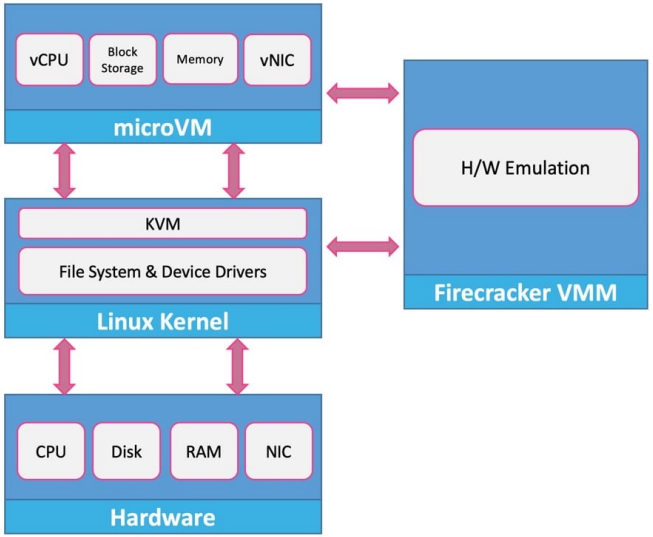
- Bridge the gap between VM security and lightweight containers
- Good compatibility
- Simplicity to use
- Support device hotplug and machine accelerators (QEMU, Cloud Hypervisor/KVM)

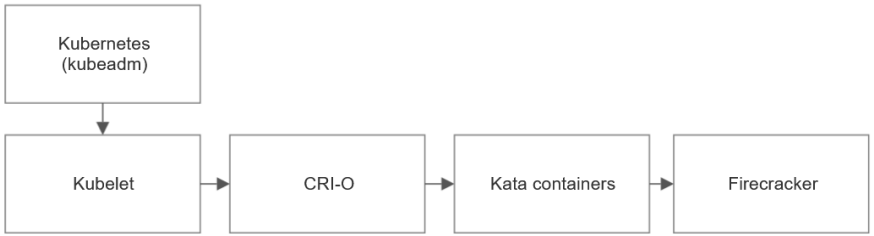
Cons:

- No support for checkpoint and restore
- Doesn’t fully implements events command. OOM notifications are not e fully supported
- HostNetwork Is not supported
- Does not support namespace sharing
- Doesn’t support k8s volumeMount.subPaths

Firecracker

- An OpenSource project by Amazon
- A Virtual machine monitor (VMM) written in Rust
- Uses Linux KVM for management of microVMs
- Performance optimized – about 100ms startup time
- Security Optimized
- Designed for running transient and short-lived workloads – serverless infrastructures
- Based on Googles Chromium VMM (crosvm)





Firecracker

Pros:

- Very quick boot times
- Very small memory overhead <= 5MiB
- Reduce attack surface by removing unnecessary devices and guest-facing functionality.
- Advanced, threat-specific seccomp filters for enhanced security
- Supports adding multiple network interfaces, disks, rate limiting
- Can achieve very high throughput with dedicated host cpu cores

Cons:

- Not a Container Runtime
- Specialized on Linux x86 bare-metal machines (little endian only) with Linux and OSv (unikernel project for cloud computing) guests only – other platforms are not supported
- Not suitable for general purpose workloads

Securing Container workloads

Security Context

Configure and enforce

- Discretionary Access Control (DAC) – UID/GID
- Linux Security Modules – SELinux, AppArmor
- Linux Capabilities
- Seccomp

```
...
securityContext:
  seccompProfile:
    type: Localhost
    localhostProfile: my-profiles/profile-allow.json
```

```
...
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456"
```

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox:1.28
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
      allowPrivilegeEscalation: false
```

Pod Security Standards

- Replace pod security policies.
- Define three different isolation levels for pods
- Broadly cover the security spectrum.
- A built-in *Pod Security* admission controller to enforce the Pod Security Standards.

Profile	Description
Privileged	Unrestricted policy, providing the widest possible level of permissions. This policy allows for known privilege escalations.
Baseline	Minimally restrictive policy which prevents known privilege escalations. Allows the default (minimally specified) Pod configuration.
Restricted	Heavily restricted policy, following current Pod hardening best practices.

Mode	Description
enforce	Policy violations will cause the pod to be rejected.
audit	Policy violations will trigger the addition of an audit annotation to the event recorded in the audit log, but are otherwise allowed.
warn	Policy violations will trigger a user-facing warning, but are otherwise allowed.

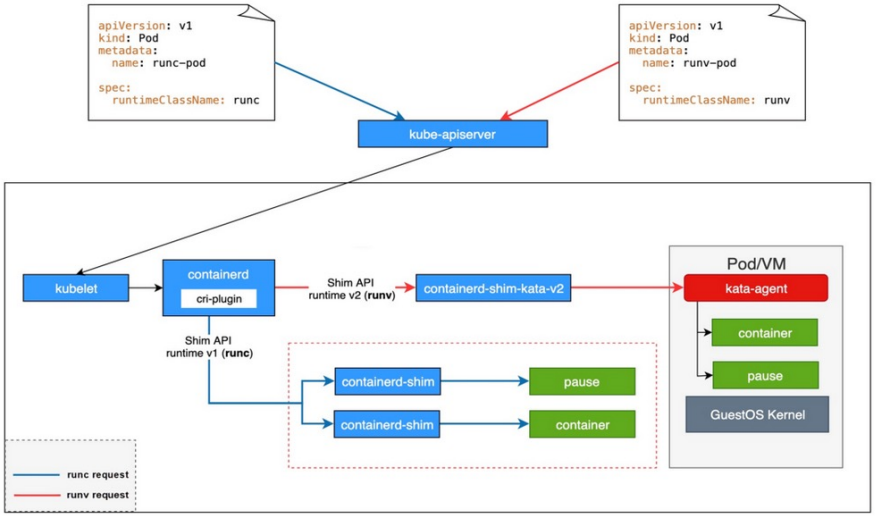
Kubernetes Runtime Class

- Cluster-scoped resource introduced 2018 in K8s 1.12
- Select between different runtimes - provide a balance of performance versus security on a POD level
- Assigned to pods through a runtimeClass field on the PodSpec.
- Can provide additional settings to the runtimes

```
# RuntimeClass is defined in the node.k8s.io
API group
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  # The name the RuntimeClass will be
referenced by.
  # RuntimeClass is a non-namespaced resource.
  name: myclass
# The name of the corresponding CRI
configuration
handler: myconfiguration
```

RuntimeClass Scheduling

- Enables native support for heterogeneous clusters
- Pod authors can select a RuntimeClass without worrying about cluster topology.
- NodeSelector rules and tolerations ensure the pods land on supporting nodes.
- The tolerations are merged with the pod's tolerations in admission



Runtime Class usage

```yaml
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: kata-fc
handler: kata-fc
overhead:
  podFixed:
    memory: "120Mi"
    cpu: "250m"
scheduling:
 nodeSelector:
    katacontainers.io/kata-runtime: "true"
 tolerations:
 - effect: NoSchedule
   key: runtime
   operator: Equal
   value: ”sandbox"
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  runtimeClassName: kata-fc
  containers:
  - name: busybox-ctr
    image: busybox:1.28
    stdin: true
    tty: true
    resources:
      limits:
        cpu: 500m
        memory: 100Mi
```


Q&A

References:

- https://attack.mitre.org/matrices/enterprise/containers/
- Cgroups, namespaces, and beyond: what are containers made from?
- https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/
- The True Cost of Containing: A gVisor Case Study - Ethan G. Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
- Container Isolation at Scale - Dawn Chen and Zhengyu He
- Unikernel Monitors: Extending Minimalism Outside of the Box - Dan Williams Ricardo Koller

Thank You!