Kata Containers: Challenges and State of the Art

Rodrigue Chakode - Continental

Journées Cloud 2019,IRIT

Short Curriculum

PhD Computer Science -- HPC, Distributed Systems & Cloud - INRIA/LIG/Grenoble University (2012)

→ Heavy Grid'5000 practitioner (deployment challenge, 2011)

Various software engineering positions since 2006

→ Currently Cloud Software Architect, Continental

Continental
The Future in Motion

Links

→ github.com/rchakode

→ medium.com/@rodrigue.chakode

→ slideshare.net/RodrigueChakode

→ www.linkedin.com/in/rodriguechakode/

Breadcrumb

How to run multitenant, i.e. untrustworthy and potentially malicious, container workloads on today's microservices based architectures so to guaranteeing high performance and strong security isolation.

What'll be discussed

Trends on computing technologies

→ Bare metals

→ Virtual machines, cloud computing

→ Containers, cloud native computing

Multitenancy, a challenge for cloud native platforms

→ Kata Containers, or the emerging of *virtual machines as "containers"*

→ Leverage the strong isolation of virtual machines with the speed of containers

→ Research perspectives

Computing technologies: Before 2000

On-premises data centers

→ Mainframes, bare metal servers

→ Clusters

→ Grids

→ ...

Source

Computing technologies: 2000-2015

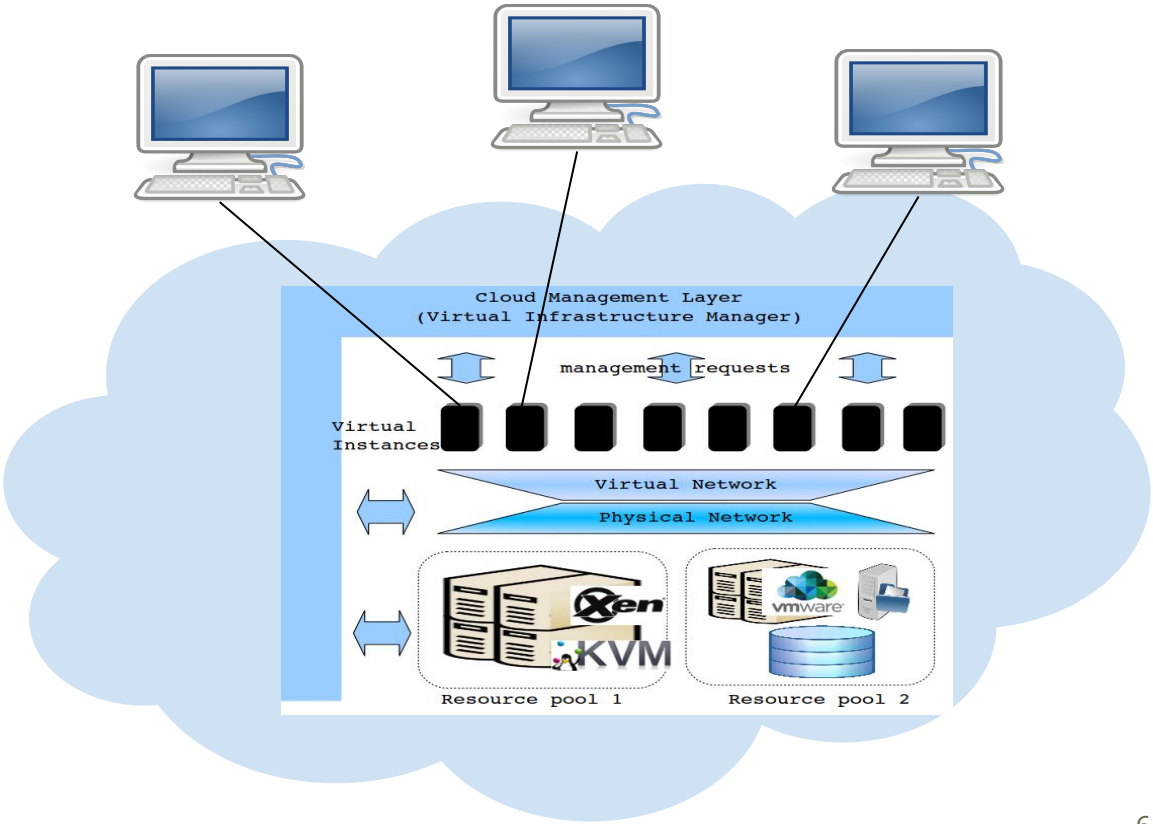
Virtualization and Cloud Computing

→ Virtual machines

- VMware, Xen, KVM…

→ On-demand virtual resources

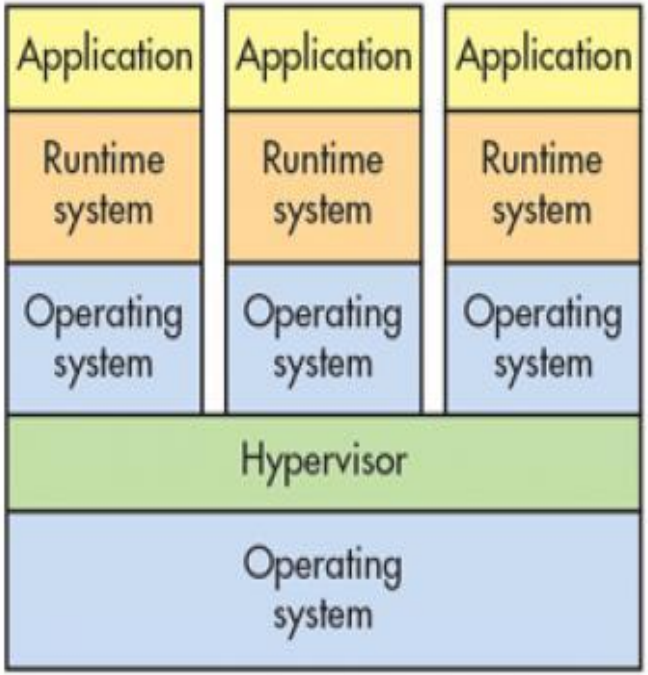
- Public clouds
- Private and hybrid clouds



Review basics on virtual machines

Virtual machine (VM)

- Emulated HW
- Dedicated host/guest OSes



Pic. Source

Pros & cons of virtual machines

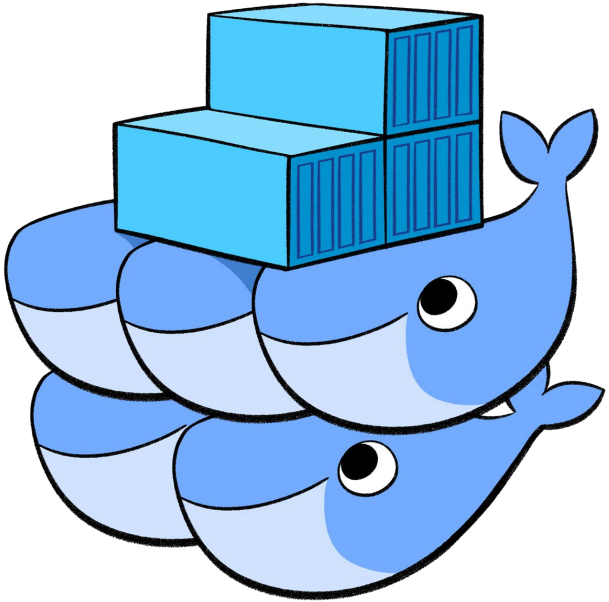
	Pros	Cons
Virtual machine • Emulated HW • Dedicated host/guest OSes	• Strong workload isolation • Reduced attack surface • Live migration	• Slow startup time • High performance overhead • Low workload density/host

Computing technologies: 2015 onwards

Containers

→ Docker on production

→ Tools to orchestrate containers at scale (Mesos, Swarm, Rancher, …)



Source

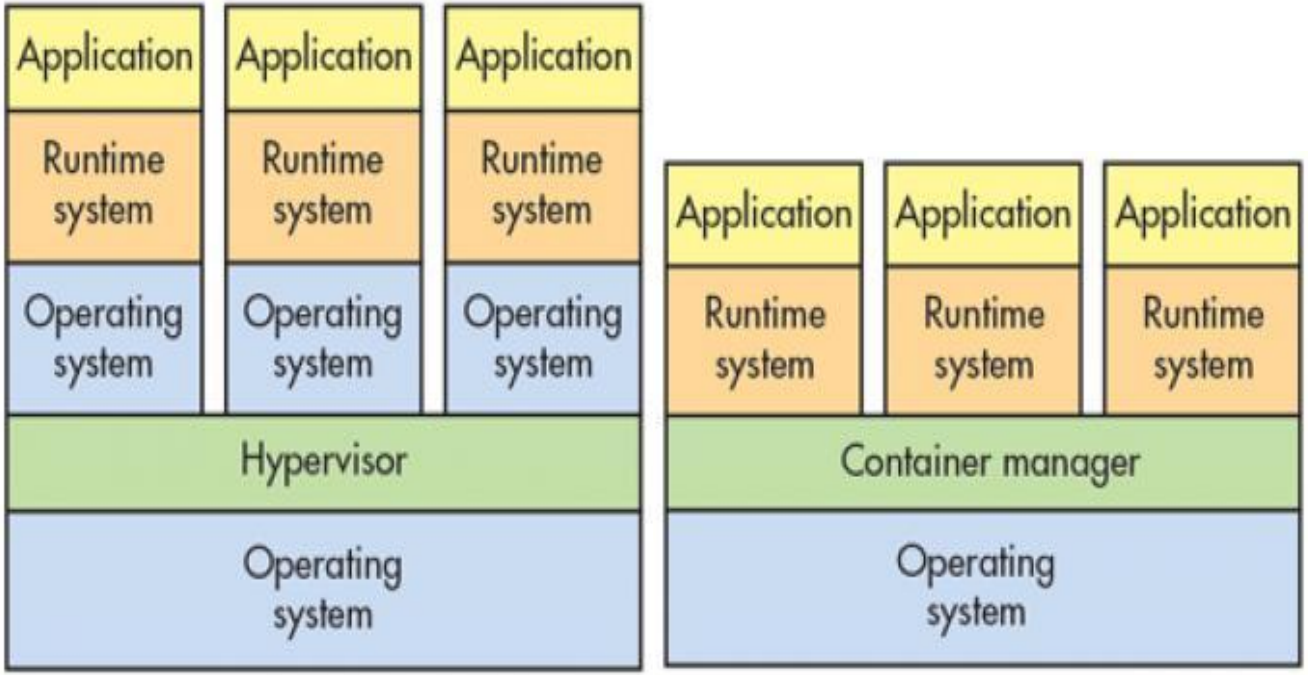
Review basics: Container

Virtual machine (VM)

- Emulated HW
- Dedicated host/guest OSes

Container

- No HW emulation
- Linux namespaces/cgroups
- Shared host OS



Source

Pros & cons: Container

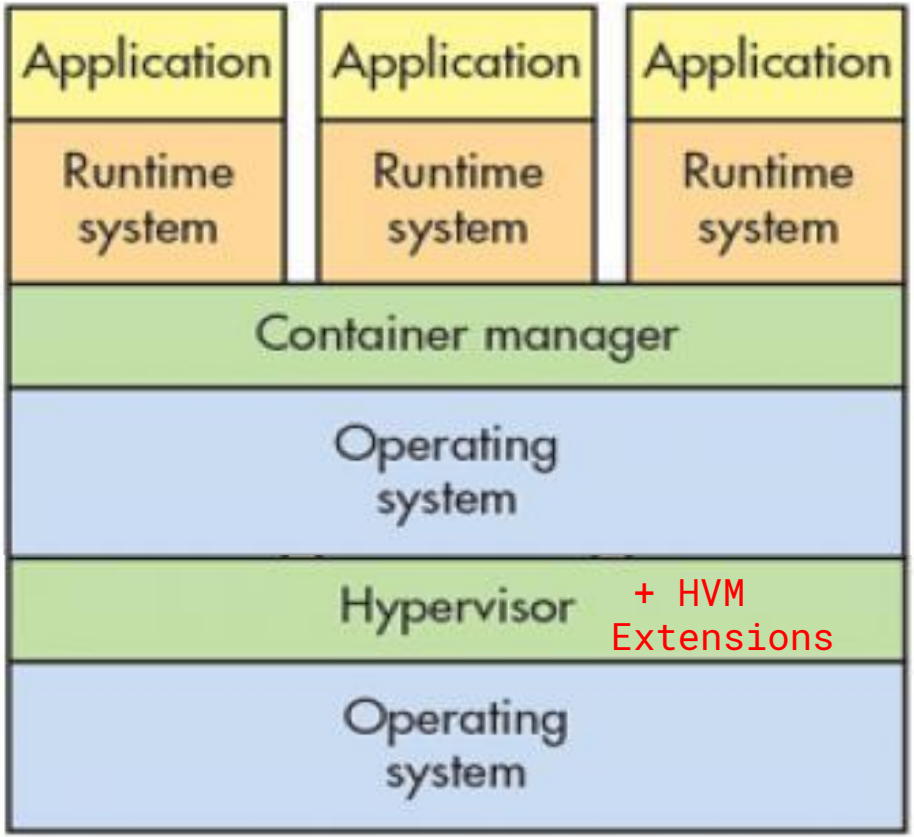
	Pros	Cons
Virtual machine • Emulated HW • Dedicated host/guest OSes	• Strong security isolation • Reduced attack surface • Live migration	• Slow startup time • High performance overhead • Low workload density/host
Container • No HW emulation • Linux namespaces/cgroups • Shared host OS	• Fast startup time • High performance (near native) • High workload density/host	• Shared host OS • Low isolation, security threats • No live migration

Anecdotal facts

In many organizations containers are actually executed inside VMs

→ Hosted on public clouds

→ Few, yet expensive, bare metal instances on public clouds



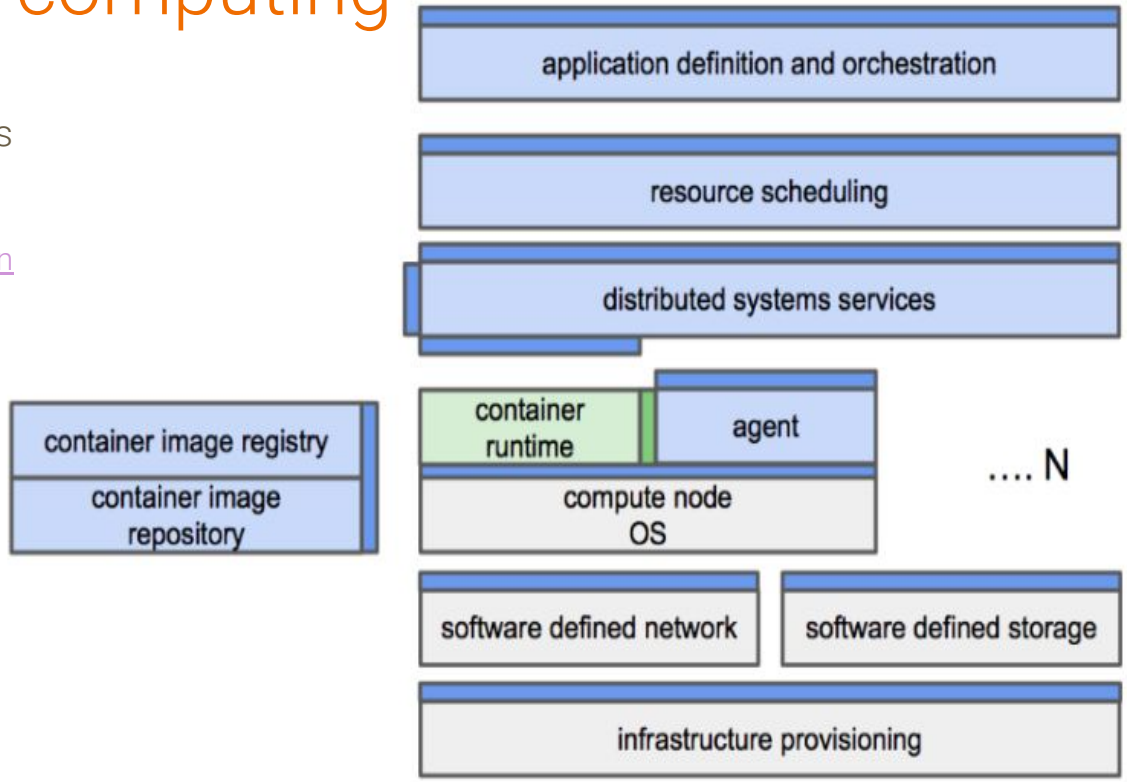
Containers on virtual machine

Rise of cloud native computing

→ Driven by open governance initiatives

- Open Containers Initiative
- Cloud Native Computing Foundation

→ Kubernetes, the "mascot"



First conceptual architecture by CNCF (2015)

Cloud Native is the new way to go

Kubernetes to automate the deployment, the orchestration, and the scaling of containerized workloads

→ Created by Google and donated to CNCF/Linux Foundation (2015)

→ Increasingly huge ecosystem

→ *"Kubernetes is Linux of cloud"*, everything should be run in Kubernetes?

35K contributors

148K code commits

83K pull requests

1.1M contributions

2,000+ contributing companies

Source

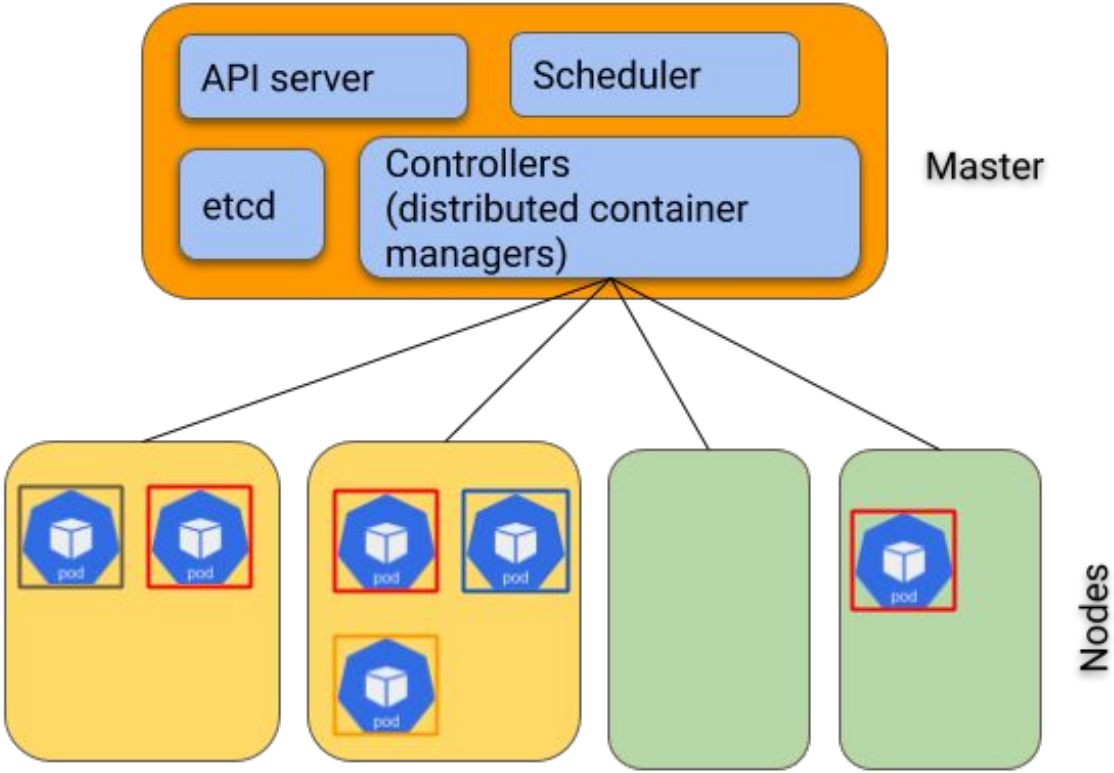
Review of basics: Kubernetes

Resource sharing is at its core design

→ Control plane (1+ masters)

→ 1+ worker nodes

→ Workload (Pod) = 1+ containers



Kubernetes multitenancy attack surfaces

Control plane shared among users

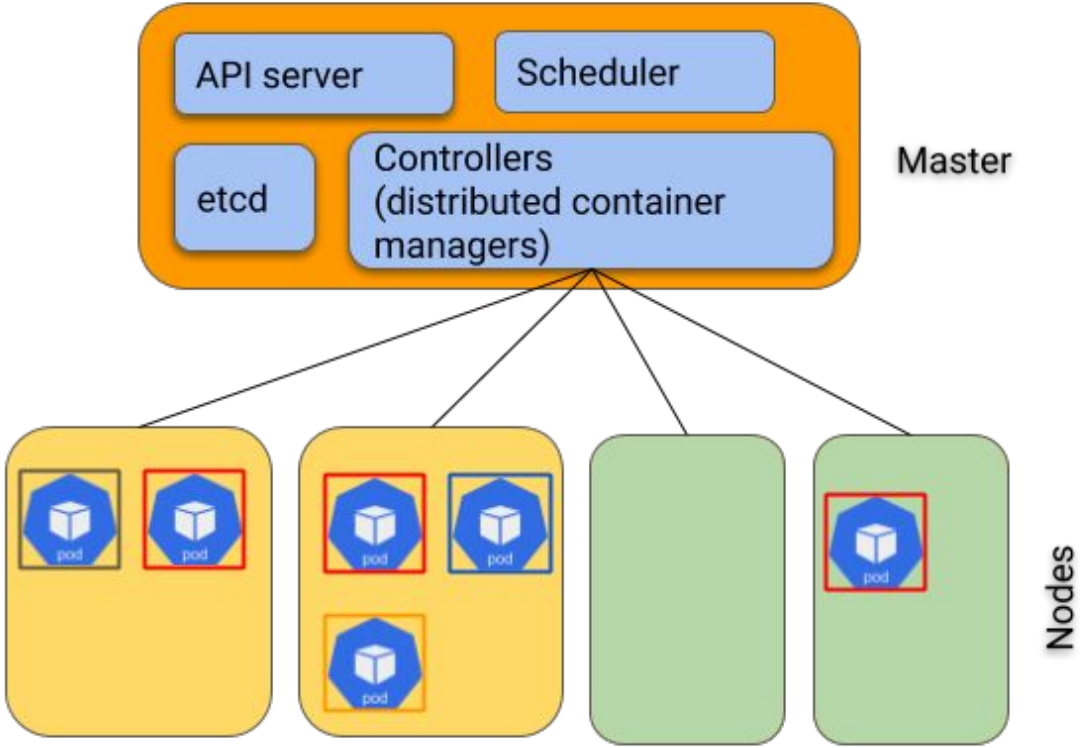
→ DoS attacks on API service

Node OS shared among containers

→ OS is a SPOF on each node

→ Risks on privileged containers

→ Trade-offs on app compatibility



Current multitenancy with Kubernetes is **soft**

Restrictive containers

→ Restricted Linux capabilities, no privileges escalation

- E.g. Docker containers with blocked syscalls
  - settimeofday
  - swapon/off
  - (u)mount
  - More: https://docs.docker.com/engine/security/seccomp/

→ Security hardening through Seccomp, AppAmore, SELinux, …

→ **Trade-offs on application compatibility and capabilities, no protection on control plane**

Each technology solves and brings problems

Kubernetes solves the problem of highly scalable containerized applications

→ Stateless applications at first

→ All workloads assume a trustworthy execution environment (multitenancy is not built-in)

New usages, not always carefully analyzed (e.g. risk of misuse or no actual added values)

→ **Running (untrusted) multitenant workloads on a single cluster**

→ **Anti-patterns?** (e.g. deploy any stateful applications on Kubernetes)

Refocus and fix up problems

Facts

→ Containers become a kind of "de-facto standard" to run enterprise applications

→ Containers orchestrated at scale, Kubernetes being driving the transformation at fast pace

→ Multitenant workloads executed as containers on virtual machines

Problems

→ Handle **hard** multitenancy with untrustworthy and potential malicious workloads

Hard multitenancy with Kubernetes platforms?

Separated resources per tenant

→ Resource sprawl, with impacts on costs

Native platform-wide mechanism

→ Requires means to handle hard segregation regarding the orchestrator and workloads

→ Active topic in Kubernetes community (e.g. Kubernetes Multitenancy Working Group)

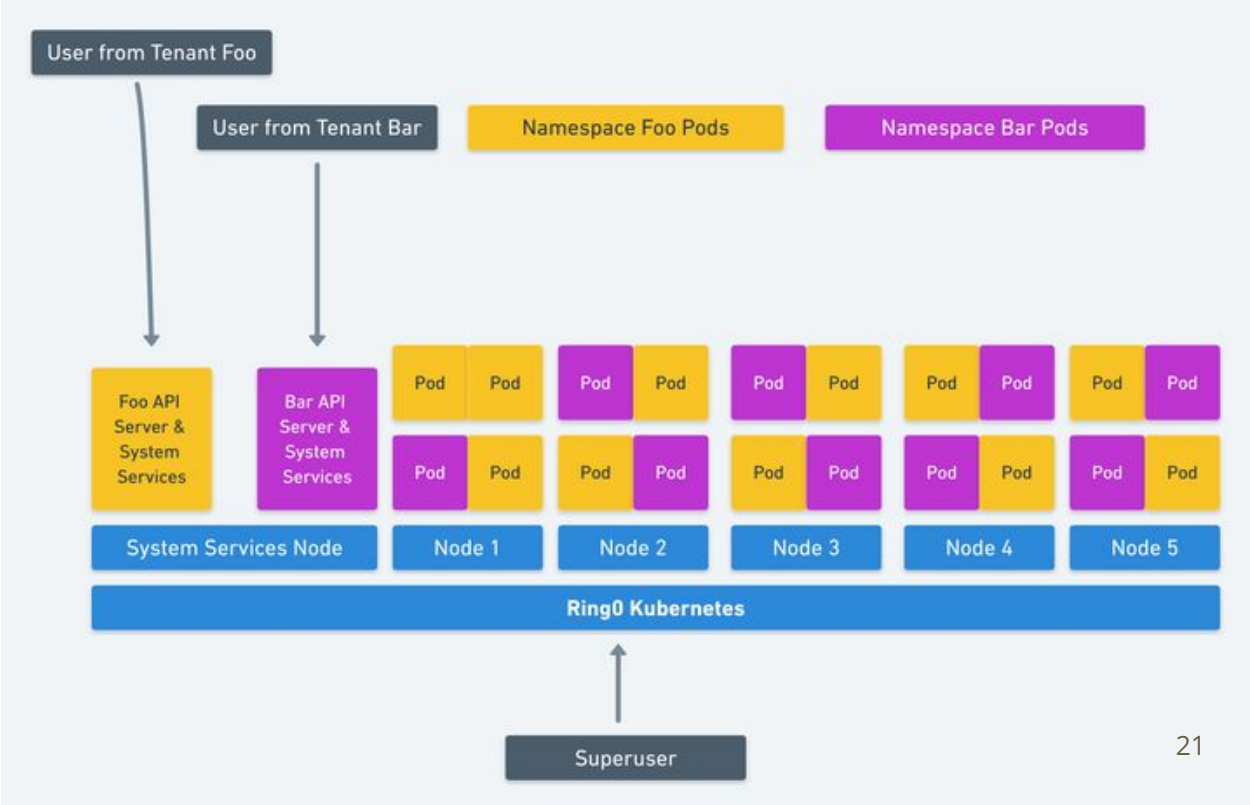
→ **Hardened sandboxed workloads (restrictive containers, *virtual machines as "containers"*)**

A vision of **hard** multitenancy in Kubernetes

By Jessie Frazelle

→ Deeply inspired from Linux (namespaces/cgrougs)

→ Hard isolation through virtual machines as containers



Source

Virtual machines as “containers”

Hardening container workloads by leveraging the strong isolation of VMs and the speed of containers

→ Strong isolation of virtual machines

→ Performant as containers

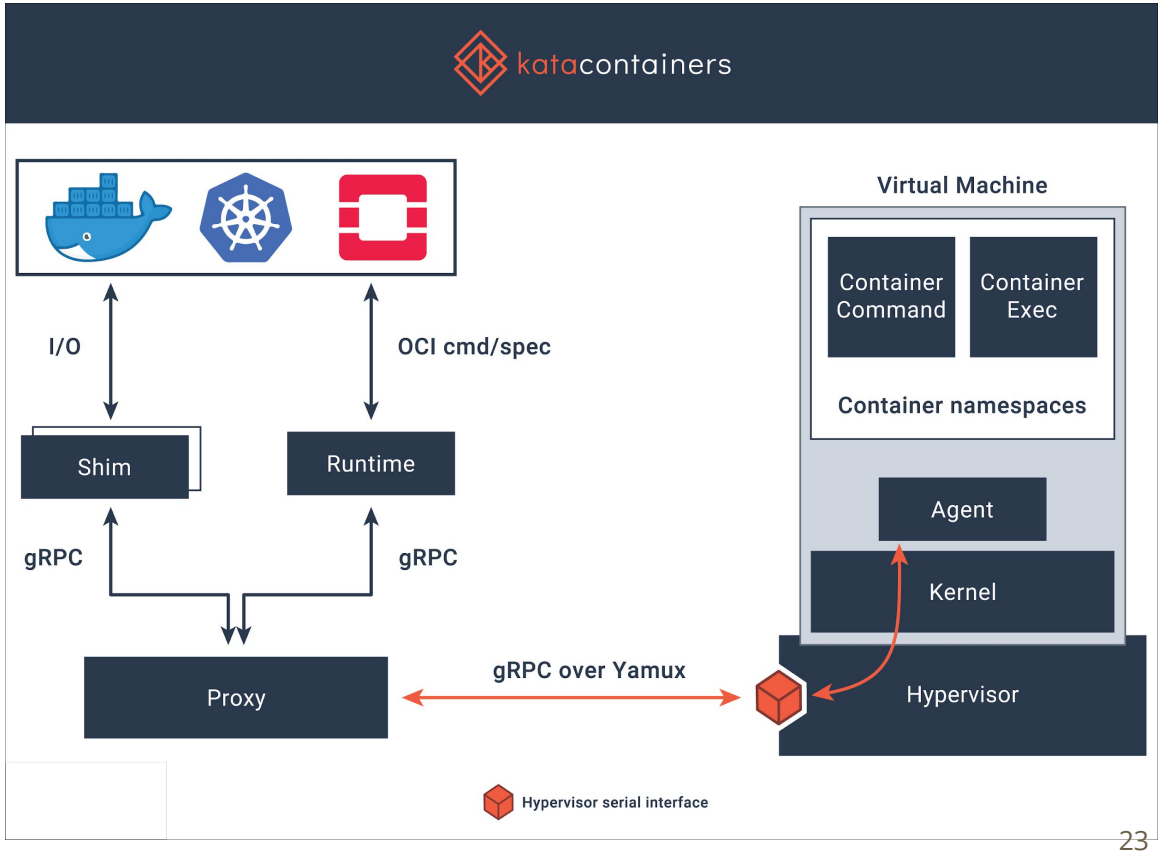
→ Compliant with existing OCI-compliant container engines

→ Active area of innovation with projects such as Kata Containers and Firecracker

Kata Containers

Secure container runtime with lightweight virtual machines

- → Launched in 2017
- → Driven by a open source community
- → **Hypervisor + microVM**
- → OCI-compliant

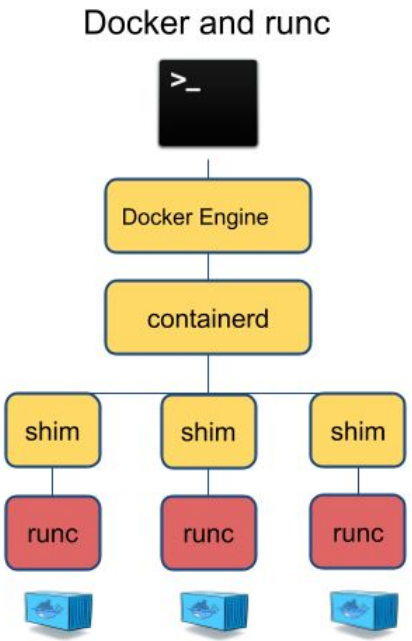


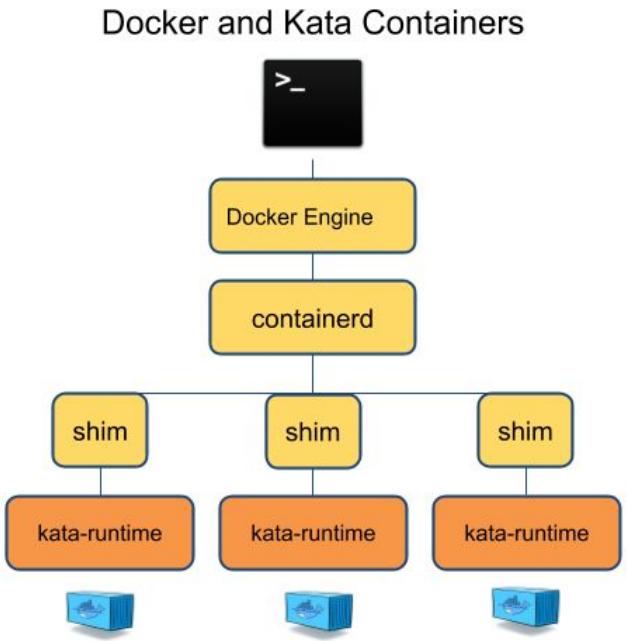
Kata Containers brings seamless integration

OCI runtime implementation

→ Containerd, CRI-O

→ Multi-architectures: x86_64, AMD64, ARM, IBM p/z-series





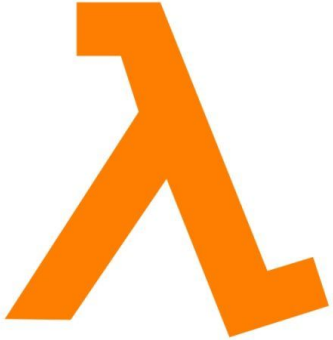
Firecracker

Multitenancy with efficient resource use

→ Initially designed for AWS, open sourced in nov. 2018

→ Improved security and reduced startup time

→ Not OCI-compliant, but planned to work as runtime for Kata Containers



Firecracker

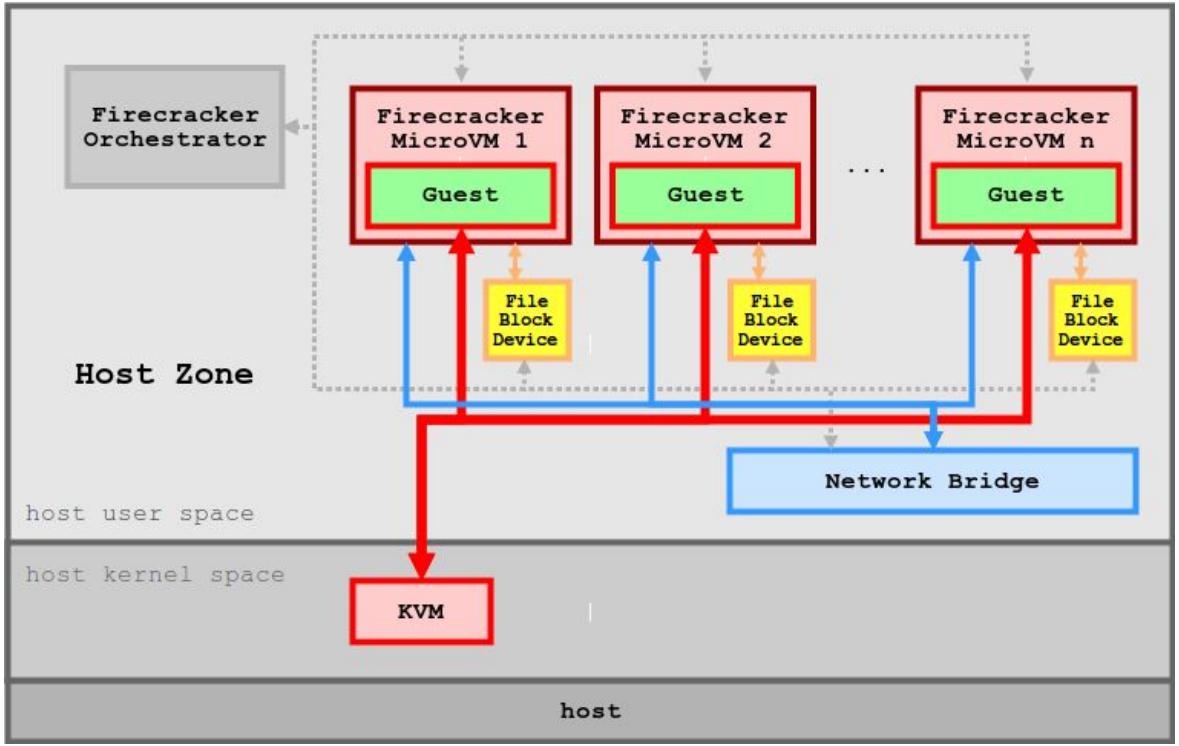
Hypervisor and runtime for

lightweight multitenant workloads

→ KVM, with restricted devices and functionality

→ **microVM** in user space

→ Only Intel CPUs currently



Source

Kata Containers Demo

My Setup

- → Ubuntu 16.04
- → Docker 18.09.5
- → Kata Container 1.9.0-alpha0
- → Officiation Installation Guide

Add Kata Runtime on Docker

```
$ vi /etc/docker/daemon.json

    {
        "kata-runtime": {
          "path": "/usr/bin/kata-runtime",
          "runtimeArgs": [
            "--kata-config",
            "/opt/kata/share/defaults/kata-containers/configuration-fc.toml"
          ]
        }
      }
    }
```

Run a Kata Container on Docker

```
$ uname -a

$ docker  run \
          -it \
          --runtime=kata \
          --cpus=1 \
          --memory=512MB \
          busybox \
          sh

$ free -m

$ cat /proc/cpuinfo

$ uname -a
```

Kata Containers base configuration

```
$ cat /etc/kata-containers/configuration.toml
```

Kata Containers Limitations (vs containers)

→ Pre-allocated computing resources (CPU, memory, network bridges)

→ Minimum resource footprint can be significant (`default_cpus`, `default_memory`) [1]

→ Need to keep `default_cpus=1` for fast boot time and small memory footprint [2]

→ Young technologies, still needing to be improved [3]

- Join an existing VM network (similar to `$ docker run --net=containers`)
- Docker host network support (`docker --net=host`)
- More

Other research perspectives

Performance on HPC/computing intensive applications

→ Raw performances compared against bare metals, VMs and containers (e.g. [1])

→ Performances with nested virtualization (microVM in VM, [2])

Distributed systems and clouds

→ Need to optimize placement and consolidation of microVMs to avoid wasting resources [3]

High performance storage drivers

→ Current drivers [4] (`virtio-9p`, `virtio-fs`) have poor performances [1] or are still young

Q/A

Thanks