Bringing container security to the next level using Kata Containers

Flavio Castelli
Senior Engineering Manager
fcastelli@suse.com

Application containers

- A way to distribute and run applications
- Portable
- Consistent
- Reproducible

What is needed to run a container?

- Container image: a minimal OS + the application and its dependencies
- Container engine: takes container images, spawns a container based on it

Containers have standards

The Open Container Initiative (OCI) is a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime.

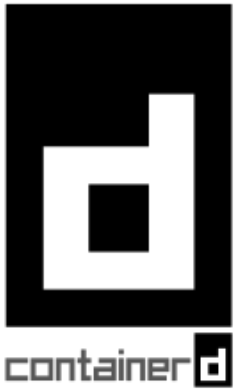
OCI Image Specification (image-spec)

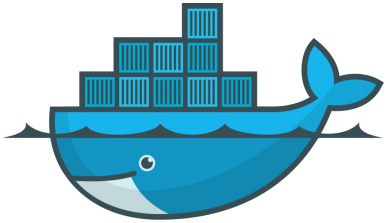
- Covers how a container image is structured
- All container image building tools are producing OCI images (docker, buildah, umoci, KIWI,...)

OCI Runtime Specification (runtime-spec)

- **Specification of a container runtime (API, data structures,...)**
- **Purpose:**
  - Given an extracted container image and some settings…
  - Creates a container running the user-specified application
- **OCI provides a reference implementation of the spec: runC**

Everybody ❤ runC





runC

Take-away lesson

- **Thanks to OCI image spec: I can use the same container image with different container engines**
- **Thanks to OCI runtime spec: I can swap the container runtime underneath a container engine without too much fuss**

Container security

What are “traditional” containers made of

- At the core, containers are just processes running on the host
- Container and host share the same kernel
- Isolation features: namespaces (kernel feature)
- Resource control: cgroups (kernel feature)

Security threats

- If attacker breaks out of the container he’s on the host system
- Attacker can find himself being root on the host!

How to reduce the attack surface

- **Leverage “Linux Capabilities”:**
  - More granular way of providing privileges than just “root”
  - If compromised, the process has limited rights
  - Safer than running with full-root privileges
- **Write seccomp profiles:**
  - Fine-grained syscall blocking and filtering
  - Reduces the attack surface
- **Simply do not run as root inside of the container**

How to do damage control

- **Leverage “user namespaces”:**
  - `root` inside of the container is mapped to an unprivileged user
  - Limit what the attacker can do
- **Write SELinux or AppArmor profiles:**
  - Define what a user can access/perform on the host
  - Good also for auditing

What about using traditional virtualization?

- It’s possible to run containers inside of virtual machines
- Pretty common deployment method (think about the cloud)
- Introduces another layer of protection: the hypervisor

But…

- Not flexible: cannot mix trusted and untrusted workloads inside of the same VM (e.g. CNI deployments on kubernetes)
- Hypervisors can have security bugs as well

Kata Containers

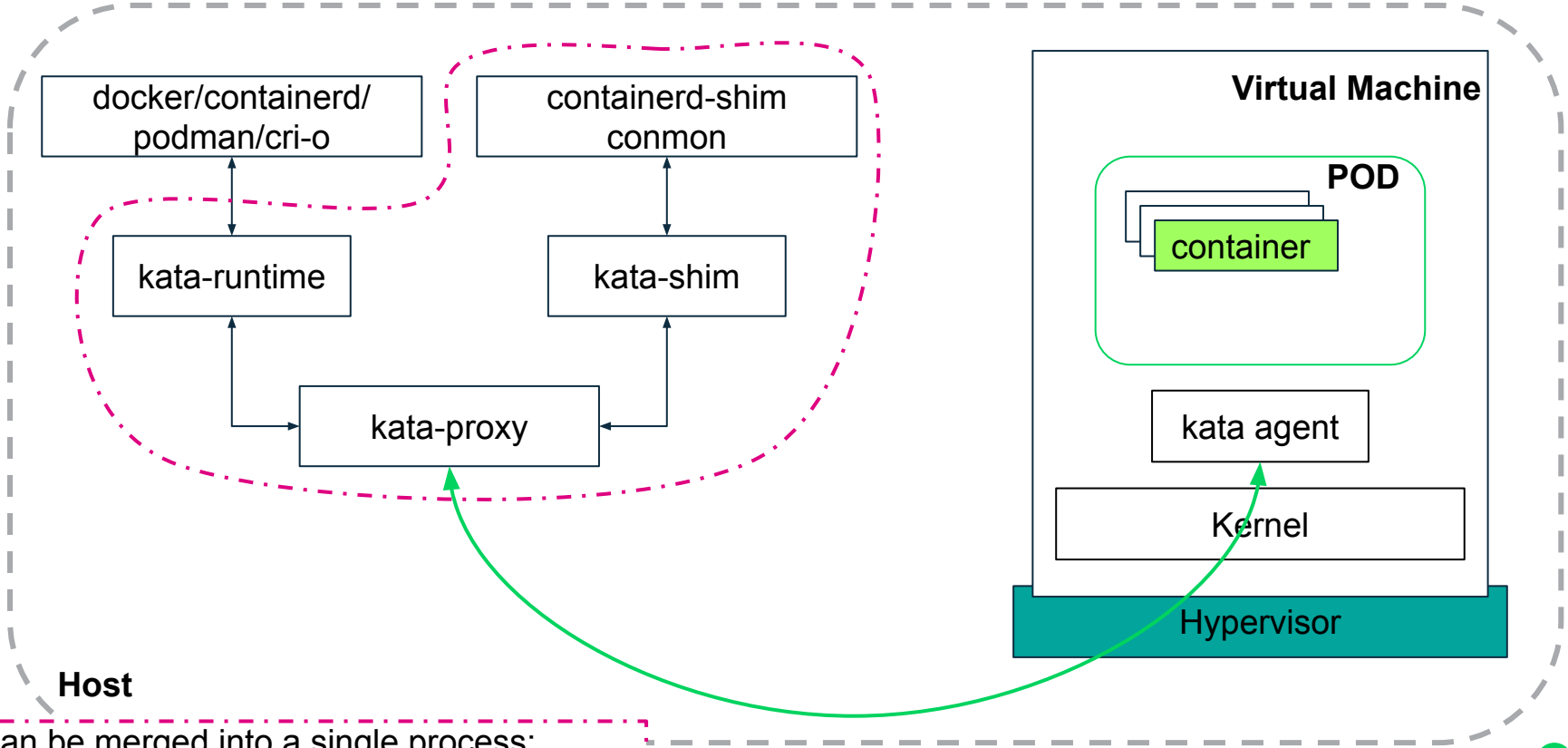
Quick summary

- Born in 2017 from the merge of Intel’s Clear Containers and Hyper’s runV
- Project managed by the OpenStack Foundation
- “Wraps” containers into dedicated virtual machines

Plug & play

- **OCI runtime implementation: can be plugged into the container engine instead of runC**
- **Can consume the same container images already existing:**
  - No need to rebuild them
  - No need to find new ways to distribute them

Components



Can be merged into a single process:
`containerd-shim-kata-v2`

Components - host

- **kata-runtime:**
  - OCI runtime implementation
  - Manages container states
  - Control VMM
- **kata-proxy:**
  - Multiplexes I/O streams and commands to kata-agent in a single "serial"
  - Relies on `virtio-serial`
- **kata-runtime can be configured to use `vsock`: proxy becomes redundant**
- **kata-shim: container engine point of interaction**
  - Container reaping
  - stdin/stdout/stderr handling
  - Signal handling

Components - guest

- **Guest VM:**
  - Trimmed down GNU/Linux system
  - Special kernel: optimized for fast boot and reduce memory usage
  - `systemd` spawns `kata-agent`
- **kata-agent:**
  - container management inside of the VM
  - uses “`libcontainer`” -> shares most of its code with runC

Virtual Machine Manager

- **Kata backend is pluggable: multiple VMM are supported**
- **Supports:**
  - NEMU
  - QEMU
  - QEMU-lite
  - firecracker
- **Machine accelerators:**
  - Improve performance
  - Architecture specific
  - Some of them are shipped only as part of qemu-lite
- **Choose QEMU when extensive HW support is needed**
- **Choose firecracker for low memory footprint and microservices workloads**

Storage

- **Two possible ways to “inject” container image into the VM:**
  - 9pfs: works at file level
  - devicemapper storage: works at block level
- **Devicemapper provides better performances but:**
  - Is not going to work with recent versions of docker
  - Copying files in/out of the container (e.g.: `docker cp`) doesn’t work

Network

- Network implemented using CNI (podman, k8s) and CNM (docker)
- Kata Containers creates a network namespace for each VM
- Known limitation: Kata Containers cannot reuse the host network

Kubernetes integration

Kata Containers and kubernetes

- **kubelet has the concept of Container Runtime Interface (CRI)**
- **Abstracts the container engine to be used**
- **Allows kubernetes to use different container runtime like:**
  - docker
  - containerd
  - cri-o
  - rkt
  - ...

Integrating kata with kubernetes

- **Kubernetes supports different container runtimes at the same time**
- **A node can run pods with different CRI**
- **Great to separate trusted from untrusted workloads:**
  - Trusted: managed via runC
  - Untrusted: managed via Kata Containers

Kubernetes RuntimeClass

- **Introduced with kubernetes 1.12, graduated to beta with 1.14**
- **Allows to define different multiple CRI and/or multiple configurations of the same CRI:**
  - CRI-O with runC
  - CRI-O with kata with QEMU-lite
  - CRI-O with kata with firecracker
  - CRI-O with kata with QEMU-lite with option X, Y and Z
  - …

Kubernetes RuntimeClass

- **Define a default CRI configuration**
- **Allow user to specify a different one for certain workloads**
- **Can be used in conjunction with kubernetes admission controllers:**
  - Rewrite the certain workloads definitions to use a specific CRI
- **Possible scenario:**
  - All system workloads (eg: CNI, logging, monitoring,...) use runC
  - All workloads started by group “developers” are forced to use Kata Containers with QEMU-lite

Kubernetes RuntimeClass example

```yaml
apiVersion: node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
  name: crio-kata-qemu-lite
  # RuntimeClass is a non-namespaced resource
handler: qemu-lite   # The name of the corresponding CRI configuration
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  runtimeClassName: crio-kata-qemu-lite
  # ...
```

Special guest: firecracker

A surprise from AWS

- Project published by AWS during Nov 2018
- Used internally by AWS Fargate and AWS Lambda
- Virtual Machine Manager that leverages KVM
- Currently supports Intel CPUs; AMD and Arm is on the roadmap

Focus areas

- **Security:**
  - Minimalist design to reduce attack surface
  - Designed to execute untrusted code
  - Written using Rust language
  - Several security barriers: seccomp, cgroups, namespaces, rate limiting,...
- **Fast**
- **Reduced footprint**

Bleeding edge

- Requires kernel >= 4.14
- Requires devicemapper driver (works at block level)
- Kata Containers seems eager to make that the default VMM

DEMO

1 billion $ question...

Multi-tenants kubernetes: are we there yet?

- Unfortunately not
- Lot of variables to consider
- Quite some work is being done in this area

That would be worth a dedicated talk...

- **In the meantime take a look at these blog posts:**
  - https://blog.jessfraz.com/post/secret-design-docs-multi-tenant-orchestrator/
  - https://blog.jessfraz.com/post/thoughts-on-conways-law-and-the-software-stack/


SUSE
We adapt. You succeed.