KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

CRI-O
Under the Hood

Mrunal Patel and Sascha Grunert

Last year in CRI-O

Increasing contribution statistics

increased up to 10 OWNERS from three different companies

around 30 individual contributors per release

around 190 changes per release

We would be happy to welcome you inside our community!

Last year in CRI-O

Achievements

released CRI-O 1.14, 1.15, 1.16 and a certain amount of patch releases

better out of the box behavior by improved configuration and packaging

providing community feedback by testing Kubernetes master and alpha/beta releases

Highlights from CRI-O 1.16.0:

- Support for Manifest lists
- IPv6 dual-stack support
- conmon is now a dedicated project

Getting started with CRI-O

Responsibilities of CRI-O inside a Kubernetes cluster

Interfacing with lower level container runtimes, like runc, kata-containers or crun

Fulfilling the Kubernetes Container Runtime Interface (CRI) API

- Managing container images
- Managing pod sandboxes and containers

Providing metadata around containers via monitoring and logging

Providing the right amount of resource isolation

Getting started with CRI-O

Running Kubernetes with CRI-O

Runtime dependencies: runc (or any other drop-in replacement), conmon, cni-plugins

Main configuration parameters of CRI-O:

- storage drivers: overlay, btrfs, vfs
- the listener socket

The kubelet connects to CRI-O:

```
> kubelet --container-runtime=remote \
    --container-runtime-endpoint=unix:///var/run/crio/crio.sock
```

CRI-O: Under the Hood

Lifecycle of a Container Workload in kubernetes

A complete workload creation needs multiple kubelet gRPC requests to succeed:

1. Initialize the sandbox: `RunPodSandboxRequest`
2. Continuously verify that the sandbox is running: `PodSandboxStatusRequest`
3. Pull the container image: `PullImageRequest`
4. Create the containers: `CreateContainerRequest`
5. Start the containers: `StartContainerRequest`

CRI-O: Under the Hood

Container Networking

CRI-O interferes with the Kubernetes Container Networking Interface (CNI)

Different available network types for pod sandboxes:

- Host Network
- Default configured CNI plugin

CRI-O picks up the CNI configuration from the node on startup and during runtime

Multiple IP address assignments possible due to IPv6 support

Necessary port mappings are applied on sandbox setup

CRI-O: Under the Hood

Interacting with Containers

Two main types of container interactions: unary and streaming

Most user interactions are done via `kubectl`, (like `exec`, `port-forward`, `logs`)

Streaming interface of CRI API allows streaming based interactions

- Unary gRPC request returns streaming server endpoint to Kubelet
- Kubelet streams results directly to the user

Unary interactions (like `kubectl get pods`) are done periodically (resync)

CRI-O: Under the Hood

Container Workload Cleanup

CRI-O restores the systems state on container removal

1. `StopContainerRequest`
    a. Verify that container is not paused or already stopped
    b. Stop the processes and wait for their removal
2. `StopPodSandboxRequest`
    a. Verify that no containers are still running and stop them if necessary
    b. Tear down the network
    c. Unmount the shared memory for the pod

CRI-O: Under the Hood

Container Image Management in CRI-O

github.com/containers/image for image management (used by other projects too)

kubelet ensures that image gets pulled before starting the container

kubelet garbage collection removes unneeded images

available images on the node are used by the Kubernetes scheduler, too

CRI-O supports authentication file support for pulling from private registries:

```
[crio.image]
global_auth_file = "/path/to/config.json"
```

CRI-O: Under the Hood

Registry configuration possibilities in CRI-O

Container image locations can be defined as:

```
host[:port]/namespace[/namespace…]/repo(:tag|@digest)
```

Simply registry rewrites possible via `/etc/containers/registries.conf`:

```
[[registry]]
prefix = "example.com/foo"
location = "internal-registry-for-example.com/bar"
```

CRI-O: Under the Hood

Registry configuration possibilities in CRI-O

```
[[registry]]
location = "internal-registry-for-example.com/bar"
prefix = "example.com/foo"
insecure = false
blocked = false
mirror = [
    { location = "example-mirror-0.local/mirror-path" },
    { location = "example-mirror-1.local", insecure = true },
]
```

CRI-O: Under the Hood

Container Storage

multiple supported storage drivers: overlay, btrfs, vfs, devmapper, aufs, zfs

overlay driver still the recommended driver

all necessary CRI-O state is stored on disk

CRI-O uses github.com/containers/storage (used by other projects, too)

fine granular storage settings possible via `/etc/containers/storage.conf`

global container mounts via `/etc/containers/mounts.conf`

Thank you!

Mrunal Patel and Sascha Grunert