Container Platforms

What is available and which should I use?

Lucas Chaufournier

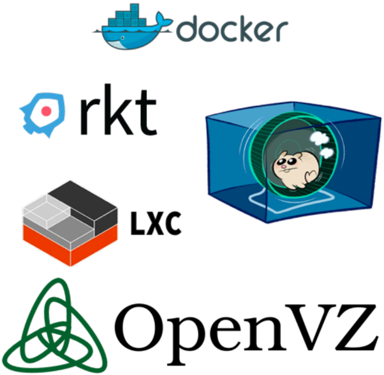
June 7, 2017

Table of Contents

- General Overview
  - Available Platforms
  - Docker
  - runc
  - rkt
  - LXC
- Using Containers
- cgroups and Containers

Available Platforms

- Docker
- runc
- rkt
- LXC/LXD
- Openvz



Docker

- Most well known platform
- Common container images are readily available for immediate use (i.e. Docker Hub)
- Meant to only run one process at a time.
- Good manageability tools
- Versioned /layered images
- Good Marketing and Branding

runc

- Most lightweight of all platforms
- Lots of manual configurations.
- Similar interface to docker for setting resource limits
- Relies on docker containers for creating rootfs
- Export docker rootfs and use json to setup a container.

rkt

- From CoreOS, designed to address some of the security concerns of docker.
- Use of prepackaged containers from `quay.io` or docker containers
- Works with kubernetes
- Uses Systemd for limited container resource management
- Pods act as a grouping of app images. Talk to each other over localhost. Similar setup to kubernetes pods.
- Pods mean that all apps in a pod are scheduled together on the same machine and configured with the same resources.

LXC

- From the team at ubuntu.
- Use preconfigured images from image store.
- Creating new images from scratch not as convenient as other platforms.
- LXC containers have an init process and can run multiple processes. Meant to be a machine container rather than an app container.
- Most similar to a VM.

Creating an Image

Docker

```
$ vim Dockerfile
{EDIT DOCKERFILE}
$ Docker build .
```

runc

```
$ mkdir /c1 && cd /c1
$ mkdir rootfs
$ docker export $(docker create busybox) | tar -C rootfs -xvf -
$ runc spec
$ runc create c1
```

rkt

```
$ acbuild begin
$ acbuild set-name example.com/hello
$ acbuild copy hello /bin/hellp
$ acbuild set-exec /bin/hello
$ acbuild write hello-latest-linux-amd64.aci
$ acbuild end
```

LXC

```
$ lxc-create -t download -n c1
```

Starting an Image

Docker

```
$ docker run -it ubuntu /bin/bash
$ docker run -d httpd
```

runc

```
# cd /c1 && runc run c1
# runc start c1
```

rkt

```
# rkt run --interactive quay.io/coreos/apline-sh
# rkt run --interactive docker://ubuntu --insecure-options=image
# systemd-run --slice=machine rkt run quay.io/coreos/alpine-sh
# rkt run example.com/app1 example.com/app2 ##Pod
```

LXC

```
$ lxc start c1 && lxc exec c1 -- /bin/bash
```

Stopping a Container

Docker

```
$ docker stop c4a5ec20a9ec
```

runc

```
$ runc kill c1 KILL
```

rkt

```
$ Does not exist
```

LXC

```
$ lxc stop c1
```

Deleting a Container

Docker

```
$ docker rm c4a5ec20a9ec
$ docker rmi httpd
```

runc

```
$ runc delete c1
```

rkt

```
$ rkt rm 203d0797
```

LXC

```
$ lxc delete c1
```

Mounting a volume

We are going to mount a host directory /data in the container as /data2.

Docker

```
$ docker run –it –v /data:/data2 ubuntu /bin/bash
```

runc

```
$ Does not exist
```

rkt:

```
$ rkt run c1 —volume logs,kind=host,source=/data —mount volume=logs,target=/data2
```

LXC

```
# lxc config device add c1 sdb disk source=/data path=data2
```

Setting cgroup Limits

We are going to set the maximum memory limits to 8gb. **Note:** You can always use manually cgroups interface for setting limits.

Docker

```
$ docker run -it   --memory=8g ubuntu /bin/bash
```

runc

```
$ runc update --memory 8000000000 c1
```

rkt

```
Use systemd config:
[Service]
Slice=machine.slice
MemoryLimit=8G
ExecStart=/usr/bin/rkt run --interactive docker://ubuntu --insecure-options=image --volu
ExecStopPost=/usr/bin/rkt gc --mark-only
KillMode=mixed
Restart=always
```

LXC

```
$ lxc config set c1 limits.memory 8g
```

Notes on cgroup interaction

So how does each interface with cgroups?

Docker and runc

- Docker runs atop runc.
- Relies on libcontainer from Docker.
- libcontainer provides all functionality for creating containers.
- libcontainer allows interfacing with cgroups using the native interface or systemd.

rkt

- Takes advantage of the systemd interface for cgroups.
- Relies on setting resource limits in a systemd service definition rather than implementing their own system.
- Only supports: cpu, cpuset and memory.

LXC

- Uses liblxc for interacting with cgroups.
- Filesystem limits can only be set when using zfs or btrfs.