Rocket and the Application Container Spec

Kelsey Hightower
CoreOS

Why

Why

Why

Why

Goals

- Provide an overview of the Application Container Spec (appc) and Rocket (rkt)
- Highlight where appc and the Docker image format agree and differ
- Demonstrate how to convert Docker images to ACIs
- Demonstrate how to sign and distribute ACIs
- Demonstrate how to deploy a complex application stack (Kubernetes) using rkt

Application Container Spec

A well-specified and community developed specification for application containers.

github.com/appc/spec (https://github.com/appc/spec)

- Image format (ACI)
- Discovery mechanism
- Runtime environment
- Tooling

Image Format (ACI)

An ACI contains all files and metadata needed to execute a given app.

- root file system
- image manifest

Image layout

```
$ tar -tf kube-apiserver-0.19.0-linux-amd64.aci
```

```
rootfs/kube-apiserver
manifest
```

Image Format (ACI)

Image manifest

```
{
    "acKind": "ImageManifest",
    "acVersion": "0.5.1",
    "name": "kube-apiserver",
    "labels": [
        {"name": "version", "value": "0.19.0"},
        {"name": "os", "value": "linux"}
    ],
    "app": {
        "exec": ["/kube-apiserver"],
        "user": "0",
        "group": "0",
        "mountPoints": [
            {
                "name": "volume-etc-ssl-certs",
                "path": "/etc/ssl/certs"
            }
        ]
    }
}
```

Discovery Mechanism

Translates an ACI name into a downloadable image.

```
https://rkt.io/kube-apiserver-0.19.0.aci
https://rkt.io/kube-apiserver-0.19.0.aci.asc
https://rkt.io/pubkeys.gpg
```

Simple Discovery

```
https://rkt.io/{name}-{version}.{ext}
```

Meta discovery

```
https://rkt.io
```

```
<head>
 <meta charset="utf-8">
 <meta name="ac-discovery" content="rkt.io/kube-apiserver https://rkt.io/{name}-{version}.{ext}">
 <meta name="ac-discovery-pubkeys" content="rkt.io/kube-apiserver https://rkt.io/pubkeys.gpg">
<head>
```

Runtime Environment

Defines how ACIs are executed.

- Filesystem Layout
- Volumes
- Networking
- Resource Isolators (cgroups)
- Logging

Tooling

actool

Build an ACI

```
$ actool build kube-apiserver-0.19.0-linux-amd64/ kube-apiserver-0.19.0-linux-amd64.aci
```

Extract and print an image manifest

```
$ actool cat-manifest -pretty-print kube-apiserver-0.19.0-linux-amd64.aci
```

Validate an image manifest

```
$ tar -xvf kube-apiserver-0.19.0-linux-amd64.aci manifest
$ actool validate -type=manifest manifest
```

Validate an ACI

```
$ actool validate -type=appimage kube-apiserver-0.19.0-linux-amd64.aci
```

Tooling

docker2aci

Small library and CLI tool to convert Docker images to ACI.

github.com/appc/docker2aci (https://github.com/appc/docker2aci)

Download and convert a Docker image to an ACI

```
$ docker2aci docker://quay.io/kelseyhightower/kube-apiserver:0.19.0
```

```
Downloading c6b09d8961e4: [==================================] 32 B/32 B
Downloading a30359211e41: [==================================] 7.87 MB/7.87 MB
Downloading ac615c26fbda: [==================================] 32 B/32 B
Downloading d59e6dd43c6c: [==================================] 32 B/32 B

Converted volumes:
    name: "volume-etc-kubernetes", path: "/etc/kubernetes", readOnly: false
    name: "volume-etc-ssl-certs", path: "/etc/ssl/certs", readOnly: false
    name: "volume-var-run-kubernetes", path: "/var/run/kubernetes", readOnly: false

Generated ACI(s):
kelseyhightower-kube-apiserver-0.19.0.aci
```

App Container Implementations

Libraries

- libappc - C++ library
- Nose Cone - Linux/C++
- appc - Go

Runtime environments

- Jetpack - FreeBSD Jails + ZFS
- Kurma - Container management and orchestration from Apcera
- rkt - systemd-nspawn + overlayfs

rkt (pronounced "rock-it")

rkt

A CLI for running app containers on Linux.

github.com/coreos/rkt (https://github.com/coreos/rkt)

- Swappable execution engines based on systemd or QEMU/KVM
- Docker Compatibility: rkt can run Docker images

First-class integration

- Init systems (systemd, upstart)
- Cluster orchestration tools (fleet, Kubernetes)

Pods

- Run one or more containers as a single unit
- Shared namespaces and volumes (optional)

rkt

Intel Clear Containers couples rkt with KVM execution engine.

clearlinux.org/clear-containers (https://clearlinux.org/features/clear-containers)

- secure containers boot in 150 milliseconds
- 18 to 20 MB memory overhead
- kvmtool skips BIOS jumps directly into the Linux kernel

rkt

Trust an image signing key

```
$ sudo rkt trust --root https://storage.googleapis.com/rktscience/pubkeys.gpg
```

```
Prefix: ""
Key: "https://storage.googleapis.com/rktscience/pubkeys.gpg"
GPG key fingerprint is: CDFF 0C6A EE50 D93A 5E71  A738 B6F7 807B 1EB4 DDAE
    Subkey fingerprint: 8FB7 603F 1238 E44C B127  6028 1F84 E96C 07B2 596F
        Rocket Science (ACI Builder) <release@rktscience.io>
Are you sure you want to trust this key (yes/no)? yes
Trusting "https://storage.googleapis.com/rktscience/pubkeys.gpg" for prefix "".
Added root key at "/etc/rkt/trustedkeys/root.d/cdff0c6aee50d93a5e71a738b6f7807b1eb4ddae"
```

rkt

Download and verify an ACI

```
$ sudo rkt fetch https://storage.googleapis.com/rktscience/kube-apiserver-0.19.0-linux-amd64.aci
```

List downloaded ACIs

```
$ sudo rkt images
```

```
KEY                                                             APPNAME
sha512-998cd0d20e7a3185425103ad4253622a21d6a937002094...        kubelet:0.19.0
sha512-0e7400d85814ca8fa827d184c950abc57fd0de215fc6bf...        kube-controller-manager:0.19.0
sha512-b78c03310fb49638ef89aa45691ccdba1192e4f6b74abf...        coreos.com/rkt/stage1:0.0.1
```

rkt

Launch a pod from an ACI

```
$ sudo rkt run \
--volume=volume-etc-kubernetes,kind=host,source=/etc/kubernetes \
--volume=volume-etc-ssl-certs,kind=host,source=/usr/share/ca-certificates \
--volume=volume-var-run-kubernetes,kind=host,source=/var/run/kubernetes \
https://storage.googleapis.com/rktscience/kube-apiserver-0.19.0-linux-amd64.aci -- \
--etcd-servers=http://127.0.0.1:2379 \
--logtostderr=true \
--service-cluster-ip-range=10.200.20.0/24
```

List pods

```
$ sudo rkt list
```

```
UUID        ACI                     STATE   NETWORKS
2131936c    kube-proxy              running
2a7aac55    kube-controller-manager running
54c545b8    kube-scheduler          running
7b27fb92    kubelet                 running
c712555c    kube-apiserver          running
```

rkt

Garbage collect old pods

```
$ sudo rkt list
```

```
UUID      ACI             STATE  NETWORKS
49d36db0  kube-apiserver  exited
54b38486  kube-apiserver  exited
e734dda3  kube-apiserver  exited
```

```
$ sudo rkt gc
```

```
Moving pod "49d36db0-3505-49c5-b1c4-f08215879d94" to garbage
Moving pod "54b38486-1d9a-4f20-a218-ac256752323a" to garbage
Moving pod "e734dda3-44ca-4c35-b564-0775eff6bc31" to garbage
```

```
$ sudo rkt gc --grace-period=10s
```

```
Garbage collecting pod "49d36db0-3505-49c5-b1c4-f08215879d94"
Garbage collecting pod "54b38486-1d9a-4f20-a218-ac256752323a"
Garbage collecting pod "e734dda3-44ca-4c35-b564-0775eff6bc31"
```

rkt

Systemd integration

```
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
Requires=etcd2.service
After=etcd2.service

[Service]
ExecStart=/usr/bin/rkt run \
 --volume=volume-etc-kubernetes,kind=host,source=/etc/kubernetes \
 --volume=volume-etc-ssl-certs,kind=host,source=/usr/share/ca-certificates \
 --volume=volume-var-run-kubernetes,kind=host,source=/var/run/kubernetes \
 https://storage.googleapis.com/rktscience/kube-apiserver-0.19.0-linux-amd64.aci -- \
 --etcd-servers=http://127.0.0.1:2379 \
 --service-cluster-ip-range=10.200.20.0/24
Restart=on-failure
RestartSec=5
```

```
$ sudo systemctl start kube-apiserver
```

Hands on with appc and rkt

Thank you

Kelsey Hightower
CoreOS
kelsey.hightower@coreos.com (mailto:kelsey.hightower@coreos.com)
@kelseyhightower (http://twitter.com/kelseyhightower)