The Open Container Initiative

Establishing standards for an open ecosystem

Jonathan Boulle

@baronboulle | jonathan.boulle@coreos.com

A short agenda

- Why do we care about standards?
- Where have container standards come from?
- Where are they now?
- Where are they going?

A short agenda

- **Why do we care about standards?**
- Where have container standards come from?
- Where are they now?
- Where are they going?

Standards: why do we care?

Why should *you* care?

CoreOS

Standards: why do we care?

Why should *you* care?
"You" as software developer or ops engineer

CoreOS

you

you as a sw engineer

your

```ada
with Ada.Text_IO;

procedure Hello_World is
 use Ada.Text_IO;
begin
    Put_Line("Hello, world!");
end;
```

```c
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
}
```

```go
package main

import "fmt"

func main() {
      fmt.Println("Hello, world!")
}
```

CoreOS

your

CoreOS

your

CoreOS

your

CoreOS

your

d474e8c57737625c

CoreOS

your

Signed By: Alice

d474e8c57737625c

CoreOS

you as a software engineer

A standard image format allows you to...

- **build** your container images how you want
- **distribute** them in a consistent, secure way under your control
- **re-use** other people's container images with whatever tooling you want
- **run** them anywhere that supports the format

CoreOS

you

CoreOS

you as an ops engineer

CoreOS

your

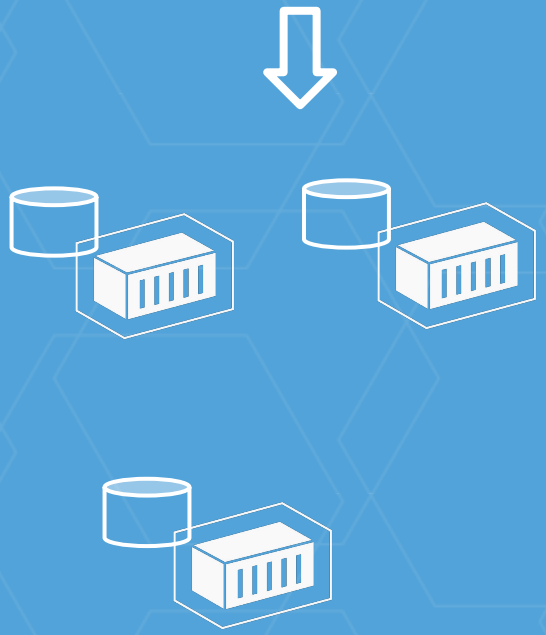
CoreOS

example.com/app
x3

your

CoreOS

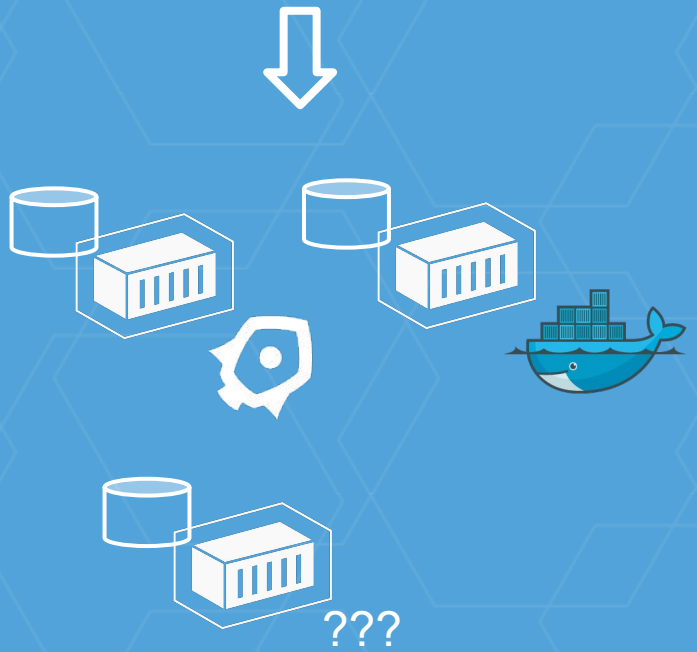
example.com/app
x3



your

CoreOS

example.com/app
x3



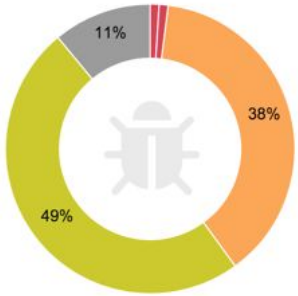
your

you as an ops engineer

A standard image format allows you to...

- **deploy** your developers' images + third-party images securely and consistently in the cluster
- use your **tooling of choice** to process and run container images
- **introspect** and **audit** container images

4f3f3b6e0b74



Quay Security Scanner has detected **100** vulnerabilities.

- **1** Critical-level vulnerabilities.
- **1** High-level vulnerabilities.
- **38** Medium-level vulnerabilities.
- **49** Low-level vulnerabilities.
- **11** Negligible-level vulnerabilities.

Image Vulnerabilities

Filter Vulnerabilities...

☐ **Only display vulnerabilities with fixes**

CVE	CVSS / SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN IMAGE
CVE-2014-9488	10	less	458-2	(None)	ADD file:9b5ba3935021955492697a...
CVE-2015-8391	9	pcre3	1:8.31-2ubuntu2.1	(None)	ADD file:9b5ba3935021955492697a...
CVE-2015-8380	7.5	pcre3	1:8.31-2ubuntu2.1	(None)	ADD file:9b5ba3935021955492697a...
CVE-2015-8472	7.5	libpng	1.2.50-1ubuntu2.14.04.1	1.2.50-1ubuntu2.14.04.2	ADD file:9b5ba3935021955492697a...
CVE-2015-8390	7.5	pcre3	1:8.31-2ubuntu2.1	(None)	ADD file:9b5ba3935021955492697a...

A short agenda

- **Why should we care about standards?**
- Where have container standards come from?
- Where are they now?
- Where are they going?

A short agenda

- Why should we care about standards?
- **Where have container standards come from?**
- Where are they now?
- Where are they going?

How did we get here?

The journey to OCI

How did we get here?

The *abbreviated* journey to OCI
(since ~2013)

First: an abbreviated history of containers

CoreOS

First: an abbreviated history of containers

where "containers" = "Linux containers"

CoreOS

Pre-2013: the early (Linux) container era

- Roll-your-own process containers
  - ulimit, cgroups, chroot, namespaces
  - more focused on noisy-neighbour problem

- LXC (since 2008)
  - powerful, complex (ops more than dev)
  - oriented more to "full-OS" containers

2013/14: dawn of application container age

Enter Docker

- easy-to-use, developer friendly
- popularised the application-centric container
- simple, centralised image distribution

CoreOS

2015/16: app containers go mainstream

FLATPAK

CoreOS

and the journey to standards

CoreOS

An image format
A container runtime
A log collection daemon
An init system and process babysitter
A container image build system
A remote management API

CoreOS

An image format

A container runtime

A log collection daemon

An init system and process babysitter

A container image build system

A remote management API

CoreOS

An image format

... the thing we want to standardise

Two years ago

Mid 2014

Docker Image Format Circa 2014

- Fluid format and evolution
  - No specification, just implementation
  - No guarantees of interoperability for other toolwriters
- Not content-addressable
  - No way to verify integrity or leverage CAS
- No name delegation/discovery (e.g. MX records)
  - Centralised/siloed distribution
- No mechanism for signing
  - No way to attest content

CoreOS

23 months ago

December 2014

CoreOS

appc

App Container (appc)

github.com/appc

appc motivation

- Write down what a container image is so anyone can build and run one
- Decompose the tooling and decentralise distribution
- Introduce features that were lacking in other container formats
- Two key areas: ***image format*** and ***runtime***

appc image in a nutshell

- **Image Format (ACI)**
  - what does an application consist of?
- **Image Discovery**
  - how can an image be located?
- **Content-addressability**
  - what is the cryptographic id of an image?
- **Signing**
  - how is an image signed and verified?

appc image (ACI) example

Simple tarball, containing root filesystem + configuration manifest

```
$ tar tf /tmp/my-app.aci
/manifest
/rootfs
/rootfs/bin
/rootfs/bin/my-app
```

```
{
  "acKind": "ImageManifest",
  "acVersion": "0.6.1",
  "name": "my-app",
  "labels": [
    {"name": "os", "value": "linux"},
    {"name": "arch", "value": "amd64"}
  ],
  "app": {
    "exec": [ "/bin/my-app" ],
    "user": "1000",
    "group": "1000"
  }
}
```

appc runtime in a nutshell

- **Application Container Executor (ACE)**
  - what environment can the application expect?
  - e.g. isolators (memory, CPU), network, etc
- **OS/Platform agnostic**
- **Pods**
  - Minimum execution unit (i.e. everything is a pod)
  - Grouping of applications with shared fate

appc in practice

- Diversity of image tooling
  - Build-from-scratch or build-from-language projects
    - shell scripts, `acbuild`, `dgr`, `goaci`
  - Convert from other packaging formats
    - `docker2aci`, `deb2aci`
- Diversity of runtimes
  - rkt (Linux)
  - Kurma (Linux)
  - Jetpack (FreeBSD)

19 Months Ago

April 2015

Docker v2.2 Image Format Circa 2015

- Versioned v2.0, v2.1, v2.2 schema
  - Still vendor-specific, but (mostly) documented!
- Content-addressable
- No name delegation/discovery
- Optional and separately-defined signing

Two separate worlds...

aka the "Container Wars"

- appc starting to see some traction, but conspicuously lacking Docker support
- Meanwhile, Docker image format gaining several of the key features that motivated appc

Two separate worlds...

- How can we end the "war" and all work together?

Two separate worlds...

- How can we end the "war" and all work together?
- Enter OCI!

17 Months Ago

June 2015

OPEN CONTAINER INITIATIVE

OPEN CONTAINER INITIATIVE

AN OPEN GOVERNANCE STRUCTURE FOR THE EXPRESS PURPOSE OF CREATING OPEN INDUSTRY STANDARDS AROUND CONTAINER FORMATS AND RUNTIME

CoreOS

Open Container Initiative (OCI)

- See: Chris's earlier talk :-)

- Original objective: merge everything we liked from appc, then deprecate appc in favour of OCI as the "true" standard
- However...

OCI Specification

- Originally only a *runtime* specification
  - What a container looks like on disk, just before it is run
  - A lot of system-specific state (e.g. mount/cgroup paths)
  - Not a portable, distributable format
  - Doesn't help with any of our earlier motivations

OCI Specification (runtime only)

- Several releases (v0.1.0 - v0.4.0)
- Continued disagreements and debate on scope of the project... until...

CoreOS

7 Months Ago

April 2016
(blog post)

OCI Image Format Spec Project

- A serialized, distributable image format
  - Content-addressable
  - Platform-agnostic
- Optional extras:
  - Signatures based on image content address
  - Federated, delegatable naming based on DNS

OCI Image Format Spec Project

- Backwards-compatible with Docker:
  - Taking the *de facto* standard Docker v2.2 format and writing it down for everyone to use
- Shared starting point for future innovation in container image format and distribution
- Intended to interoperate with Runtime Spec (similar to how appc defined both sections)

A short agenda

- Why should we care about standards?
- **Where have container standards come from?**
- Where are they now?
- Where are they going?

A short agenda

- Why should we care about standards?
- Where have container standards come from?
- **Where are they now?**
- Where are they going?

Today

15 November 2016

Today

15/11/2016?
11/15/2016?
2016年11月15日?

Today

~~15 November 2016~~

2016-11-15 (ISO 8601 standard)

OCI Today

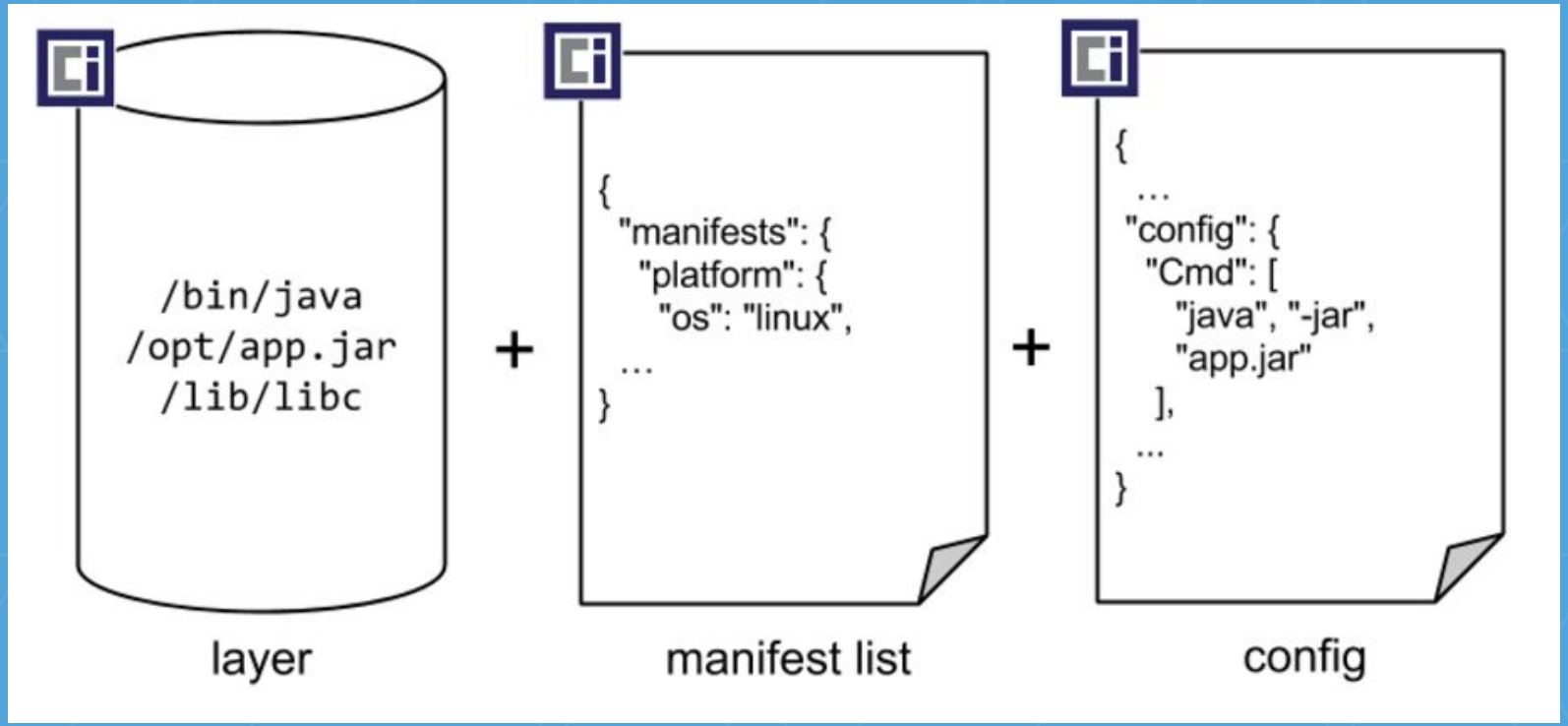
Two separate but connected specifications

- **image-spec**: what's in a container
- **runtime-spec**: how to run a container

OCI Image Spec

- Portable archive format
  - "The thing to distribute"
  - Structured tarball
- Image Manifest, Manifest List, and Config
  - Metadata about the container image
  - References to *layers*, containing root filesystem
- Cryptographic addressability
  - How to securely reference images and parts of images

Anatomy of an OCI Image



Inside the tarball

```
$ find busybox/
busybox/
busybox/refs
busybox/refs/latest
busybox/oci-layout
busybox/blobs
busybox/blobs/sha256
busybox/blobs/sha256/d09bddf0432...
busybox/blobs/sha256/56bec22e355...
busybox/blobs/sha256/e02e811dd08...
```

```
$ cat busybox/blobs/sha256/d09bddf043...
{
  "layers" : [
      {  "digest" : "sha256:56bec22e355981d...",
         "size" : 668151,
         "mediaType" : application/vnd.oci.image.layer.v1.tar+gzip"
      } ],
   "mediaType" : "application/vnd.oci.image.manifest.v1+json",
   "schemaVersion" : 2,
   "config" : {
      "digest" : "sha256:e02e811dd08fd49e7f6...",
      "mediaType" : "application/vnd.oci.image.config.v1+json",
      "size" : 1464
 }
```

OCI Runtime Spec

- On-disk layout of a container
  - Extracted root filesystem and configuration, ready to run
- Lifecycle verbs
  - `create, start, kill, delete, state`
- Multi-platform support
  - Shared general configuration
  - Windows/Solaris/Linux-specific bits

OCI Runtime Spec

Example: container state

```
{
    "ociVersion": "v1.0.0-rc2",
    "id": "oci-container1",
    "status": "running",
    "pid": 4422,
    "bundlePath": "/containers/redis",
    "annotations": {
        "myKey": "myValue"
    }
}
```

A short agenda

- Why should we care about standards?
- Where have container standards come from?
- **Where are they now?**
- Where are they going?

A short agenda

- Why should we care about standards?
- Where have container standards come from?
- Where are they now?
- **Where are they going?**

Where are we going?

- First things first: **1.0**
  - OCI Runtime Spec and OCI Image Spec 1.0
- Minimum viable product we can all agree on
- ETA: 1-2 months to finish release candidate process for both specifications

Where are we going?

- OCI Image Spec 1.0+
  - Image signatures
    https://github.com/opencontainers/image-spec/issues/400
    https://github.com/opencontainers/image-spec/issues/22
  - Image distribution
    https://github.com/opencontainers/image-spec/issues/15
  - Image dependencies
    https://github.com/opencontainers/image-spec/issues/102

Where are we going?

- OCI Runtime Spec 1.0+
  - Live container updates?
    https://github.com/opencontainers/runtime-spec/issues/17
    https://github.com/opencontainers/runtime-spec/issues/305
  - Virtualisation support?
    https://github.com/opencontainers/runtime-spec/pull/405

Where are we going?

Goal: *Standard container*

- Common image format and runtime format
- End user can just "`run example.com/app`"
- Identity and signing, discovery and naming, distribution all just work

Where are we going?

Goal: ***Enable innovation***

- Diverse ecosystem of tooling
- Build systems (CI integration, language integration)
- Runtimes (virtualisation technologies?)
- Distribution methods (torrents? IPFS?)
- Orchestration platforms (Kubernetes, Mesos, Nomad)

Where are we going?

Goal: *Ubiquity through organic adoption*

- Industry-standard in the container ecosystem
- Support in Kubernetes, Docker, Mesos, and more
- Magical world of interoperability!

Where are we going?

Join us!

- All OCI standards work happens in the open
- GitHub:
  - https://github.com/opencontainers/image-spec
  - https://github.com/opencontainers/runtime-spec
- Email:
  - dev@opencontainers.org

Jonathan Boulle

@baronboulle | jonathan.boulle@coreos.com | coreos.com

Thank you!

Extra/unused slides

appc specifications

- appc tried to define the application container story from the end-to-end UX perspective:
  - Users should be able to securely discover, download, and run an application container with a simple command-line (e.g. "`run example.com/app`")
- appc specifies two key areas:
  - *image format*
  - *runtime environment*

CoreOS

Image formats: a summarised history

	Docker v1	appc	Docker v2.2	OCI (in progress)
Introduced	2013	December 2014	April 2015	April 2016
Content-addressable	No	Yes	Yes	Yes
Signable	No	Yes, optional	Yes, optional	Yes, optional
Federated namespace	Yes	Yes	Yes	Yes
Delegatable DNS namespace	No	Yes	No	Yes

OCI: other things

- Reference runtime implementation (runc)
  - Widespread production use
  - Integral part of Docker and many others
- Nascent tooling for images and runtime
  - image-tools, runtime-tools projects