appc

App Container (appc)

github.com/appc

appc-dev@googlegroups.com

Jonathan Boulle

github.com/jonboulle
@baronboulle

CoreOS

App Container (appc)

- Announced December 2014
  - initially with rkt, but subsequently split out completely

- Umbrella project for application containers
  - eponymous specification
  - also: actool, acbuild, docker2aci, cni, …

- Project lives at github.com/appc

App Container Spec *noun*

An open specification for running applications in containers

Containers?!

Application Containers

Application Containers

self-contained, portable way to run software
(decoupled from operating system)

optionally isolated (memory, network, …)

Linux applications - pre-containers

KERNEL
SYSTEMD
SSH

PYTHON
JAVA
NGINX
MYSQL
OPENSSL

distro distro distro distro distro distro distro

APP

Linux applications - post-containers

KERNEL
SYSTEMD
SSH

LXC/DOCKER/RKT

distro distro distro distro distro distro distro distro

PYTHON
JAVA
NGINX
MYSQL
OPENSSL

APP

KERNEL
SYSTEMD
SSH

}

LXC/DOCKER/
RKT

Application Containers

aka "lightweight" virtualisation (but really, very different)

good for developers

- runs the same* on your laptop and in the cluster

good for administrators

- no need to manage software packages or give root access to pesky developers

appc motivation

Why are we doing this?

Back in late 2014...

- Multiple container tools (LXC, Docker, chroot)

- Widely used *de facto* standards, but various critical deficiencies (security, composability)

- No existing well-defined, open specification for application containers

Back in late 2014...

- We (CoreOS) decided to create new project to drive conversation around a standard

- "Write down what a container actually is"

- "Build once, sign once, run anywhere"

appc principles

What's important?

Composable

Integrate with existing systems
Non-prescriptive about build workflows
OS/architecture agnostic

Secure

Cryptographic image addressing
Image signing and encryption
Execution isolation
Container identity

Decentralized

Federated namespace without federated image hosting

Immutable labels: images are fully self-describing, globally addressable

Open

Independent GitHub organisation and governance policy
Contributors from a wide community

Simple but efficient

Simple to understand and implement, but eye to optimisation (e.g. content-based caching)

Standards-based

Well-known tools (tar, gzip, gpg, http), extensible with modern technologies (bittorrent, xz)

appc components

What's in the spec?

appc components

- Image Format
  - what does an application consist of?
- Image Discovery
  - how can an image be located?
- Pod
  - how can applications be grouped and run?
- Executor (runtime)
  - what does the execution environment look like?

Image Format

Application Container Image (.aci)
tarball of rootfs + manifest
uniquely identified by ImageID (hash)
(optionally) GPG-signed, AES-encrypted

Image Format (example)

```
{
    "acKind": "ImageManifest",
    "acVersion": "0.6.1",
    "name": "my-app",
    "labels": [
        {"name": "os",
         "value": "linux"},
        {"name": "arch",
         "value": "amd64"}
    ],
    "app": {
        "exec": [
            "/bin/my-app"
        ],
        "user": "0",
        "group": "0"
    }
}
```

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

Image Discovery

Resolves app name →artefact (.aci)

```
example.com/http-server
coreos.com/etcd
```

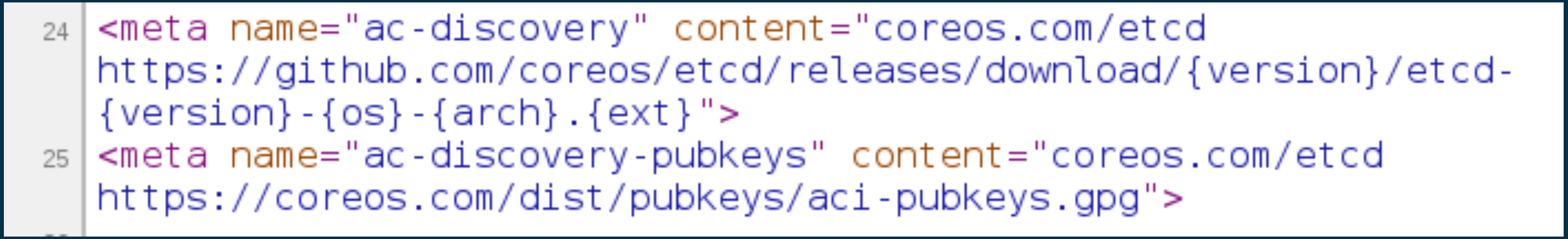
DNS + HTTPS + HTML meta tags

Image Discovery (example)

Discover the image "coreos.com/etcd"

→ `https://coreos.com/etcd`

→ inspect returned HTML for meta tags:



```
<meta name="ac-discovery" content="coreos.com/etcd
https://github.com/coreos/etcd/releases/download/{version}/etcd-
{version}-{os}-{arch}.{ext}">
<meta name="ac-discovery-pubkeys" content="coreos.com/etcd
https://coreos.com/dist/pubkeys/aci-pubkeys.gpg">
```

→ substitute template and retrieve image/sig:

```
https://github.com/coreos/etcd/releases/download/2/etcd-2-linux-x86_64.aci
https://github.com/coreos/etcd/releases/download/2/etcd-2-linux-x86_64.asc
```

→ retrieve public keyring:

```
https://coreos.com/dist/pubkeys/aci-pubkeys.gpg
```

Pods

grouping of multiple applications
(templated or deterministic)

shared execution context
(namespaces, volumes)

Pods (example)

```
...
   "acKind": "PodManifest",
   "apps": [
       { "name": "reduce-worker",
         "image": { "name": "example.com/reduce-worker" },
       },
       { "name": "backup",
         "image": { "name": "example.com/worker-backup",
         "isolators": [
              { "name": "resource/memory", "value": {"limit": "4G"} }
         ]
       },
    ]
   "isolators": [
       { "name": "resource/memory", "value": {"limit": "4G"} }
   ]
...
```

Executor

runtime environment
isolators, networking, lifecycle
metadata service

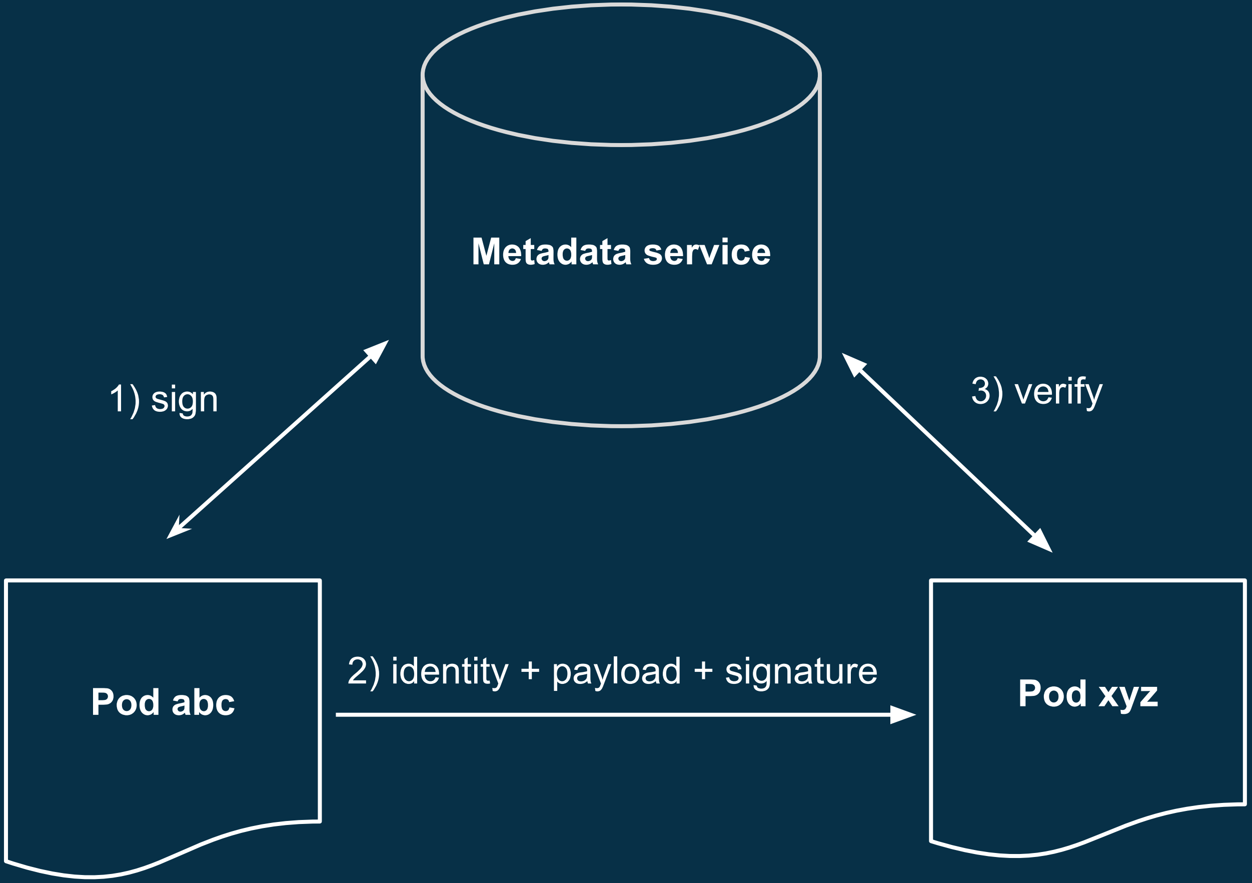
Metadata Service

`http://$AC_METADATA_URL/acMetadata`

container metadata

container identity (HMAC verification)

Metadata Service



Metadata service

1) sign

3) verify

Pod abc

2) identity + payload + signature

Pod xyz

appc today

What's the current status?

appc today

- 9 months since initial announcement
  - v0.8.0(ish) imminent
- three mature(ish) implementations
  - variety of other tooling (in various stages)
- new networking spec
  - CNI: Container Networking Interface
- appc vs OCI
  - another container standard??

appc implementations

appc implementations

coreos/rkt

- First, most complete implementation
- Go/Linux, cgroups/namespaces, lkvm

3ofcoins/jetpack

- FreeBSD Jails/ZFS-based executor
- Go, uses upstream appc schema code

apcera/kurma

- Recursive init system - Linux+kurma
- Go/Linux

appc implementations (cont)

- Various tooling:
  - github.com/sgotti/baci
  - github.com/justinsb/appc-java
  - github.com/cdaylward/libappc
  - github.com/klizhentas/deb2aci
  - github.com/appc/docker2aci
  - github.com/appc/goaci
- Mesos (partial, under development)
- Kubernetes (indirectly, via rkt)

CNI

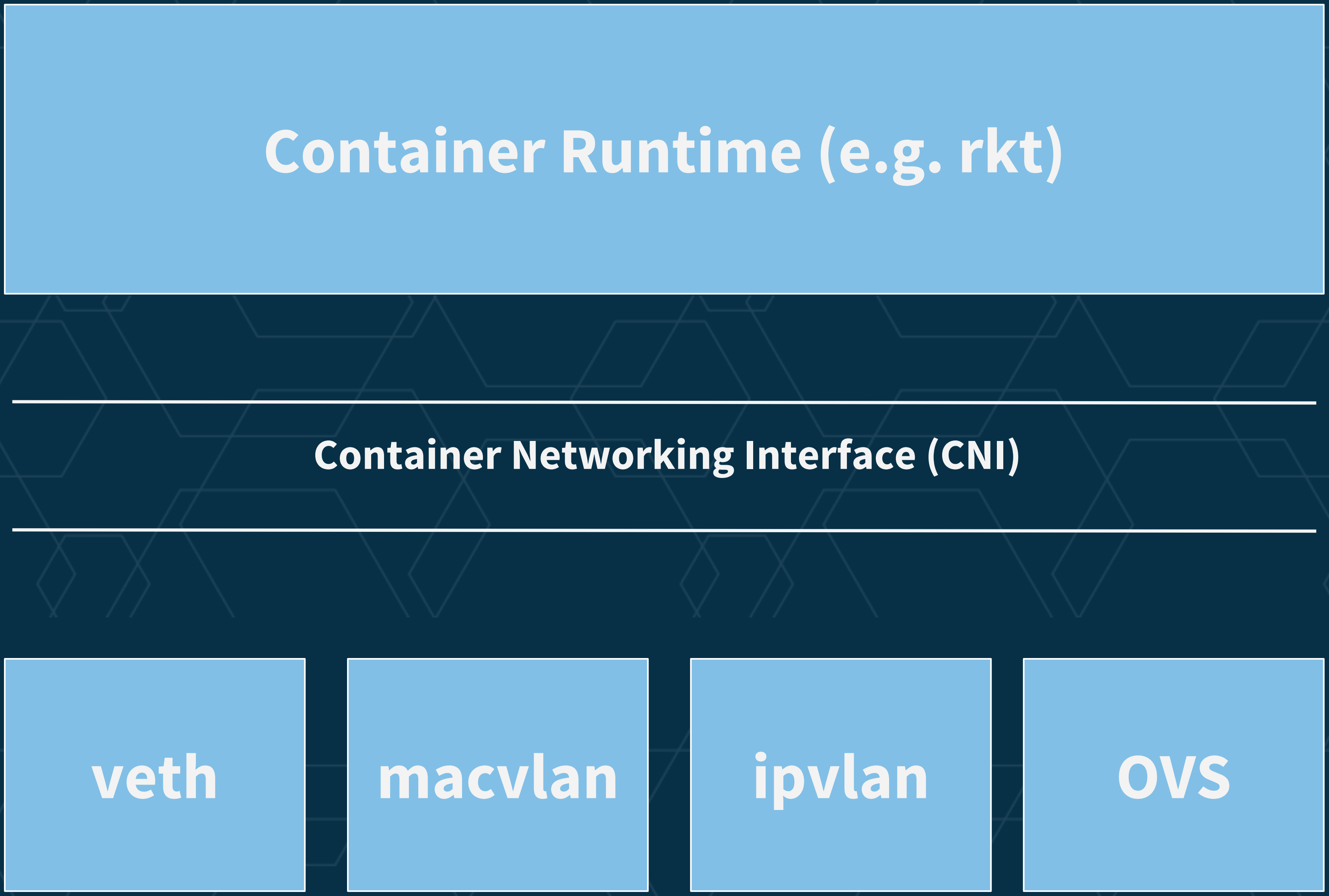
Container Networking Interface

CNI - Container Networking Interface

- Every runtime will attempt to solve networking
  - CNI emerged out of rkt's networking code

- Simple CLI interface for plugins
  - add container to network
  - delete container from network
  - L3 - allocate/maintain IP addresses
    - IPAM plugin

- Distinct from appc spec



Container Runtime (e.g. rkt)

Container Networking Interface (CNI)

veth

macvlan

ipvlan

OVS

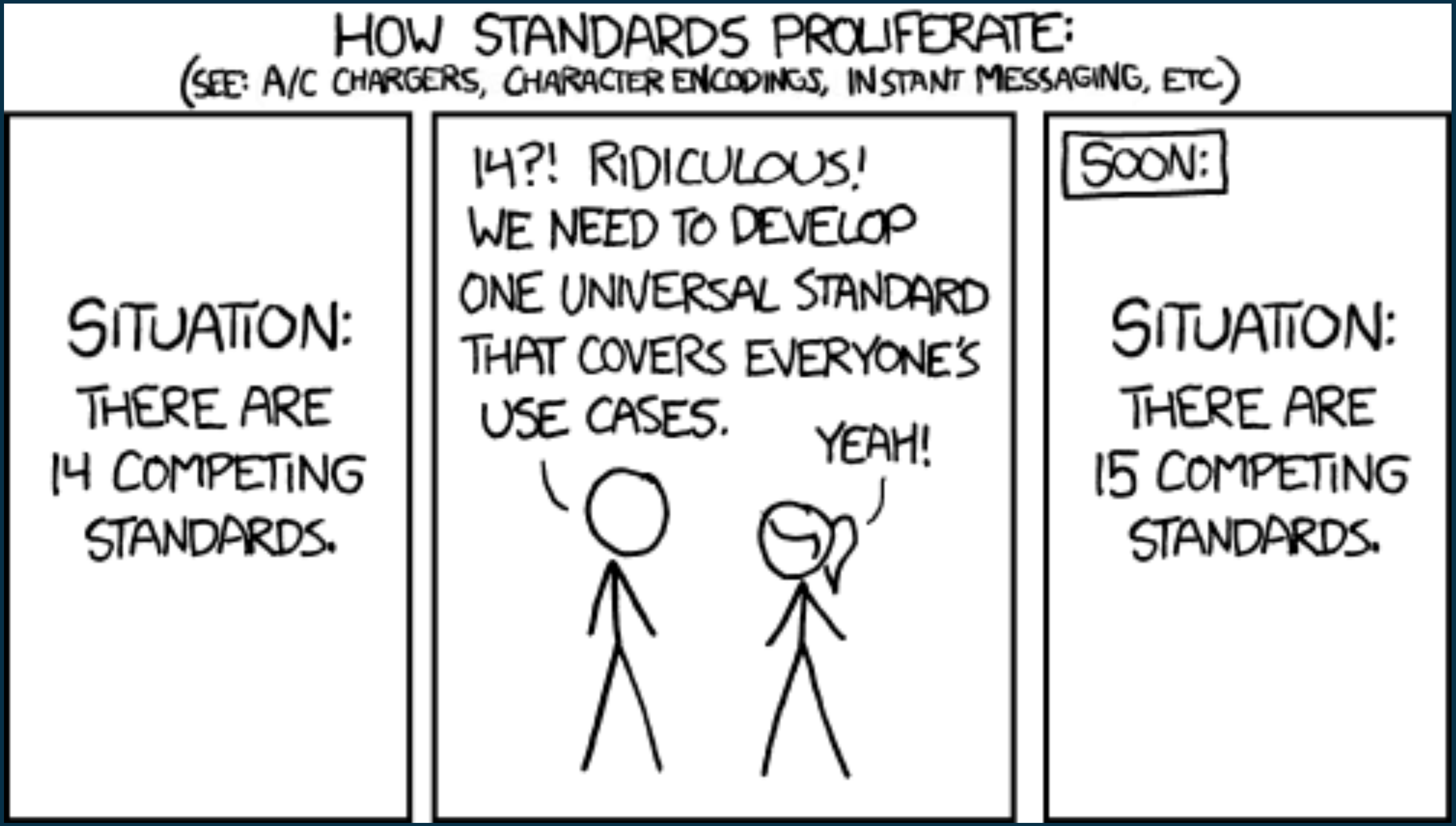
CNI - Container Networking Interface

- Currently used by rkt
- docker-run.sh wrapper script
- Exploring integration with other systems
  - weave
  - kubernetes
- For more details, see Eugene talk at 4pm on Wednesday! http://sched.co/3Xol

appc and OCI

aka https://xkcd.com/927

appc and OCI


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Standards / Randall Munroe / CC BY-NC 2.5

OCI - Open Containers Initiative

- Announced June 2015 (as OCP)
- Lightweight, open governance project
- Linux Foundation
- Container runtime format
  - configuration on disk, execution environment
- Runtime implementation (runc)

appc vs OCI

appc

- image format
- runtime environment
- pods
- image discovery

OCI

- runtime format
- runtime environment

appc vs OCI

appc runtime

- environment variables
- Linux device files
- hooks
- etc...

- descriptive
- multiple apps

OCI runtime

- environment variables
- Linux device files
- hooks
- etc...

- prescriptive
- single app (process)

appc and OCI

tl;dr: complementary
still evolving

appc lessons learned

multiple implementations are *very* useful

"one honking great idea"

multiple implementations

- spec initially developed in tandem with just rkt
- other implementations provide critical feedback:
  - areas that need to be clarified
  - things to be fixed for cross-platform
- leverage common code to improve all implementations
- define line between spec and implementation

usability matters

aka, prescriptivism is sometimes nice
aka, users (rightly) want easy-to-use tools

usability matters

example: building ACIs vs Docker images

- motivation with appc was to integrate with existing build systems, provide flexibility
- Dockerfile is opinionated, prescribed workflow
- tools like acbuild are much less accessible for developers
- many potential users are turned away

usability matters

example: appc executors vs Docker

rkt

- + power features like signature validation built-in from day 1
- - no focus on easy use like boot2docker

kurma

- + immensely powerful, building block for complex orchestration
- - steep learning curve, bleeding edge

explicit is better than implicit

aka, initial perception is critical
aka, show your work

explicit is better than implicit

example: image format

- canonical form is a single file (tarball)
- designers intended to optimise by storing exploded on disk and indexing manifests
- unfortunately, non-obvious to many approaching the spec

initial perception is critical

example: composability

- Initially, monolithic spec containing definitions of all components, for readability/context

- Lot of (indirect) feedback about spec being "too big", "too complicated to implement"

- To solve, split spec into distinct, discrete sections while clearly discussing interlinks

takeaway: show your work

appc is the result of extensive thought, experimentation, discussion...

...but this isn't always obvious in the text

"why are things the way they are?"

- "hmm, I don't quite remember.."
- revisit a lot of old discussions

Mix RFC-style spec and user stories/examples

appc tomorrow

What's next?

appc tomorrow

- Development still active!
    - Push towards 1.0
- Better build tooling
    - github.com/appc/acbuild
- Production-ready implementations!
    - rkt 1.0 Real Soon Now™
- Continue harmonisation effort with OCI

get involved!

github.com/appc

appc-dev@googlegroups.com

Questions?