

%title: CoreOS Fest Berlin 2016 - rkt for Platform Builders

%author: @lukeb0nd

%date: 2016-05-09

```
\_--  
_|||_|  
|'|/|_| |
|||<|_|  
|||\\|_|
```

-> for Platform Builders <-

-> # CoreOS Fest Berlin <-

-> ## May 9-10 2016 <-

-> Luke Bond <-

-> @lukeb0nd <-

IN THIS TALK

- Some history
- Some opinions about rkt and Docker in relation to building platforms with them
- A practical / demo section on how to use rkt

This is essentially "rkt 101" with a few of my opinions thrown in.

WHO AM I?

- I'm a backend developer, DevOps-curious
- Mostly I do Node.js and Docker
- Built some things with Docker on CoreOS
 - Including a thing called "Paz" - [<http://paz.sh>]

I work for YLD.io, a London-based software engineering consultancy that specialises in Node.js, Docker and React.

HISTORY - ORIGINS OF RKT

- CoreOS felt Docker was no longer "the simple composable building block [they] had envisioned"
- rkt announced alongside the appc set of specifications
 - Specifications for runtime, image and discovery
- Focus was on the composability, security, image distribution, and openness of containers
- Docker's daemon and monolithic CLI tool make composability a problem

- Leading to wrapping the Docker CLI by some tool makers (pre plugins)
-

HISTORY - STANDARDS ETC.

- Open Container Initiative (initially the Open Container Project) launched on June 22nd, 2015
- Docker's container image format and runtime donated to the project
- Appc shook things up enough to get Docker to commit to open standards
- Progress on standards appeared slow initially, and limited to the runtime
 - Recently work on the image and distribution has begun
 - See @vbatts' talk later today for an update

For a more detailed coverage of this history, check out [this talk](https://skillsmatter.com/skillscasts/7443-introduction-to-rkt-luke-bond) (<https://skillsmatter.com/skillscasts/7443-introduction-to-rkt-luke-bond>).

DOCKER & RKT

- rkt doesn't have a daemon architecture like Docker does
 - Docker's daemon architecture is a key differentiator for Docker
 - This difference is what makes rkt great for building platforms
-

THE DOCKER DAEMON

- The Docker daemon model makes for a great integrated solution if you don't mind lock-in
 - Docker's lack of composability a limiting factor for platform-builders
 - I know this from building Paz
 - Those building platforms on Docker's tools are in a conundrum
 - Go all in on Compose & Swarm and have no differentiating tech
 - Build only on the daemon (i.e. build your own multi-host networking), and suffer lack of control over processes
 - This is also about who owns PID1- systemd, Docker, ... unikernels?
-

RKT FOR PLATFORM BUILDERS

This is what makes rkt so attractive to platform builders.

There are really two buckets of users for Rocket and they could both be considered "platform builders."

The first set of platform builders are companies like Cloud Foundry, Mesosphere, or cloud service providers (Amazon Web Services, Google, Rackspace) that are building a platform as their product. Rocket allows them to add containers to their platform while keeping the rest of what they do today.

The second set are enterprises that already have an existing environment and want to add containers to it. These would typically be large enterprises that have already invested in their own internal platform and want to layer in containers.

-> -- Alex Polvi, CoreOS, 7/1/2015 on readwrite.com <-

HISTORY - THE CONTAINER WARS

- Docker is [making a platform play \(https://blog.docker.com/2014/09/docker-closes-40m-series-c-led-by-sequoia/\)](https://blog.docker.com/2014/09/docker-closes-40m-series-c-led-by-sequoia/).

My understanding is that *the Docker Platform will be a choice for companies that want vSphere for containers*— that is, they want a whole platform off the shelf.

-> -- Alex Polvi, CoreOS, 7/1/2015 on readwrite.com <-

RKT FOR PLATFORM-BUILDING

- Tools should follow the Unix philosophy
- The Docker monolith does too much; as root
- rkt is composable
- Much of Docker's functionality is done well by other tools
 - Linux init systems like systemd manage processes perfectly well
 - tar/gzip are fine for packaging and distribution
 - etc.

RKT IN PRACTICE

- rkt is a container runtime, not an image build tool
- rkt started out as an appc reference implementation
- Therefore it natively runs appc images:

acbuild is a command line utility to build and modify App Container Images (ACIs), the container image format defined in the App Container (appc) spec.

acbuild commands are analogous to lines of a Dockerfile.

A shell script of acbuild commands is the appc equivalent of a Dockerfile.

Let's try an example.

RKT IN PRACTICE

Building Images with acbuild

Let's take the most basic of example applications:

```
$ cat hello.c
#include <stdio.h>
int main (int argc, char** argv) {
    printf("Hello, world!\n");
    return 0;
}
$ gcc -o hello -static hello.c
```

RKT IN PRACTICE

We could use the following script to make an appc image:

```
$ cat appc-hello.sh
#!/usr/bin/env bash
acbuild begin
acbuild set-name hello
acbuild copy hello /app/hello
acbuild set-working-directory /app
acbuild set-exec -- /app/hello
acbuild write --overwrite hello-latest-linux-amd64.aci
```

Run the build script to build the image:

```
$ ./appc-hello.sh
$ ls -l hello-latest-linux-amd64.aci
-rw-r--r-- 1 luke users 331282 May  6 01:08 hello-latest-linux-amd64.aci
```

RKT IN PRACTICE

Launching Containers

Let's launch the container:

```
$ sudo rkt --insecure-options=image run hello-latest-linux-amd64.aci
image: using image from local store for image name coreos.com/rkt/stage1-
coreos:1.4.0
image: using image from file hello-latest-linux-amd64.aci
networking: loading networks from /etc/rkt/net.d
networking: loading network default with type ptp
[28959.906955] hello[4]: Hello, world!
```

As in Docker, the container is still present after execution has completed:

```
$ sudo rkt list
UUID          APP      IMAGE NAME  STATE   CREATED          STARTED          NETWORKS
3fa4bc21     hello    hello      exited 3 minutes ago 3 minutes ago
```

RKT IN PRACTICE

Housekeeping

Cleaning up completed containers is simple:

```
$ sudo rkt gc
```

Similarly for images that are no longer needed:

```
$ sudo rkt image gc
```

RKT IN PRACTICE

More Examples

Let's run something with networking:

```
$ git clone https://github.com/lukebond/demo-api.git
$ cd demo-api
$ sudo ./appc.sh
$ sudo rkt --insecure-options=image run demo-api-latest-linux-amd64.aci
image: using image from local store for image name coreos.com/rkt/stage1-
coreos:1.4.0
image: using image from file demo-api-latest-linux-amd64.aci
image: searching for app image quay.io/coreos/alpine-sh
image: remote fetching from URL "https://quay.io/c1/aci/quay.io/coreos/alpine-
sh/latest/aci/linux/amd64/"
Downloading ACI: [=====] 2.65 MB/2.65
MB
networking: loading networks from /etc/rkt/net.d
networking: loading network default with type ptp
```

RKT IN PRACTICE

In another terminal, check its status and make a request to it:

```
$ sudo rkt list
UUID          APP          IMAGE NAME          STATE    CREATED          STARTED
NETWORKS
3fa4bc21      hello        hello                exited  17 minutes ago   17 minutes ago
5f9095eb      demo-api     lukebond/demo-api    running  52 seconds ago   52 seconds ago
default:ip4=172.16.28.9
$ curl 172.16.28.9:9000
"Hello, world 172.16.28.9!"
```

Success!

RKT IN PRACTICE

Logs

To access the logs for our containers, we use systemd's journalctl, like so:

```
$ machinectl list
MACHINE                                CLASS    SERVICE
rkt-5f9095eb-5ad0-4cb1-b54f-729b1a3b3217 container nspawn

1 machines listed.
$ sudo journalctl -M rkt-5f9095eb-5ad0-4cb1-b54f-729b1a3b3217
-- Logs begin at Fri 2016-05-06 01:37:58 BST, end at Fri 2016-05-06 01:37:5
...
```

RKT IN PRACTICE

Stopping Containers

Killing containers is a bit of a hassle at the moment:

```
$ machinectl kill rkt-5f9095eb-5ad0-4cb1-b54f-729b1a3b3217
```

This is because the implementation we're using leverages systemd-nspawn to launch containers.

A native rkt command for stopping containers is reportedly coming in a future release.

RKT IN PRACTICE

Signing images

- The `--insecure-options=image` argument to `rkt run` disables signature verification
 - But signing is important to ensure the artefact hasn't been tampered with
 - Signing images is easily done using standard gpg tools.
 - Instructions can be found [here \(https://coreos.com/rkt/docs/0.5.4/signing-and-verification-guide.html\)](https://coreos.com/rkt/docs/0.5.4/signing-and-verification-guide.html).
-

RKT IN PRACTICE

Converting Docker Images

The appc tool `docker2aci` can be used to download Docker images and convert them to appc's ACI format. Get the tool [here \(https://github.com/appc/docker2aci\)](https://github.com/appc/docker2aci).

Converting a Docker image is as simple as:

```
$ docker2aci docker://lukebond/demo-api
```

It will squash the Docker layers into one ACI file.

To keep them separate, pass `--nosquash`; it will set the correct dependencies between the layers.

RKT IN PRACTICE

Running Docker Images Directly

rkt can also run Docker images directly:

```
$ sudo rkt run --insecure-options=image docker://lukebond/demo-api
```

The "insecure" option is required here because Docker doesn't support the same image signature verification that rkt does.

Under the hood this is simply using `docker2aci` to first convert to ACI.

RKT IN PRACTICE

Image Discovery and Distribution

- Image discovery in rkt follows the appc spec
- Whereas Docker uses a registry (defaulting to the Docker Hub), rkt uses a combination of HTTPS and HTML meta tags via a discovery URL
- This is best illustrated by an example (for CoreOS' Etcd):

```
$ curl -sL https://quay.io/coreos/etcd | grep meta | grep discovery
```

- The content attributes are templated locators

For a full run-down of how appc image discovery works, [read the specification \(https://github.com/appc/spec/blob/master/spec/discovery.md\)](https://github.com/appc/spec/blob/master/spec/discovery.md).

PODS

- The term "pod" is was popularized by the Kubernetes project
- A pod is a collection of applications should be scheduled as a unit
- The appc spec defines a pod as:

"the deployable, executable unit...a list of apps that will be launched together inside a shared execution context, which includes network configuration and isolators. Whether you're running one process or multiple, rkt still considers it a pod.

- Pods are 1st-class citizens in rkt
 - So far we've been talking about running containers; really we were running pods
-

PODS

Let's try a networked example; a trivial extension of demo-api with a Redis counter.


```
$ git clone https://github.com/lukebond/demo-api-redis.git
$ cd demo-api-redis
$ sudo ./appc.sh
$ sudo rkt run --volume volume--var-lib-redis,kind=host,source=/var/lib/redis
quay.io/quay/redis \
  --insecure-options=image --port=http:9000 --set-env REDIS_HOST=localhost \
  demo-api-redis-latest-linux-amd64.aci
```

This will launch one pod, which contains two processes- Redis and a demo app, mapping necessary ports and volumes.

PODS

```
$ sudo rkt list
UUID          APP          IMAGE NAME          STATE    CREATED
STARTED      NETWORKS
e16bafd0 redis        quay.io/quay/redis:latest running 6 seconds ago 6
seconds ago default:ip4=172.16.28.6
demo-api-redis lukebond/demo-api-redis
$ curl 172.16.28.6:9000
"Hello, world 172.16.28.6! 1 hits."
$ curl 172.16.28.6:9000
"Hello, world 172.16.28.6! 2 hits."
$ curl 172.16.28.6:9000
"Hello, world 172.16.28.6! 3 hits."
```

Success!

CONCLUSION

- I hope that has brought you up to speed with the basics of rkt
 - If you're building a container-based platform, I recommend using rkt
 - Docker is great but it will make your life harder for this use case
-

FURTHER READING

[Latest rkt features, including "fly" \(https://coreos.com/blog/rkt-0.15.0-introduces-rkt-fly.html\)](https://coreos.com/blog/rkt-0.15.0-introduces-rkt-fly.html)
[rkt announcement - CoreOS blog \(https://coreos.com/blog/rocket/\)](https://coreos.com/blog/rocket/)
[Open Container Project announcement - Docker blog \(https://blog.docker.com/2015/06/open-container-project-foundation/\)](https://blog.docker.com/2015/06/open-container-project-foundation/)
[Weaveworks' Analysis of the OCP announcement - Weaveworks blog \(http://www.weave.works/docker-open-container-project-please-make-it-awesome/\)](http://www.weave.works/docker-open-container-project-please-make-it-awesome/)
[CoreOS' co-announcement of the OCP - CoreOS blog \(https://coreos.com/blog/app-container-and-the-open-container-project/\)](https://coreos.com/blog/app-container-and-the-open-container-project/)
[OCP Progress report including name change to OCI - Docker blog \(http://blog.docker.com/2015/07/open-container-format-progress-report/\)](http://blog.docker.com/2015/07/open-container-format-progress-report/)

[CoreOS on OCI, appc and other standards - CoreOS blog \(https://coreos.com/blog/making-sense-of-standards/\)](https://coreos.com/blog/making-sense-of-standards/)

[My rkt article on the Codeship blog \(https://blog.codeship.com/getting-started-rkt/\)](https://blog.codeship.com/getting-started-rkt/)

[OCI Image Spec project announcement](#)

[\(https://www.opencontainers.org/news/blogs/2016/04/new-image-specification-project-container-images\)](https://www.opencontainers.org/news/blogs/2016/04/new-image-specification-project-container-images)

[OCI Image Spec project on the CoreOS blog \(https://coreos.com/blog/oci-image-specification.html\)](https://coreos.com/blog/oci-image-specification.html)

[My talk at Docker London re rkt \(https://skillsmatter.com/skillscasts/7443-introduction-to-rkt-luke-bond\)](https://skillsmatter.com/skillscasts/7443-introduction-to-rkt-luke-bond)