Container mechanics in Linux and rkt

FOSDEM 2016
a modern, secure, composable container runtime
an implementation of appc
(image format, execution environment)
rkt

simple CLI tool
golang + Linux
self-contained
simple CLI tool

no (mandatory) daemon:
apps run directly under spawning process
rkt internals

modular architecture
execution divided into stages
stage0 → stage1 → stage2
bash/runit/systemd/...  (invoking process)

rkt  (stage0)

pod  (stage1)

app1  (stage2)

app2  (stage2)
bash/runit/systemd/...  (invoking process)

rkt  (stage0)

pod  (stage1)

app1  (stage2)

app2  (stage2)
stage0 (rkt binary)

discover, fetch, manage application images
set up pod filesystems
commands to manage pod lifecycle
bash/runit/systemd/... (invoking process)

- rkt (stage0)
  - pod (stage1)
    - app1 (stage2)
    - app2 (stage2)
stage1

"the container"
execution environment for pods
process lifecycle management
resource constraints (isolators)
stage1 (swappable)

- binary ABI with stage0
  - rkt's stage0 calls exec(stage1, args...)

- default implementation
  - based on systemd-nspawn + systemd
  - Linux namespaces + cgroups for isolation

- kvm implementation
  - based on lkvm + systemd
  - hardware virtualisation for isolation
bash/runit/systemd/... (invoking process)

- rkt (stage0)
  - pod (stage1)
    - app1 (stage2)
    - app2 (stage2)
bash/runit/systemd/... (invoking process)

- rkt (stage0)
- systemd-nspawn (stage1)
  - systemd
    - cached (stage2)
    - workerd (stage2)
bash/runit/systemd/... (invoking process)

- rkt (stage0)
- systemd-nspawn (stage1)
  - systemd
    -cached (stage2)
    - workerd (stage2)

container
Containers on Linux

namespaces
cgroups
chroot
Linux namespaces
containers

hostname: thunderstorm

hostname: rainbow

hostname: sunshine

host
Containers: no guest kernel

system calls:
example: sethostname()
Containers with an example

Getting and setting the hostname:

- The system calls for getting and setting the hostname are older than containers

```c
int uname(struct utsname *buf);
int gethostname(char *name, size_t len);
int sethostname(const char *name, size_t len);
```
Processes in namespaces

gethostname() -> "rainbow"  gethostname() -> "thunderstorm"
Linux Namespaces

Several independent namespaces

- uts (Unix Timesharing System) namespace
- mount namespace
- pid namespace
- network namespace
- user namespace
Creating new namespaces

```
unshare(CLONE_NEWUTS);
```

“rainbow”
Creating new namespaces

"rainbow"
# unshare --uts

# hostname sunshine
# hostname
sunshine
# exit
logout
#
# hostname
thunderstorm
#

PID namespace
Hiding processes and PID translation

- the host sees all processes
- the container only its own processes
Hiding processes and PID translation

- the host sees all processes
- the container only its own processes

```
Terminal

1 root   0:00 /usr/lib/systemd/systemd
2 root   0:00 /usr/lib/systemd/systemd
4 root   0:00 /bin/sh -c "sh"
5 root   0:00 sh
21 root  0:00 ps aux
```

Actually pid 30920
Initial PID namespace
Creating a new namespace

clone(CLONE_NEWPID, ...);
Creating a new namespace
**rkt**

**rkt run ...**

✦ uses clone() to start the first process in the container with a new pid namespace
✦ uses unshare() to create a new network namespace

**rkt enter ...**

✦ uses setns() to enter an existing namespace
Joining an existing namespace

setns(..., CLONE_NEWPID);
Joining an existing namespace
Mount namespaces
container

/home
/var
/etc
/my-app

host

user
Storing the container data (Copy-on-write)

Container filesystem

Overlay fs “upper” directory
/var/lib/rkt/pods/run/<pod-uuid>/overlay/sha512- ...

Application Container Image
/var/lib/rkt/cas/tree/sha512-...
rkt directories

/var/lib/rkt
  └── cas
    └── tree
      └── deps-sha512-19bf...
      └── deps-sha512-a5c2...
  └── pods
    └── run
      └── e0ccc8d8
        └── overlay/sha512-19bf.../upper
        └── stage1/rootfs/
unshare(...., CLONE_NEWNS);
Changing root with MS_MOVE

```
mount($ROOTFS, "/", MS_MOVE)
```

$ROOTFS = /var/lib/rkt/pods/run/e0ccc8d8.../stage1/rootfs
Mount propagation events

Relationship between the two mounts:
- shared
- master / slave
- private
Mount propagation events

Private  |  Shared  |  Master and slave

/home  |  /home  |  /home

/home  |  /home  |  /home

/home  |  /home  |  /home
How rkt uses mount propagation events

* / in the container namespace is recursively set as slave:

```c
mount(NULL, "/", NULL,
    MS_SLAVE|MS_REC, NULL)
```
Network namespace
Network isolation

Goal:

✦ each container has their own network interfaces
✦ Cannot see the network traffic outside the container (e.g. tcpdump)
Network tooling

- Linux can create pairs of virtual net interfaces
- Can be linked in a bridge
rkt networking

- plugin based
- Container Network Interface (CNI)
- rkt
- Kubernetes
- Calico
Container Runtime (e.g. rkt)

Container Networking Interface (CNI)

veth  macvlan  ipvlan  OVS
How does rkt do it?

- rkt uses the network plugins implemented by the Container Network Interface (CNI, https://github.com/appc/cni)

```
/var/lib/rkt/pods/run/POD_UUID/netns
```
User namespaces
History of Linux namespaces

✓ 1991: Linux
✓ 2002: namespaces in Linux 2.4.19
✓ 2008: LXC
✓ 2011: systemd-nspawn
✓ 2013: user namespaces in Linux 3.8
✓ 2013: Docker
✓ 2014: rkt

... development still active
Why user namespaces?

✦ Better isolation
✦ Run applications which would need more capabilities
✦ Per user limits
✦ Future:
  ✦ Unprivileged containers: possibility to have container without root
User ID ranges

4,294,967,295
(32-bit range)

0 65535

host

container 1

container 2
User ID mapping

/proc/$PID/uid_map: “0 1048576 65536”
Problems with container images

Web server

Application Container Image (ACI)

Downloading

Container 1
- Container filesystem
- Overlayfs "upper" directory

Container 2
- Container filesystem
- Overlayfs "upper" directory

Application Container Image (ACI)
Problems with container images

- Files UID / GID
- rkt currently only supports user namespaces without overlayfs
  - Performance loss: no COW from overlayfs
  - “chown -R” for every file in each container
Problems with volumes

- mounted in several containers
- No UID translation
- Dynamic UID maps

bind mount (rw / ro)
User namespace and filesystem problem

- Possible solution: add options to mount() to apply a UID mapping
- rkt would use it when mounting:
  - the overlay rootfs
  - volumes
Isolators
Isolators in rkt

- specified in an image manifest
- limiting capabilities or resources
Isolators in rkt

Currently implemented

- capabilities
- cpu
- memory

Possible additions

- block-bandwidth
- block-iops
- network-bandwidth
- disk-space
cgroups
What’s a control group (cgroup)

- group processes together
- organised in trees
- applying limits to them as a group
cgroup API

/sys/fs/cgroup/*

/proc/cgroups

/proc/$PID/cgroup
# systemd-cgls

```
-1 /usr/lib/systemd/systemd
  system.slice
    NetworkManager.service
      1147 /usr/sbin/NetworkManager --no-daemon
      10655 /sbin/dhcclient -d -q -sf /usr/libexec/...
...```

# cat /sys/fs/cgroup/systemd/system.slice/NetworkManager.service/cgroup.procs

```
1147
10655
#```
List of cgroup controllers

/sys/fs/cgroup/
  └── cpu
  └── devices
  └── freezer
  └── memory
  └── systemd

```
# ls -l /sys/fs/cgroup/
total 0
dr-xr-xr-x. 5 root root 0 Sep 29 14:36 blkio
lrwxrwxrwx. 1 root root 11 Sep 22 20:12 cpu -> cpu,cpuacct
lrwxrwxrwx. 1 root root 11 Sep 22 20:12 cpuacct -> cpu,cpuacct
dr-xr-xr-x. 5 root root 0 Sep 29 14:36 cpu,cpuacct
dr-xr-xr-x. 4 root root 0 Sep 29 14:36 cpuset
dr-xr-xr-x. 5 root root 0 Sep 29 14:36 devices
dr-xr-xr-x. 4 root root 0 Sep 29 14:36 freezer
dr-xr-xr-x. 3 root root 0 Sep 29 14:36 hugetlb
dr-xr-xr-x. 5 root root 0 Sep 29 14:36 memory
lrwxrwxrwx. 1 root root 16 Sep 22 20:12 net_cls -> net_cls,net_prio
dr-xr-xr-x. 3 root root 0 Sep 29 14:36 net_cls,net_prio
lrwxrwxrwx. 1 root root 16 Sep 22 20:12 net_prio -> net_cls,net_prio
dr-xr-xr-x. 3 root root 0 Sep 29 14:36 perf_event
dr-xr-xr-x. 5 root root 0 Sep 29 14:36 systemd
#```
Memory isolator

Application Image Manifest

```
"limit": "500M"
```

systemd service file

```
[Service]
ExecStart=
MemoryLimit=500M
```

systemd action

write to memory.limit_in_bytes
**CPU isolator**

- Application Image Manifest

- 

  ```
  "limit": "500m"
  ```

- systemd service file

  ```
  [Service]
  ExecStart=
  CPUShares=512
  ```

- write to `cpu.share`

- systemd action
Unified cgroup hierarchy

- Multiple hierarchies:
  - one cgroup mount point for each controller (memory, cpu...)
  - flexible but complex
  - cannot remount with a different set of controllers
  - difficult to give to containers in a safe way

- Unified hierarchy:
  - cgroup filesystem mounted only one time
  - soon to be stable in Linux (mount option "__DEVEL__sane_behavior" being removed)
  - initial implementation in systemd-v226 (September 2015)
  - no support in rkt yet
Questions?

Join us!

github.com/coreos/rkt
May 9 & 10, 2016 | Berlin, Germany

- Early bird tickets
- Sponsorships are still available
- Submit a talk before February 29th!

@coreosfest
coreos.com/fest