Journey to the Center of the Containers

What happens on my box when a container is started?

10 March 2018

Francesco Romani
Senior Software Engineer, Red Hat
fromani {gmail,redhat}

whoami

- sweng @ Red Hat: opinions and mistakes are my own!
- works daily(-ish): oVirt, libvirt, kvm, python, C
- interested in: golang, containers, kubernetes
- happy linux user (red hat linux, debian, ubuntu, fedora)
- geek
- not-so-great at drawing (just wait a few slides)

Goal of this Talk

Focus on container runtime: the "how" not the "what"

Understand the steps that brings a container to life

- What a container runtime does do?
- What is a container runtime, actually?

NON-Goals of this Talk:

We will **not** talk about:

- container images (tags, layers...)
- how to build images: assume pre-built image
- how to store and fetch images: assume image on local file system

There and Back Again

{hand-drawn map}

The Passage of the Marshes

- let's follow a trail: the kubernetes CRI api
- kube concepts are de facto more generic/reusable
- docker is a silo
- mapping to docker concepts is simple

STANDARD DISCLAIMER

I'm not a Kubernetes guru - mistakes may happen!
Please let me know of any error you may find!

So, what is a container?

What we commonly call a container is

- a set of the features of the linux kernel we use
- to (re)create a controlled process
- to run a well-known image

And what is a container runtime?

Any tool, or set of tools which can run *container image*

Examples:
- crio
- containerd
- rkt (same notes about docker)

Some well known names:

- **Docker is** a container runtime, but it is also much more
- **Kubernetes** is a management platform, requires a container runtime
- **runC** bare-bones container runtime (you most often want to augment it)

A recipe for containers

The basic building blocks:

- namespaces: process isolation
- cgroups: resource limits

Security enforcement tools:

- seccomp: limit syscall usage
- SELinux: mandatory access control
- linux capabilities: finer-grained privileges

What about pods?

A pod is a group of containers run in a shared context
(A pod with exactly one container is perfectly fine!)

The minimum schedulable entity for kubernetes

Nicely composable with containers (with what we commonly mean with "containers")

We want to distinguish to understand some key concepts later on.

Would you like to know more? (https://kubernetes.io/docs/concepts/workloads/pods/pod/)

The Building Blocks, Revisited: namespaces are the new hot topic

Namespaces: Intro

Inception: ~2002; major developments ~2006 and onwards.

A namespace...

```
wraps a global system resource in an abstraction that makes it appear to the processes
within the namespace that they have their own isolated instance of the global resource.
[...]
One use of namespaces is to implement containers.
```

Namespaces are *ephemeral* by default: they are tied to the lifetime of a process.
Once that process is gone, so is the namespace.

But we can improve this (more on later).

more documentation (http://man7.org/linux/man-pages/man7/namespaces.7.html)

Namespaces: API

A **Kernel** API, syscalls:

- unshare(2): move calling process in new namespace(s) - and more.
- setns(2): make the calling process join existing namespace(s)
- clone(2): create a new process, optionally joining a new namespace - and **much** more.

Namespaces: what we can unshare?

- cgroup: cgroup root directory (more on that later)
- ipc: System V IPC, POSIX message queues
- network: network devices, stacks, ports, etc.
- mount: mount points
- pid: process id hierarchy
- user: user and group IDs
- uts: hostname and NIS domain name

more documentation (http://man7.org/linux/man-pages/man7/namespaces.7.html)

Namespaces: /procfs goodies

```
Each process has a /proc/$PID/ns/ subdirectory containing one entry
for each namespace that supports being manipulated by setns(2)
```

Actually, the setns(2) syscall accepts a *file descriptor* as parameter.
Everything is a file!

```
Bind mounting (see mount(2)) one of the files in this directory to
somewhere else in the filesystem keeps the corresponding namespace of
the process specified by pid alive even if all processes currently in
the namespace terminate.
```

So we can make one namespace outlive a process

more documentation (http://man7.org/linux/man-pages/man7/namespaces.7.html)

Namespaces DIY: unshare

PID of the current shell:

```
///samurai7/~># echo $$
5184
```

We start a new process (bash) with different network and PID namespaces

```
///samurai7/~># unshare --net --fork --pid --mount-proc bash
///samurai7/~># echo $$
1
///samurai7/~># ifconfig
///samurai7/~>#
```

Let's doublecheck:

```
///samurai7/~># ls -lh /proc/{1,5184,5282}/ns/pid
lrwxrwxrwx. 1 root root 0 Feb 21 19:54 /proc/1/ns/pid -> pid:[4026531836]
lrwxrwxrwx. 1 root root 0 Feb 21 19:53 /proc/5184/ns/pid -> pid:[4026531836]
lrwxrwxrwx. 1 root root 0 Feb 21 19:54 /proc/5282/ns/pid -> pid:[4026532544]
```

Namespaces DIY: nsenter

Let's enter the namespaces context we created in the slide before:

```
///samurai7/~># nsenter -a -t 5282 /bin/sh
sh-4.4# ps -fauxw
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        32  0.0  0.0 122680  3864 pts/4    S    20:00   0:00 /bin/sh
root        33  0.0  0.0 149756  3700 pts/4    R+   20:00   0:00  \_ ps -fauxw
root         1  0.0  0.0 123884  5108 pts/2    S+   19:53   0:00 bash
sh-4.4# echo $$
32
```

Namespaces DIY: ip

Let's create a new network namespace cnt

```
///samurai7/~># ip netns add cnt
///samurai7/~># ip netns list
cnt
///samurai7/~># ls -lh /var/run/netns/cnt
-r--r--r--. 1 root root 0 Feb 23 19:15 /var/run/netns/cnt
```

And we connect it to the outside:

```
///samurai7/~># ip link add veth0 type veth peer name veth1
///samurai7/~># ip link list | grep veth
7: veth1@veth0: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default q
8: veth0@veth1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default q
///samurai7/~># ip link set veth1 netns cnt
///samurai7/~># ip netns exec cnt ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
7: veth1@if8: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 5a:68:bc:a2:4b:c0 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

Namespaces: wrap up

Namespaces allow us to have separate instances of system resources.

Operating System resources are still **shared**

With the linux namespaces, we have the bare bones of a simpl{e,istic} container engine!

But much more is needed.

The Building Blocks, Revisited: for isolation, meet many old friends

cgroups: intro

Inception: ~2007. Major update: ~2013

Linux **C** ontrol **Groups**: allow process to be organized in hierarical groups to do limiting and accounting of certain system resources.

Most notably, memory and CPU time (and more: block I/O, pids...)

Powerful and easy-as-possible resource control mechanism

But still quite complex to manage

cgroups: what can we control?

- blkio: limits on input/output access to and from devices
- cpu: uses the scheduler to provide cgroup tasks access to the CPU
- cpuacct: automatic reports on CPU resources used by tasks
- cpuset: assigns individual CPUs and memory nodes to tasks
- memory: sets limits on memory and reports on memory resources
- perf_event: performance analysis.

Specific Linux Distribution (e.g. RHEL) may offer more cgroups.

Add your own!

cgroups: API

Just use sysfs:

```
echo browser_pid > /sys/fs/cgroup/<restype>/<userclass>/tasks
```

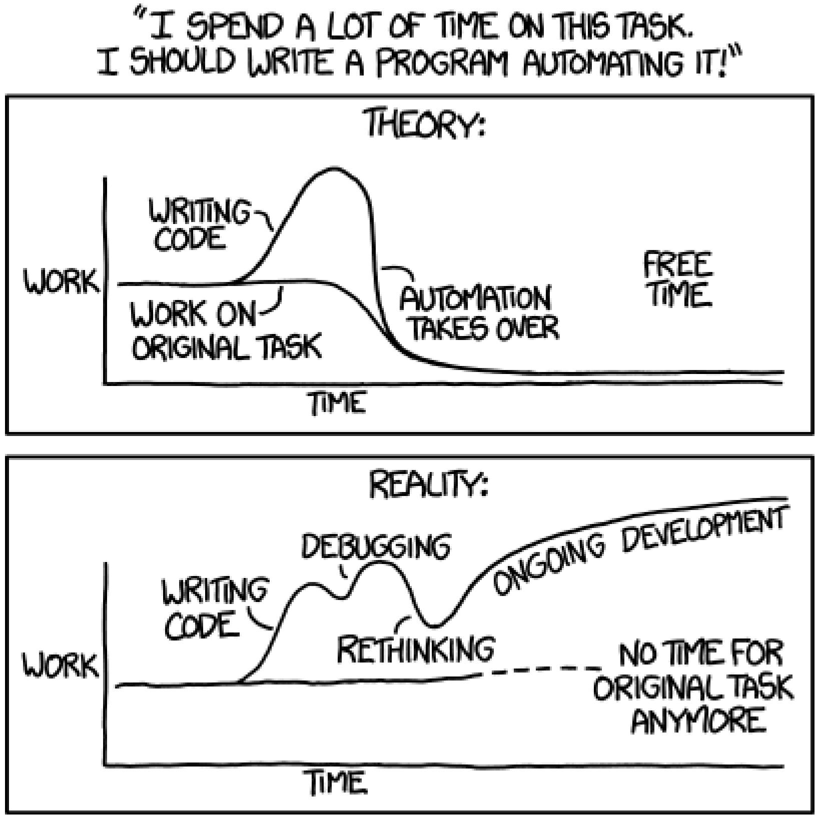
command line tools: cgcreate, cgexec, and cgclassify (from libcgroup).

Or just let your management engine do that for you:

- systemd
- libvirt
- docker
- any CRI-compatible runtime (more in the next slides)

cgroups: DIY

Mostly, you don't want to do it :)



Seriously, the management tool (whatever it is) almost always Just Works (tm) and it is simpler to tune.

cgroups: wrap-up

CGroups provide resource `limit` and `accounting`

Organized in hierarchies

A **LOT** of subtleties with respect to accounting and sensible limits

Here's why you should not DIY - don't reinvent a square wheel

Deserves a (long) talk on its own

seccomp

Inception: ~2005; Major update ~2012

Operational modes:

- 0 disabled
- 1 for strict: only *four* system calls: read, write, exit, sigreturn
- 2 for filter: allow developers to write filters to determine if a given syscall can run

seccomp: API & DIY

Kernel API (syscall), so just prctl(2) and seccomp(2)

And obviously `procfs` interface.

You can add your own syscall filters using **BPF** language (!!!)

Again, better don't reinvent the wheel, just use profiles from your management engine

If you really want to DIY, maybe start here (https://lwn.net/Articles/656307/)

SELinux

Inception: ~1998

Adds Mandatory Access Control (MAC) and Role Based Access Control (RBAC) to the linux kernel

Linux, being UNIX-Like, previously supported only Discretionary Access Control

SELinux: DAC vs MAC vs RBAC

WARNING: brutal semplification ahead

DAC: access control is based on the discretion of the owner: root can do anything.

MAC: the system (and not the users) specifies which can access what: no, even root *cannot* do that.

RBAC: in a nutshell, generalization of MAC: create and manage *Roles* to specify which entity can access which data.

beware: Again: the world is much more complex than that... (https://en.wikipedia.org/wiki/Role-based_access_control)

SELINUX: Daily usage

Mostly used on CentOS, Fedora, RHEL, RHEL-derived distributions

SELinux used to be perceived as overly complex, and overly annoying too.

"Just disable SELinux" was a recurrent advice up until not so long ago

It got **EXTREMELY** better: most of time, you don't even notice it is running. Just Works (tm)

Except when it prevents exploits :)

If you need to troubleshoot something, `audit2why` is usally a great start

Again, most often just use the profiles your distribution/management engine provides

- Lots of documentation available (http://selinuxproject.org/page/Main_Page)

We have the tools, let's build something

Kubernetes

```
Kubernetes is an open-source system for automating deployment,
scaling, and management of containerized applications.
```

source (https://kubernetes.io/)

Used to require Docker to actually run containers

Became runtime-agnostic since version 1.5 (http://blog.kubernetes.io/2016/12/container-runtime-interface-cri-in-kubernetes.html)

Kubelet

Node-agent of kubernetes: runs on every worker node of your cluster

Makes sure that containers described by POD specifications are running and healthy

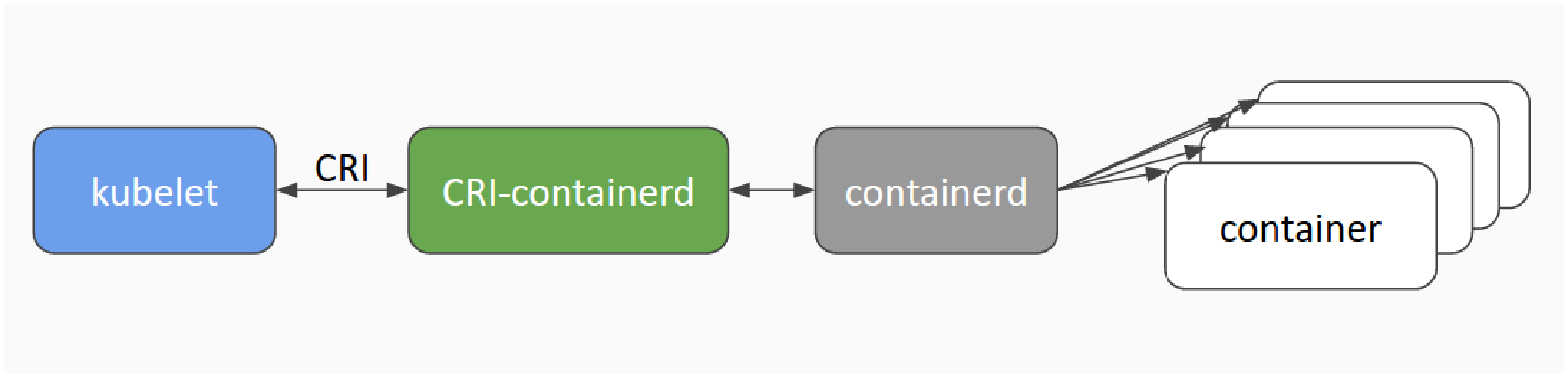
See few slides before for the definition of POD and Container

more documentation (https://kubernetes.io/docs/reference/generated/kubelet/)

The CRI API

Introduced in Kubernetes 1.5

TODO: redo image



Took some time to really have valid alternatives

Plugin interface which enables kubelet to use different container runtimes

Previously, to use a different runtime (e.g. not docker), patches to kubelet were needed.

Noteworthy CRI-compliant runtimes

cri-o: OCI Kubernetes Container Runtime daemon (https://github.com/kubernetes-incubator/cri-o)

rtklet: rkt-based implementation of Kubernetes CRI (https://github.com/kubernetes-incubator/rktlet)

cri-containerd: containerd-based implementation of Kubernetes CRI. (https://github.com/containerd/cri-containerd)

frakti: lets Kubernetes run pods and containers directly inside hypervisors via runV (https://github.com/kubernetes/frakti)

Let's get this process started (1/2)

crictl: CRI command line interface (https://github.com/kubernetes-incubator/cri-tools)

- command line is concise, easy to paste in the slides :)
- simple enough to not get in the way
- 1:1 mapping to protocol

crictl configuration (/etc/crictl.yaml): specify the UNIX domain socket to use to connect to the CRI runtime

obviously, each runtime uses a different path :(

Let's get this process started (2/2)

CRI compliant runtime: anything is fine.

But we pick cri-containerd, which in turn uses containerd.

```
[...]Containerd is used by Docker, Kubernetes CRI, and a few other projects[...]
```

source (https://blog.docker.com/2017/08/what-is-containerd-runtime/)

We can see how Kubernetes and Docker use the same building blocks!

WARNING: be careful to pick the right version of crictl and CRI runtime

The CRI interface API

from the protocol spec: (https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/apis/cri/runtime/v1alpha2/api.proto)

```
service RuntimeService {
  // Sandbox operations.
  rpc RunPodSandbox(RunPodSandboxRequest) returns (RunPodSandboxResponse) {}
  rpc PodSandboxStatus(PodSandboxStatusRequest) returns (PodSandboxStatusResponse) {}
  rpc ListPodSandbox(ListPodSandboxRequest) returns (ListPodSandboxResponse) {}
  ...
  // Container operations.
  rpc CreateContainer(CreateContainerRequest) returns (CreateContainerResponse) {}
  rpc StartContainer(StartContainerRequest) returns (StartContainerResponse) {}
  rpc ListContainers(ListContainersRequest) returns (ListContainersResponse) {}
  rpc ContainerStatus(ContainerStatusRequest) returns (ContainerStatusResponse) {}
  ...
}
```

Create workflow (aka: what to expect next)

- define a "sandbox" (in CRI jargon) using a JSON spec
- create and "run" it -> **RuntimeService.RunPodSandBox**
- define containers again using a JSON spec -> **RuntimeService.CreateContainer**
- create container(s) in the sandbox -> **RuntimeService.StartContainer**

Sandbox bootstrap

create and run

```
# cat sandbox-config.json
{
  "metadata": {
      "name": "test-sandbox",
      "namespace": "default",
      "attempt": 1,
      "uid": "hdishd83djaidwnduwk28bcsb"
  },
  "linux": {
  }
}
# crictl create sandbox-config.json
# crictl runs sandbox-config.json
```

inspect:

```
# crictl sandboxes
SANDBOX ID          CREATED             STATE               NAME                NAMESPACE           ATTE
959fcb9d9207a       3 minutes ago       SANDBOX_READY       test-sandbox        default             1
```

Sandbox: what's in there?

```
USER       PID %CPU %MEM    VSZ   RSS COMMAND
root      1436  0.0  0.3 854920 27176 /usr/local/bin/containerd
root      1823  0.0  0.0   8928  4184  \_ containerd-shim -namespace k8s.io ...
root      1839  0.0  0.0   1028     4      \_ /pause
root      1446  0.0  0.2 398328 22364 /usr/local/bin/cri-containerd ...
```

Uhm, pause?

Step back for a sec.

What is a sandbox, by the way?

A "sandbox" - in CRI parlance - is the environment on which the real container are supposed to run: namespaces, cgroups, networking, everything setup and ready.

But because of how the linux APIs work, we need a parent container which will serve as foundation for the Linux namespace sharing.

That's the pause container!

So, is this CRI Sandbox a pod?

Nope. It isn't that easy.

We can revisit the former definition we gave:
"A pod is a group of containers run in a shared context"

The shared context involves:

- namespaces
- cgroup setting
- network setup (special noteworthy case of namespaces)
- host volumes

in practice pods don't actually exist in your system (everything is a container :))

Meet the pause container

- simple as possible container - both in implementation and in resource usage
- 68 lines of C code (counting license boilerplate and #includes)
- 36 lines of *actual* C code
- free bonus: reap PIDs in the pod (no zombies!)

See it yourself (https://github.com/kubernetes/kubernetes/blob/master/build/pause/pause.c)

The pause container visualized

{ diagram of nested namespaces, cgroups, containers }

Tumbling down the rabbit hole

DISCLAIMER whirlwind tour, otherwise we would need another talk for the code walkthrough

- bookkeeping: create sandbox ID, register in some internal data store
- pull image if needed
- **set up network if needed** - more on that in the next slides
- produce container spec - augment with sandbox spec, set sandbox image name (aka the pause container), set seccomp policies)
- run the container - more on that in the next slides!
- a bit more bookkeeping (/etc/resolv.conf, mounts if needed)

Yep: a special case of a container.

Network set in this stage. Worth a closer look.

CNI - the Container Network Interface

```
specification and libraries for writing plugins to configure network interfaces in Linux containers,
along with a number of supported plugins.
```

source (https://github.com/containernetworking/cni)

used by kubernetes (http://blog.kubernetes.io/2016/01/why-Kubernetes-doesnt-use-libnetwork.html)

Pretty much popular. Docker uses a different approach (libnetwork).
But Docker may also use CNI too!

CNI in a nutshell:

- Each plugin is actually one executable (!!)
- Network defined using JSON - yet another syntax
- Network configuration is not persisted - feed to plugin via stdin
- Plugin arguments via environmental variables.
- The CNI plugin is responsible to add the container to the network (wiring)
- The CNI plugin is resposnible for **IP** *A*ddress *M*anagement (IPAM)

A plugin may do either network or IPAM, or both

CNI plugins showcase

Available plugins (on fedora 27, using stock rpm packages for CNI plugins):

```
# ls -lh /usr/libexec/cni/
total 34M
-rwxr-xr-x. 1 root root 2.9M Jan 23 19:03 bridge
-rwxr-xr-x. 1 root root 7.3M Jan 23 19:03 dhcp
-rwxr-xr-x. 1 root root 2.1M Jan 23 19:03 flannel
-rwxr-xr-x. 1 root root 2.2M Jan 23 19:03 host-local
-rwxr-xr-x. 1 root root 2.6M Jan 23 19:03 ipvlan
-rwxr-xr-x. 1 root root 2.2M Jan 23 19:03 loopback
-rwxr-xr-x. 1 root root 2.6M Jan 23 19:03 macvlan
-rwxr-xr-x. 1 root root 2.5M Jan 23 19:03 portmap
-rwxr-xr-x. 1 root root 2.9M Jan 23 19:03 ptp
-rwxr-xr-x. 1 root root 1.9M Jan 23 19:03 sample
-rwxr-xr-x. 1 root root 2.1M Jan 23 19:03 tuning
-rwxr-xr-x. 1 root root 2.6M Jan 23 19:03 vlan
```

CNI: DIY (1/3)

Let's try CNI using `cnitool`

[fetch it from there](https://github.com/containernetworking/cni/tree/master/cnitool) (https://github.com/containernetworking/cni/tree/master/cnitool)

Let's create the playground

```
# ip netns add cnitest
# cat /etc/cni/net.d/10-myptp.conf
{
  "cniVersion":"0.3.1",
  "name":"myptp",
  "type":"ptp",
  "ipMasq":true,
  "ipam":{
    "type": "host-local",
    "subnet":"172.16.29.0/24",
    "routes":[
      { "dst": "0.0.0.0/0" }
    ]
  }
}
```

CNI: DIY (2/3)

```
# CNI_PATH=/usr/libexec/cni cnitool add myptp /var/run/netns/cnitest
{
  "cniVersion": "0.3.1",
  "interfaces": [{
      "name": "veth6441a11f",
      "mac": "ae:ee:69:e4:4d:83"
    }, {
      "name": "eth0",
      "sandbox": "/var/run/netns/cnitest"
  }],
  "ips": [{
    "version": "4",
    "interface": 1,
    "address": "172.16.29.2/24",
    "gateway": "172.16.29.1"
  }],
  "routes": [{
    "dst": "0.0.0.0/0"
  }],
  "dns": {}
```

Did it worked? Let's try.

CNI: DIY (3/3)

```
# ip -n cnitest addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
  link/ether c6:6f:8b:12:0a:d5 brd ff:ff:ff:ff:ff:ff link-netnsid 0
  inet 172.16.29.2/24 scope global eth0
   valid_lft forever preferred_lft forever
  inet6 fe80::c46f:8bff:fe12:ad5/64 scope link
   valid_lft forever preferred_lft forever
# ip netns exec cnitest ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=12.3 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 12.363/12.363/12.363/0.000 ms
```

Yep, it worked! Back to sandboxes.

At last, containers

The container spec

A bare bones container spec

```
# cat container-config.json
{
  "metadata": {
      "name": "busybox"
  },
  "image":{
      "image": "busybox"
  },
  "command": [
      "top"
  ],
  "linux": {
  }
}
```

Will be consumed by `runc` - see next slides

container ready to takeoff

Let's run our container in the sandbox we created - yep, we must feed the sandbox-config again

```
# crictl create 959fcb9d9207a container-config.json sandbox-config.json
54b8f3e3ea8ab70b966c0ac6b60b415eb14305f92f80b2fad015b22a0ecff836
```

Created, but not started!

```
# crictl ps
CONTAINER ID        IMAGE               CREATED             STATE               NAME                ATTE
# crictl ps -a
CONTAINER ID        IMAGE               CREATED             STATE               NAME                ATTE
54b8f3e3ea8ab       busybox             15 seconds ago      CONTAINER_CREATED   busybox             0
```

Starting:

```
crictl start 54b8f3e3ea8ab
```

container run!

```
USER       PID %CPU %MEM    VSZ   RSS COMMAND
root      1436  0.0  0.3 854920 27176 /usr/local/bin/containerd
root      1823  0.0  0.0   8928  4184  \_ containerd-shim -namespace k8s.io ...
root      1839  0.0  0.0   1028     4      \_ /pause
root      2813  0.0  0.0   7520  4308  \_ containerd-shim -namespace k8s.io ...
root      2829  0.0  0.0   1240     4       \_ top
root      1446  0.0  0.2 398328 22364 /usr/local/bin/cri-containerd ...
```

container sharing is container caring

Let's check how the namespace are set

```
cgroup -> cgroup:[4026531835]
ipc -> ipc:[4026532380]
-mnt -> mnt:[4026532378]
+mnt -> mnt:[4026532382]
net -> net:[4026532299]
-pid -> pid:[4026532381]
-pid_for_children -> pid:[4026532381]
+pid -> pid:[4026532383]
+pid_for_children -> pid:[4026532383]
user -> user:[4026531837]
uts -> uts:[4026532379]
```

Containers share all namespaces but `mnt` and `pid`*

Looks good. How did we do that?

meet runC

runC is the de facto standard to run containers
many management engines (e.g. containerd) use the runC cli interface as ABI
you can use any tool that implements the same CLI - but why bother? :\

originated by Docker in the ~2014, now standalone project

Includes a command line tool `runc` and a package `libcontainer` to run containers (surprise!)

how runC works (1/3)

In a nutshell, it uses the building blocks we introduced before

- containers defined by a spec (JSON document)
- spec is pretty detailed (selinux, seccomp, capabilites, apparmor...)
- assumes to be run in the root of a `bundle`
- management engines like containerd takes care of setting everything (config files, directories) and then spawns runc
- by default, leverages systemd for cgroups managemnt

how runC works (2/3)

serverless: it spawns the container, actually the executable that was requested to run, and exits

supervisor-less: you can use systemd directly (but most likely you shouldn't - lot of stuff to take care of) use a management tool instead!

stateless: everything stored under /run

how runC works (3/3)

Starting a container is actually surprisingly hard

because linux namespaces and go don't mix (https://www.weave.works/blog/linux-namespaces-and-go-don-t-mix)

runC solves lots of complexity and pitfalls

more than enough for another talk on this topic only.

some noteworthy pitfalls

Let's just highlight the deadliest traps

the golang runtime multiplexes goroutines on threads

threads can be spawned any time

so it is unsafe to use setns(2) in go code: the goroutine environment can be corrupted without warning

- it is a known issue (https://github.com/golang/go/issues/20676)
- fork()ing in go code is not safe either (https://groups.google.com/forum/#%21topic/golang-nuts/ss1gEOcehjk/discussion)

black magic

runC does its job with a complex dance featuring:

a two-stage fork

calling itself while starting a container

- using helper code in C to be called before the golang runtime starts.

  (https://github.com/opencontainers/runc/blob/master/libcontainer/nsenter/README.md)

But in the end

It just works :)

```
docker run redis
```

wrap-up/takeaways

- pod doesn't really exist on your system, only as concept
- a container is the execution of a given *image*
- under the hood containers are agglomerates of cooperating features of the Linux Kernel
- many of those features are quite old, and only recently they started to be integrated
- lot of moving parts, **and** the environment is still evolving and changing
- interoperability is improving (also thanks to kubernetes and CRI?)
- a lot of integrated tools: you can plug in at pretty much ever layer

Q? A!

Thank you

Francesco Romani
Senior Software Engineer, Red Hat
fromani {gmail,redhat}
http://github.com/{mojaves,fromanirh} (http://github.com/%7Bmojaves,fromanirh%7D)