Building blocks of Linux Containers

Motiejus Jakštys
motiejus@uber.com
@mo_kelione

UBER

2016-11-18

Table of Contents

Conclusion!

Conclusion!

Devil Hides in The Details.

Conclusion!

Devil Hides in The Details?

Conclusion!

Devil Hides in The Details?

- Many use Docker.

Conclusion!

Devil Hides in The Details?

- Many use Docker.
- We lack time to understand.

Conclusion!

Devil Hides in The Details?

- Many use Docker.
- We lack time to understand.
    - You need to understand infra to successfully troubleshoot infra.

Conclusion!

Devil Hides in The Details?

- Many use Docker.
- We lack time to understand.
    - You need to understand infra to successfully troubleshoot infra.
    - There are trade-offs in the configuration.

Conclusion!

Devil Hides in The Details?

- Many use Docker.
- We lack time to understand.
  - You need to understand infra to successfully troubleshoot infra.
  - There are trade-offs in the configuration.
- Make container engine in 30 minutes.

Conclusion!

Devil Hides in The Details?

- Many use Docker.
- We lack time to understand.
    - You need to understand infra to successfully troubleshoot infra.
    - There are trade-offs in the configuration.
- Make container engine in 30 minutes.
- Details!

Conclusion!

Devil Hides in The Details?

- Many use Docker.
- We lack time to understand.
  - You need to understand infra to successfully troubleshoot infra.
  - There are trade-offs in the configuration.
- Make container engine in 30 minutes.
- Details! $\rightarrow$ You will still pick existing tools.

Why me

My resume: oncall experience.

- 2009 – 2012 Telecom (Dev + Ops).
- 2012 – 2014 Online Gaming (Dev + Ops).
- 2014 – 2016 Amazon (Dev + Ops).
- 2016 – *now* Uber (Dev + Ops):
    - From 2016.02: Dev.
    - From 2016.11: SRE.

Why me

My resume: oncall experience.

- 2009 – 2012 Telecom (Dev + Ops).
- 2012 – 2014 Online Gaming (Dev + Ops).
- 2014 – 2016 Amazon (Dev + Ops).
- 2016 – *now* Uber (Dev + Ops):
  - From 2016.02: Dev.
  - From 2016.11: SRE.

I had to understand how exactly infrastructure works.

A container in Linux is ...

A container in Linux is ...

Fork/exec with bells & whistles:

A container in Linux is ...

Fork/exec with bells & whistles:

- Fancy tarball for distribution.

A container in Linux is ...

Fork/exec with bells & whistles:

- Fancy tarball for distribution.
- COW filesystem to make it start fast.

A container in Linux is ...

Fork/exec with bells & whistles:

- Fancy tarball for distribution.
- COW filesystem to make it start fast.
- Cgroups for fairness.

A container in Linux is ...

Fork/exec with bells & whistles:

- Fancy tarball for distribution.
- COW filesystem to make it start fast.
- Cgroups for fairness.
- Namespaces for isolation.

Table of Contents

We will cover

We will cover

- User namespaces.

We will cover

- User namespaces.
- Pid namespaces.

We will cover

- User namespaces.
- Pid namespaces.
- Mount namespaces.

We will cover

- User namespaces.
- Pid namespaces.
- Mount namespaces.
- Network namespaces.

We will cover

- User namespaces.
- Pid namespaces.
- Mount namespaces.
- Network namespaces.
- There are more, but not today.

User namespace

Become container-local root.

```
unshare --map-root-user
```

Mount namespace

Hide container mounts.

```
unshare --mount
```

Pid namespace

Hide other pids.

```
unshare --pid --mount-proc --fork
```

Network namespace

Demonstrate this:

Network namespace

Demonstrate this:

- Create namespace.

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:
    - `veth1b` will go to the namespace.

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:
    - `veth1b` will go to the namespace.
    - `veth1a` will stay in default.

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:
  - `veth1b` will go to the namespace.
  - `veth1a` will stay in default.
- Add ip addresses.

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:
  - `veth1b` will go to the namespace.
  - `veth1a` will stay in default.
- Add ip addresses.
- curl and ping.

Network namespace

Demonstrate this:

- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:
  - `veth1b` will go to the namespace.
  - `veth1a` will stay in default.
- Add ip addresses.
- curl and ping.
- lsof, bind on ports separately.

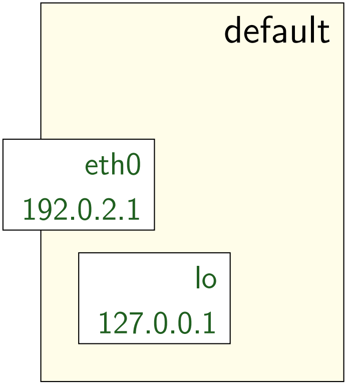
Network namespace

Demonstrate this:

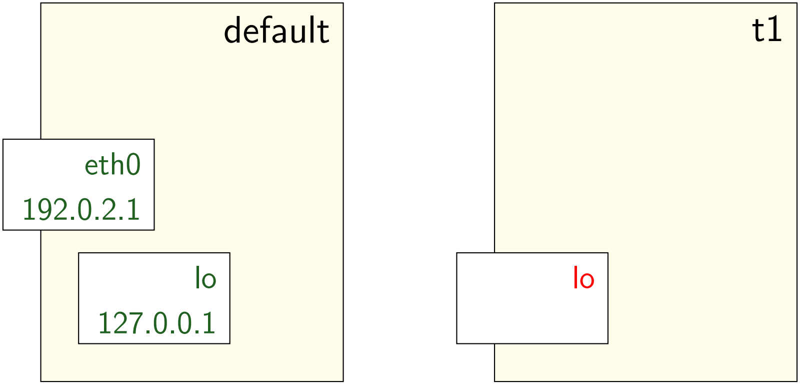
- Create namespace.
- Activate loopback (`lo`).
- Create pair of devices `veth1a` and `veth1b`:
  - `veth1b` will go to the namespace.
  - `veth1a` will stay in default.
- Add ip addresses.
- curl and ping.
- lsof, bind on ports separately.

Ever wanted to run `tcpdump` on an *application*?

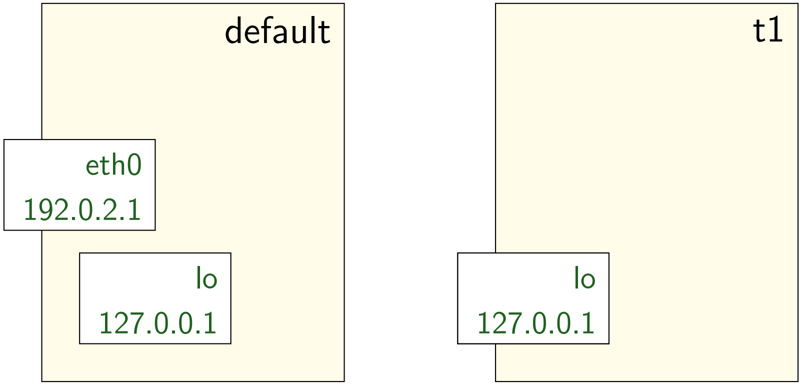
Network namespace



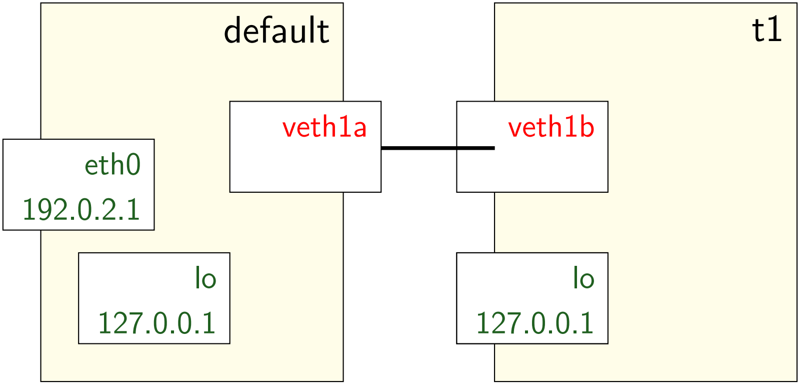
Network namespace



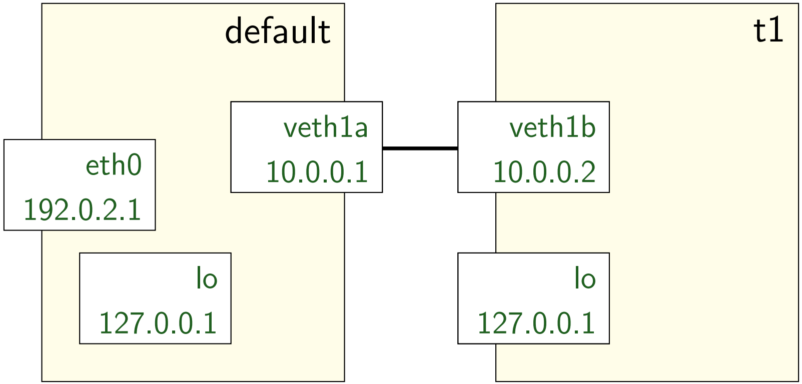
Network namespace



Network namespace



Network namespace



What did we just do

Created a container:

What did we just do

Created a container:

User namespace apt-get, iptables, mount, etc.

What did we just do

Created a container:

User namespace apt-get, iptables, mount, etc.

Isolated pids no `nobody`, isolate from each other.

What did we just do

Created a container:

User namespace apt-get, iptables, mount, etc.

Isolated pids no `nobody`, isolate from each other.

Isolated mounts e.g. for /tmp.

What did we just do

Created a container:

User namespace apt-get, iptables, mount, etc.

Isolated pids no `nobody`, isolate from each other.

Isolated mounts e.g. for /tmp.

Isolated network safely bind to `:80`.

What did we just do

Created a container:

User namespace apt-get, iptables, mount, etc.

Isolated pids no `nobody`, isolate from each other.

Isolated mounts e.g. for /tmp.

Isolated network safely bind to :80.

An improvement over "run and hope it doesn't affect anything else".

Table of Contents

File systems and COW

A container:

- Needs a file system.

File systems and COW

A container:

- Needs a file system.
- Starts quickly regardless of size.

File systems and COW

A container:

- Needs a file system.
- Starts quickly regardless of size.

Do not want to copy 1GB with every startup.

File systems and COW

A container:

- Needs a file system.
- Starts quickly regardless of size.

Do not want to copy 1GB with every startup.
Copy On Write!

File systems and COW

A container:

- Needs a file system.
- Starts quickly regardless of size.

Do not want to copy 1GB with every startup.

Copy On Write!

lvm? zfs? btrfs?

A quick demo

- Create `tank/images/debian@latest`
- Create `tank/containers/t1` from `@latest`
- `unshare --mount --pid --fork chroot .  bash`

Table of Contents

Leftover elephants in the room

Leftover elephants in the room

- Trivial to escape this ”container”.

Leftover elephants in the room

- Trivial to escape this "container".
- Sec: no leftover file descriptors.

Leftover elephants in the room

- Trivial to escape this ”container”.
- Sec: no leftover file descriptors.
- Resource fairness.

Leftover elephants in the room

- Trivial to escape this ”container”.
- Sec: no leftover file descriptors.
- Resource fairness.
- Sec/DoS: shared kernel resources.

Leftover elephants in the room

- Trivial to escape this ”container”.
- Sec: no leftover file descriptors.
- Resource fairness.
- Sec/DoS: shared kernel resources.
- Supervision, daemonization and cleanup.

Leftover elephants in the room

- Trivial to escape this ”container”.
- Sec: no leftover file descriptors.
- Resource fairness.
- Sec/DoS: shared kernel resources.
- Supervision, daemonization and cleanup.
- Logging.

Leftover elephants in the room

- Trivial to escape this "container".
- Sec: no leftover file descriptors.
- Resource fairness.
- Sec/DoS: shared kernel resources.
- Supervision, daemonization and cleanup.
- Logging.
- Collect zombie processes.

Leftover elephants in the room

- Trivial to escape this "container".
- Sec: no leftover file descriptors.
- Resource fairness.
- Sec/DoS: shared kernel resources.
- Supervision, daemonization and cleanup.
- Logging.
- Collect zombie processes.
- Image management.

Leftover elephants in the room

- Trivial to escape this "container".
- Sec: no leftover file descriptors.
- Resource fairness.
- Sec/DoS: shared kernel resources.
- Supervision, daemonization and cleanup.
- Logging.
- Collect zombie processes.
- Image management.

Should someone else do it?

We almost have a container engine

We almost have a container engine

- But look at my conclusions again.

We almost have a container engine

- But look at my conclusions again.
- Devil hides in the details.

We almost have a container engine

- But look at my conclusions again.
- Devil hides in the details.
- Tooling companies (Docker, CoreOS, etc) raised $> \$10^8$.

To recap

- Easy to understand kernel facilities.

To recap

- Easy to understand kernel facilities.
- Devil hides in the details.

To recap

- Easy to understand kernel facilities.
- Devil hides in the details.
- Either spend a lot of time and headache, or re-use existing tools.

Table of Contents

We're hiring!

Uber SRE locations: SF, NYC, Seattle, Vilnius.

- Check out join.uber.com
- Also, contact me at motiejus@uber.com