An Introduction to

Thomas Calmant INRIA Grenoble Rhône-Alpes SED/Tyrex February 2018

1
A bit of context

The big questions

For administrators and packagers:

- How to ensure an application will work everywhere ?
- How to avoid it messing with my system ?
- How to isolate the components of my application ?

The big questions

For administrators and packagers:

- How to ensure an application will work everywhere ?
- How to avoid it messing with my system ?
- How to isolate the components of my application ?

For developers:

- How to ensure everybody has the same build environment ?
- How to provide a sample to reproduce a bug ?

The Concept of Container

Concept of *Containerization* from freight transport

Transport

Isolation

The Concept of Container

Concept of *Containerization* from freight transport

Transport

Isolation

- can be (un-)loaded/stacked efficiently
- can be loaded on ships, trains, trucks, ...
- can be handled without being opened

The Concept of Container

Concept of *Containerization* from freight transport

Transport	Isolation
▶ can be (un-)loaded/stacked efficiently ▶ can be loaded on ships, trains, trucks, ... ▶ can be handled without being opened	▶ OpenContainer Runtime Specification

The Concept of Container

Concept of *Containerization* from freight transport

Transport

- can be (un-)loaded/stacked efficiently
- can be loaded on ships, trains, trucks, ...
- can be handled without being opened

Isolation

- OpenContainer Runtime Specification

- are tracked with an identification number
- have ISO-standard sizes (5 classes)

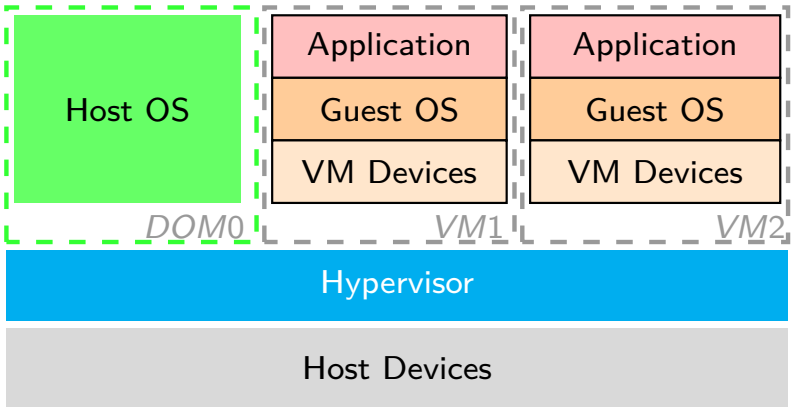
The Concept of Container

Concept of *Containerization* from freight transport

Transport	Isolation
▶ can be (un-)loaded/stacked efficiently ▶ can be loaded on ships, trains, trucks, ... ▶ can be handled without being opened	▶ OpenContainer Runtime Specification
▶ are tracked with an identification number ▶ have ISO-standard sizes (5 classes)	▶ OpenContainer Image Specification

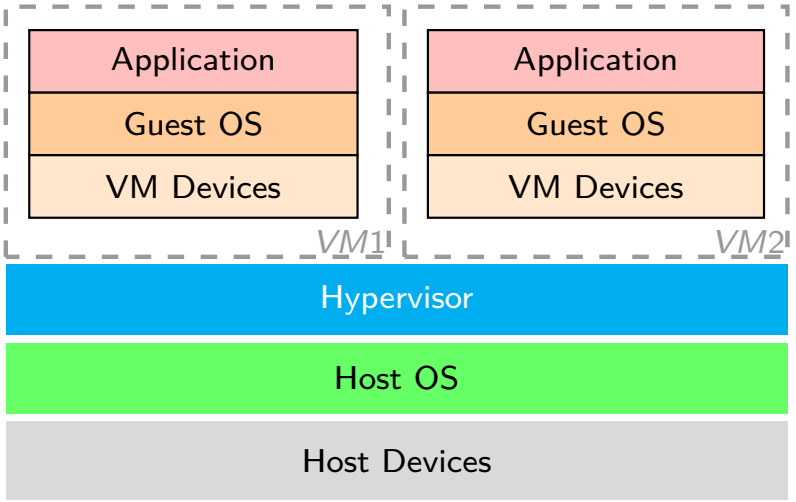
Types of Virtualizations

Type I



- The Hypervisor is a lightweight kernel
- Examples: Xen, Hyper-V, vSphere/ESXi, ...

Type II



- The Hypervisor runs above (partly inside) the host OS
- Examples: VirtualBox, VMWare Workstation, ...

A history of Isolation

1979 `chroot` (Version 7 Unix)

2000 `jail` (FreeBSD 4.0)

2005 Solaris Containers: *"chroot on steroids"* (Solaris 10)

A history of Isolation

1979 `chroot` (Version 7 Unix)

2000 `jail` (FreeBSD 4.0)

2005 Solaris Containers: *"chroot on steroids"* (Solaris 10)

2008/01 cgroups: Task Control Groups (Linux Kernel 2.6.24)

2008/08 LXC: Linux Containers (based on cgroups)

2013/02 User Namespaces (Linux Kernel 3.8)

A history of Isolation

1979 `chroot` (Version 7 Unix)

2000 `jail` (FreeBSD 4.0)

2005 Solaris Containers: *"chroot on steroids"* (Solaris 10)

2008/01 cgroups: Task Control Groups (Linux Kernel 2.6.24)

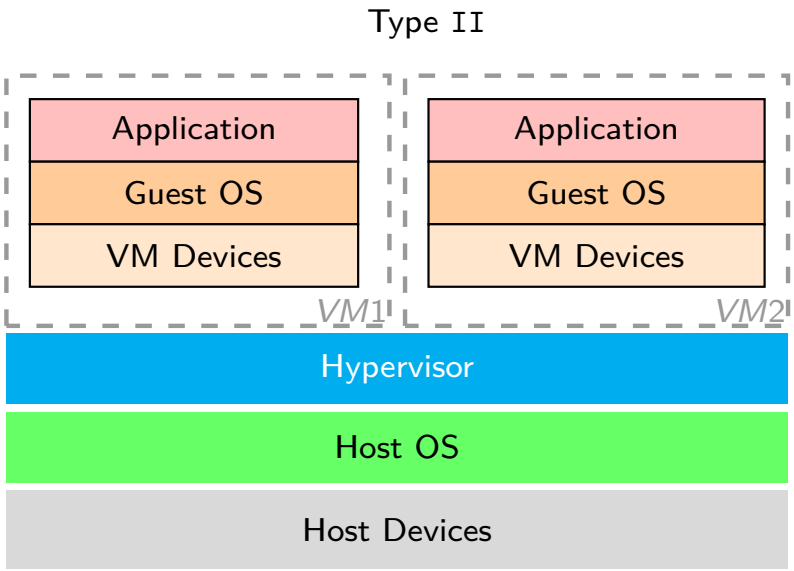
2008/08 LXC: Linux Containers (based on cgroups)

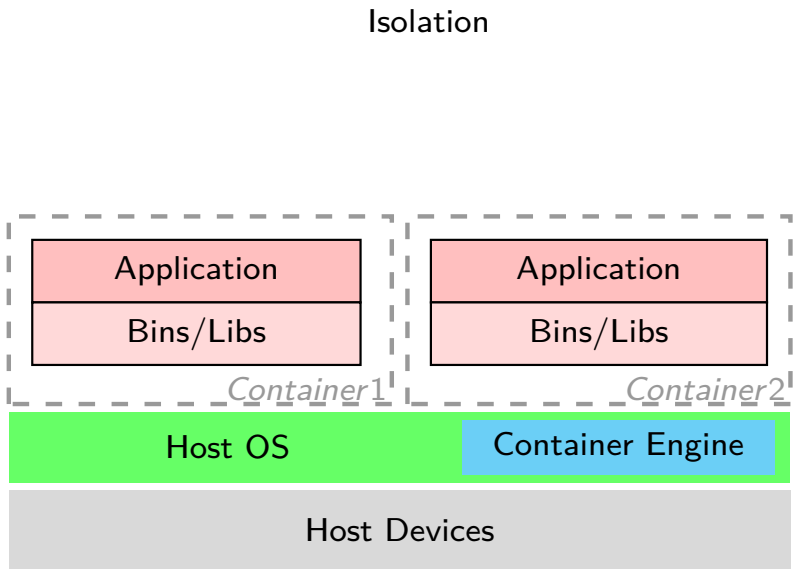
2013/02 User Namespaces (Linux Kernel 3.8)

2013/03 Docker (based on LXC),
announced in a Lightning Talk at PyCon 2013

2015/06 Open Container Initiative (by Docker)

Virtualization vs. Isolation





- Ability to run different kernel/OS
- Possibility to attach some of host devices

- Shared Kernel, handling isolation
- Kernel-handled virtual devices (network)

Different targets, different advantages

Virtualization

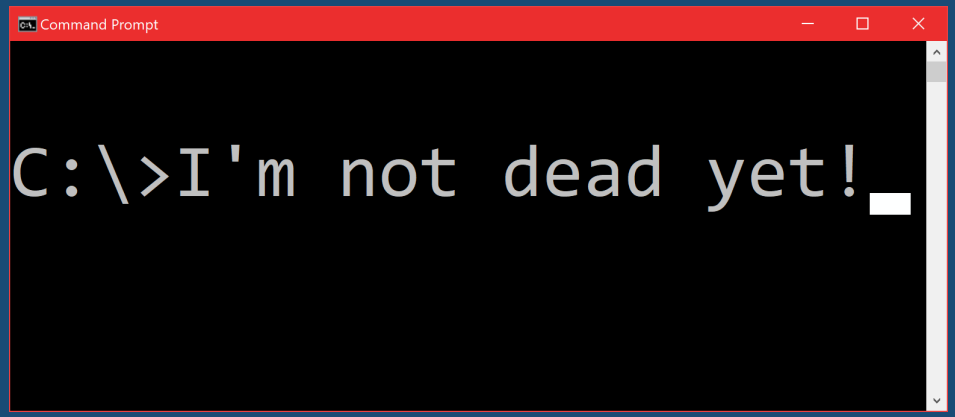
- Best isolation from the host
- Fine tuned resource quota
- Runs any guest OS
- Lots of management tools

Isolation

- Good enough isolation
- Benefit from kernel optimizations & quota
- Very low footprint
- Ease of use

Agenda

1. A bit of context
2. Playing with `docker`
3. Basic interaction with the host
4. Link containers together
5. Create a Docker image
6. Security
7. Scale up with Swarm
8. Miscellaneous



2
Playing with docker

Because nothing beats the command line

Warm up

- Check if `docker` works:
  - `docker info`
  - `docker run hello-world`
- If not...
  - Check if `docker` is installed
  - Check if your user is in the `docker` group:
    `sudo gpasswd -a $USER docker`

We're ready to go!

Docker on a Linux system

- On your machine:
  - Docker storage: `/var/lib/docker`
    - Only `root` can access this folder
    - Contains images, volumes and containers storage

Docker on a Linux system

- On your machine:
  - Docker storage: `/var/lib/docker`
    - Only `root` can access this folder
    - Contains images, volumes and containers storage
  - Docker UNIX Socket: `/var/run/docker.sock`
    - Only `root` and the `docker` group can access it
    - Default & recommended access to the local Docker Daemon

Docker on a Linux system

- On your machine:
  - Docker storage: `/var/lib/docker`
    - Only `root` can access this folder
    - Contains images, volumes and containers storage
  - Docker UNIX Socket: `/var/run/docker.sock`
    - Only `root` and the `docker` group can access it
    - Default & recommended access to the local Docker Daemon
- Docker can access remote locations:
  - Docker Daemon:
    - Docker official registry: Docker Hub
    - Private registries
  - Docker CLI
    - Manage a remote daemon via TCP/TLS
    - Manage a Docker Swarm

Time to work

Hands on: Running a container

- `docker run debian`

Hands on: Running a container

- `docker run debian`

- `docker run -it --name MyContainer debian`

Hands on: Running a container

- `docker run debian`
  - Starts a container based on the **debian** image
  - No `stdin`, so `bash` exits immediately (end of file)
- `docker run -it --name MyContainer debian`

Hands on: Running a container

- `docker run debian`
  - Starts a container based on the **debian** image
  - No `stdin`, so `bash` exits immediately (end of file)
- `docker run -it --name MyContainer debian`
  - `-i`: interactive mode (with `stdin`, `stdout`, `stderr`)
  - `-t`: with a valid TTY (screen size, coloration, ...)
  - `--name`: Set a name to ease management (unique per host)

Hands on: Running a container

- `docker run debian`
  - Starts a container based on the **debian** image
  - No `stdin`, so `bash` exits immediately (end of file)
- `docker run -it --name MyContainer debian`
  - `-i`: interactive mode (with `stdin`, `stdout`, `stderr`)
  - `-t`: with a valid TTY (screen size, coloration, ...)
  - `--name`: Set a name to ease management (unique per host)
- `docker ps`
  - Prints the list of active containers

Hands on: Running a container

- `docker run debian`
  - Starts a container based on the **debian** image
  - No `stdin`, so `bash` exits immediately (end of file)
- `docker run -it --name MyContainer debian`
  - `-i`: interactive mode (with `stdin`, `stdout`, `stderr`)
  - `-t`: with a valid TTY (screen size, coloration, ...)
  - `--name`: Set a name to ease management (unique per host)
- `docker ps -a`
  - Prints the list of active containers
  - `-a`: also shows stopped containers

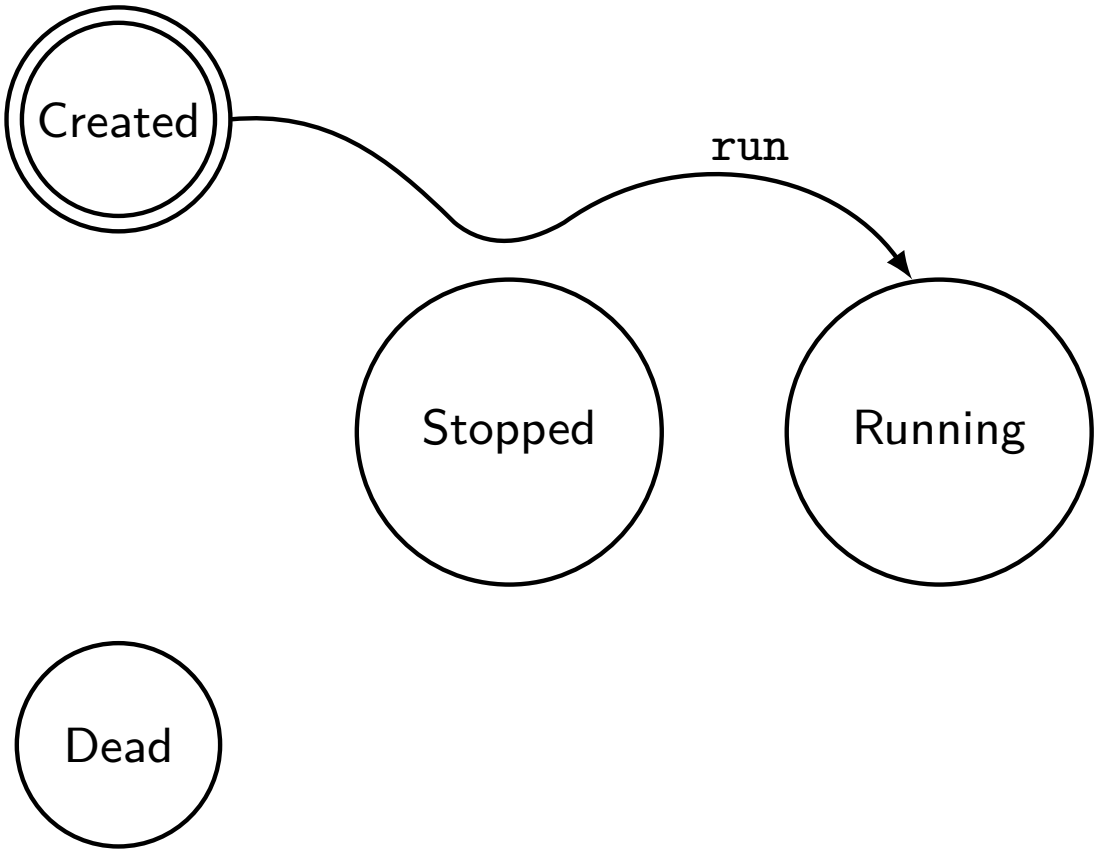
Hands on: Running a container

- `docker run debian`
  - Starts a container based on the **debian** image
  - No `stdin`, so `bash` exits immediately (end of file)
- `docker run -it --name MyContainer debian`
  - `-i`: interactive mode (with `stdin`, `stdout`, `stderr`)
  - `-t`: with a valid TTY (screen size, coloration, ...)
  - `--name`: Set a name to ease management (unique per host)
- `docker ps -a`
  - Prints the list of active containers
  - `-a`: also shows stopped containers
- `docker rm <CID/name>`
  - Removes a stopped container

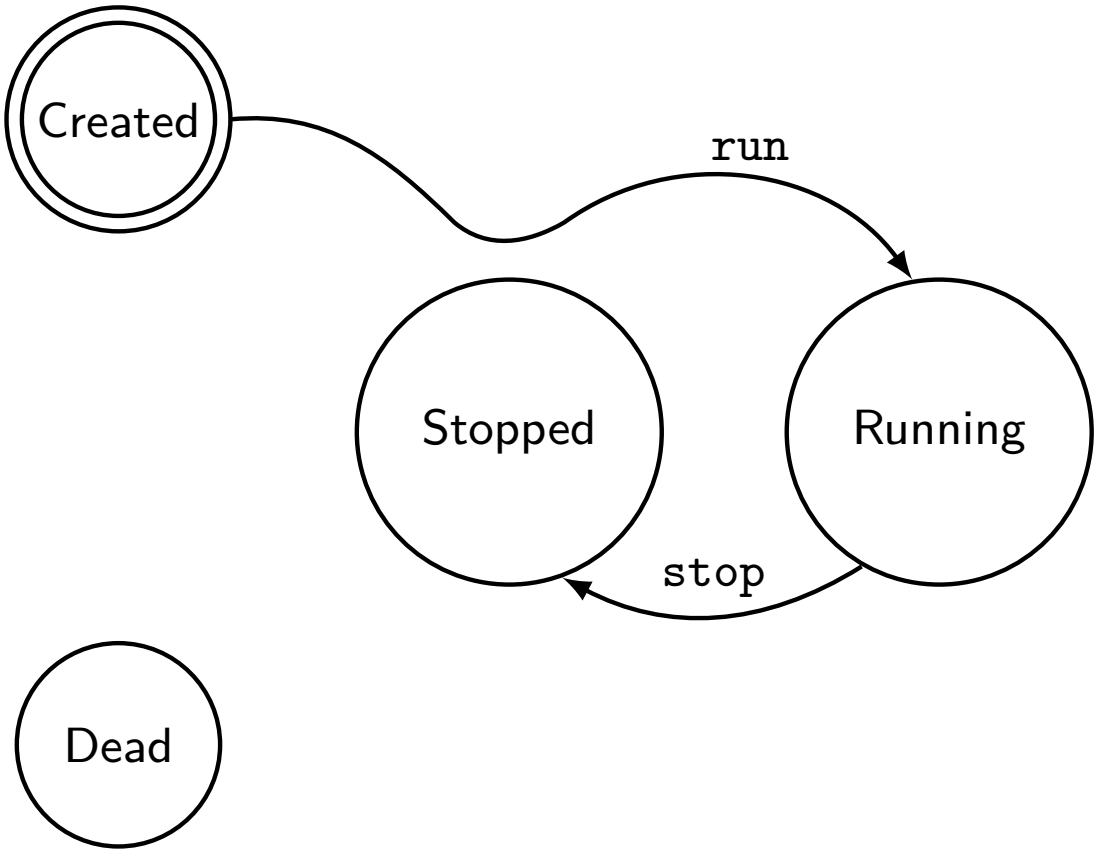
Hands on: Running a container

- `docker run debian`
  - Starts a container based on the **debian** image
  - No `stdin`, so `bash` exits immediately (end of file)
- `docker run -it --name MyContainer debian`
  - `-i`: interactive mode (with `stdin`, `stdout`, `stderr`)
  - `-t`: with a valid TTY (screen size, coloration, ...)
  - `--name`: Set a name to ease management (unique per host)
- `docker ps -a`
  - Prints the list of active containers
  - `-a`: also shows stopped containers
- `docker rm -f <CID/name>`
  - Removes a stopped container
  - `-f` stops the container if necessary

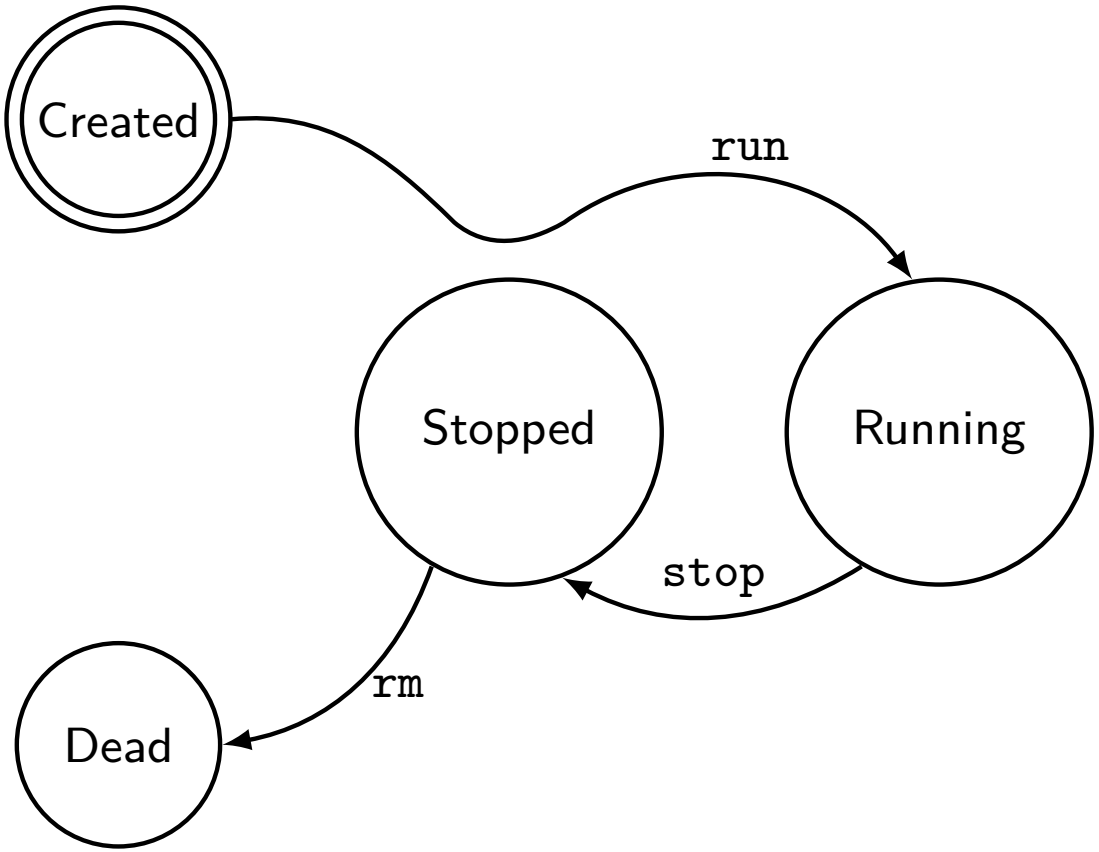
Container life cycle



Container life cycle

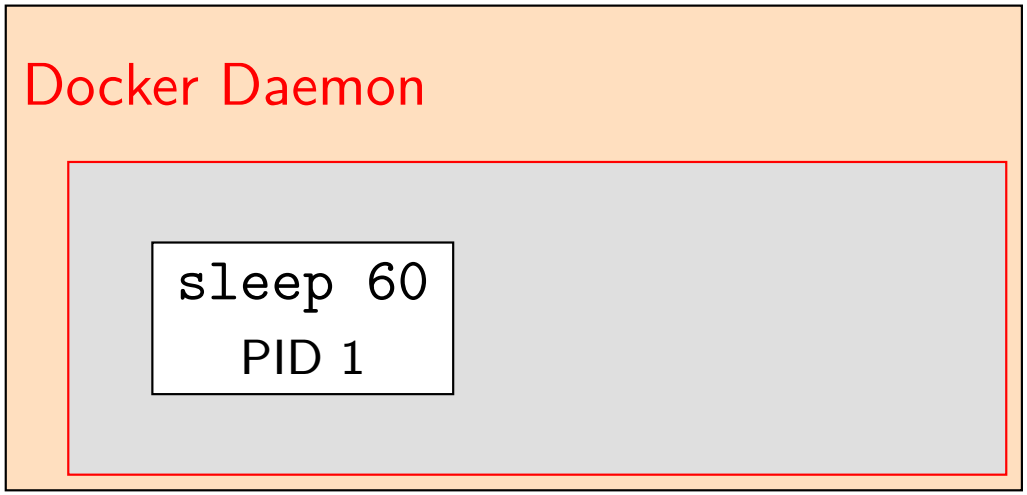


Container life cycle



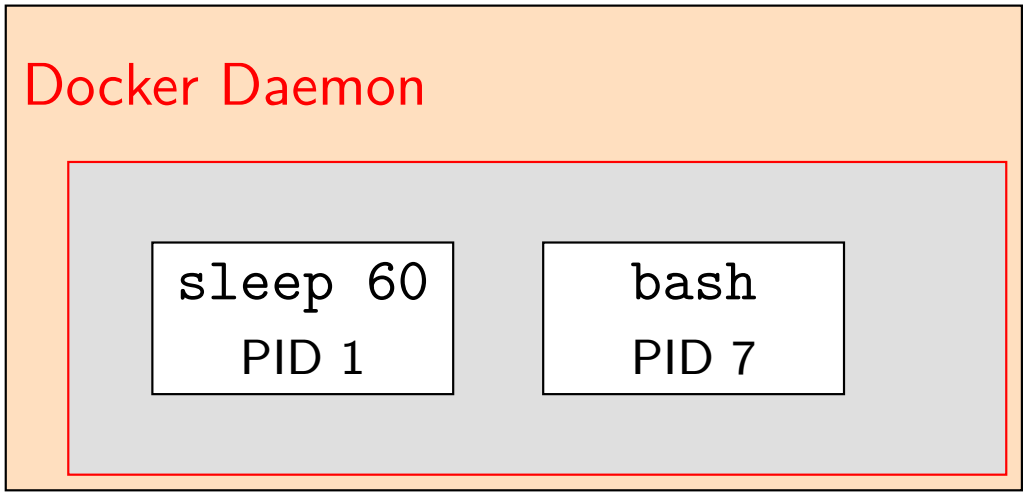
Running inside a container

- `docker run --name MyContainer -d debian sleep 60`
  - The container is started *detached* (`-d`)

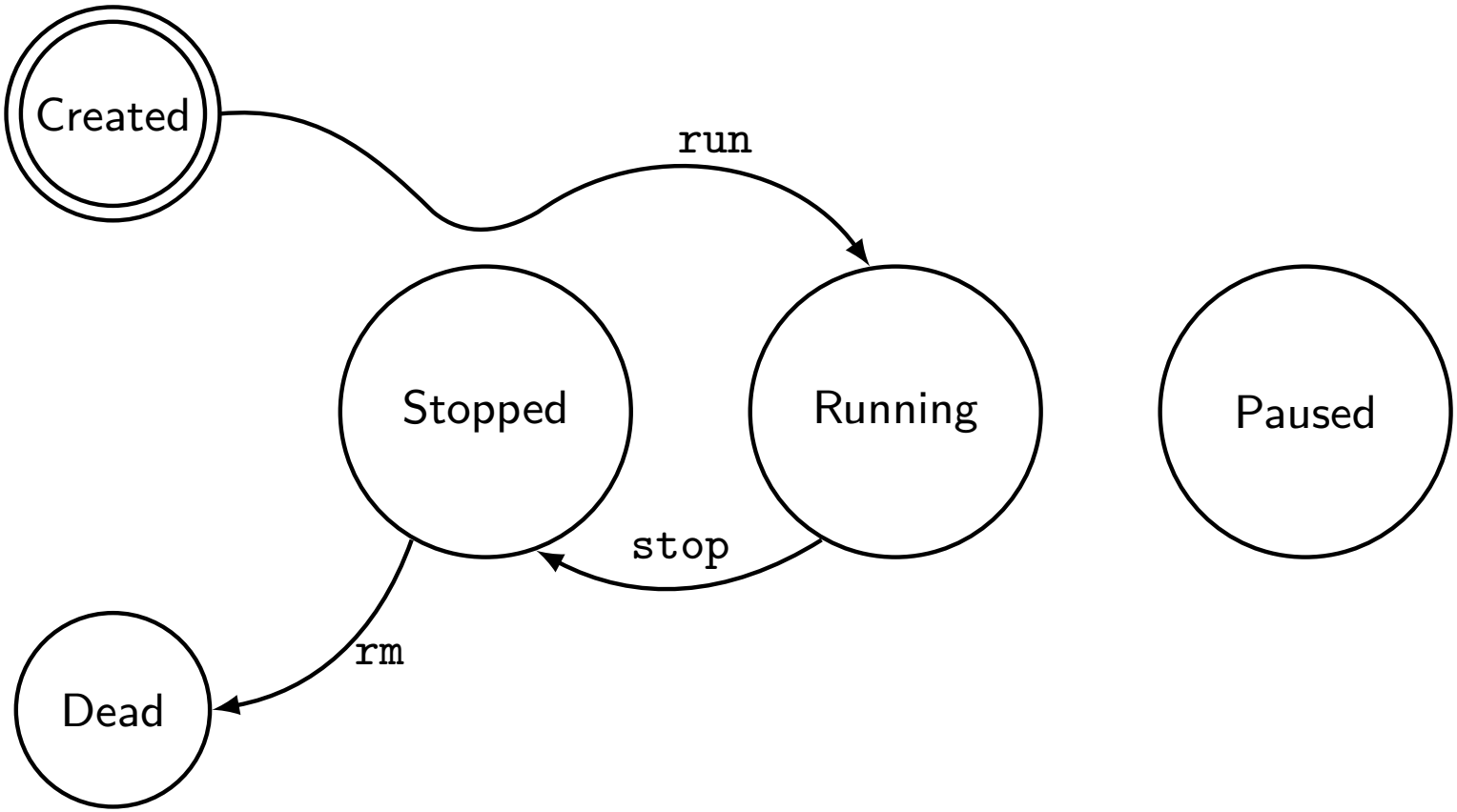


Running inside a container

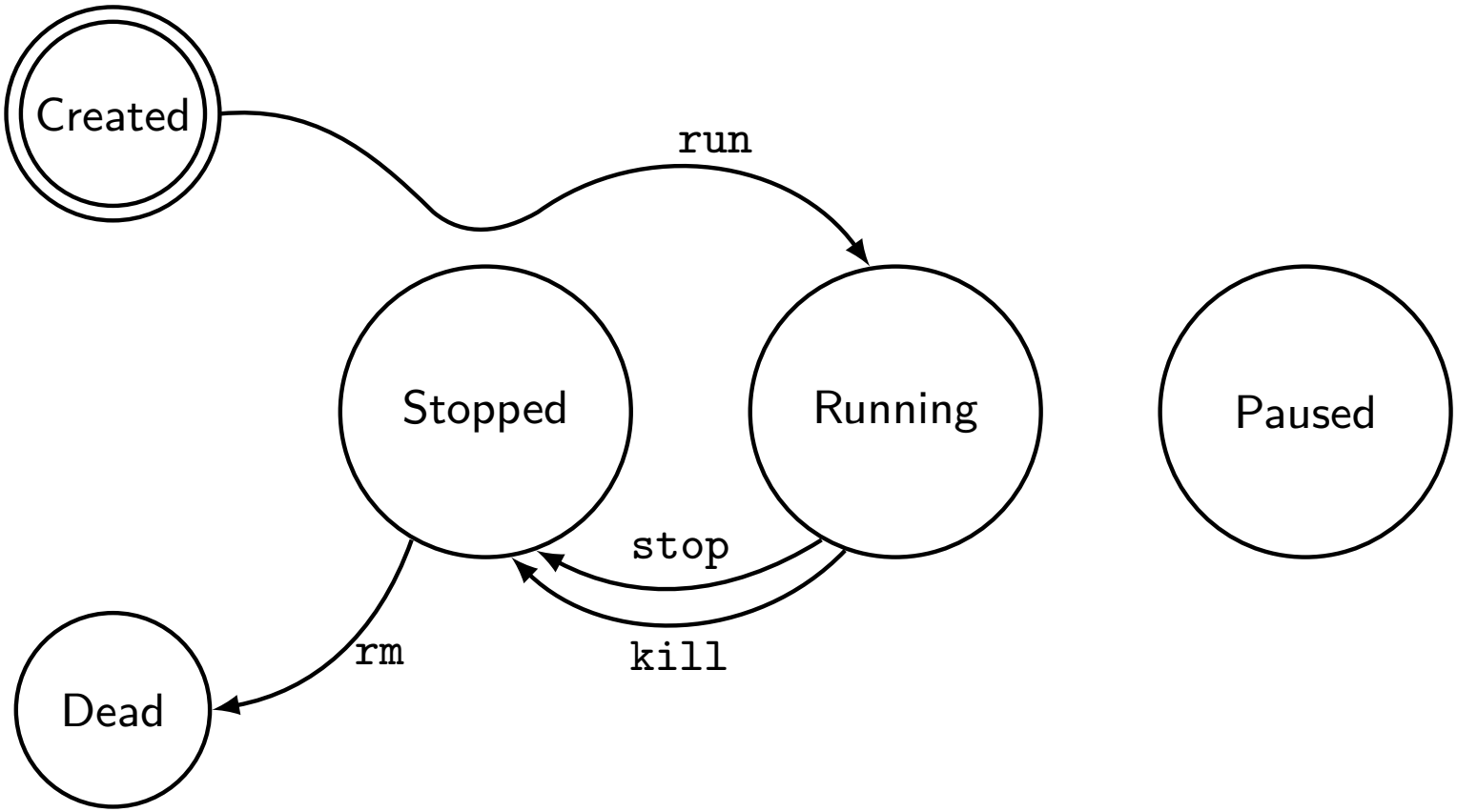
- `docker run --name MyContainer -d debian sleep 60`
  - The container is started *detached* (`-d`)
- `docker exec -it MyContainer bash`
  - Starts a new `bash` process in the container



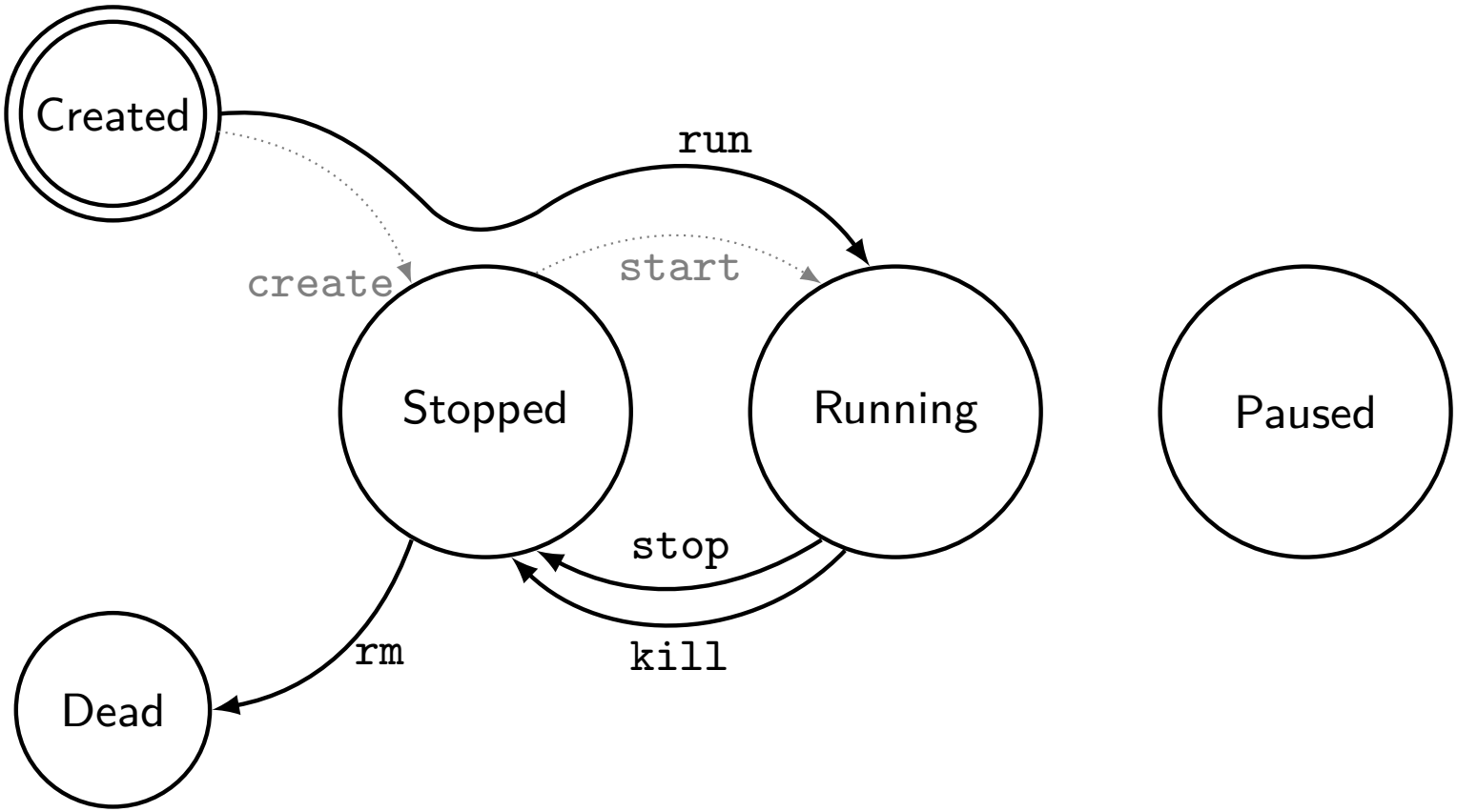
Container life cycle (continued)



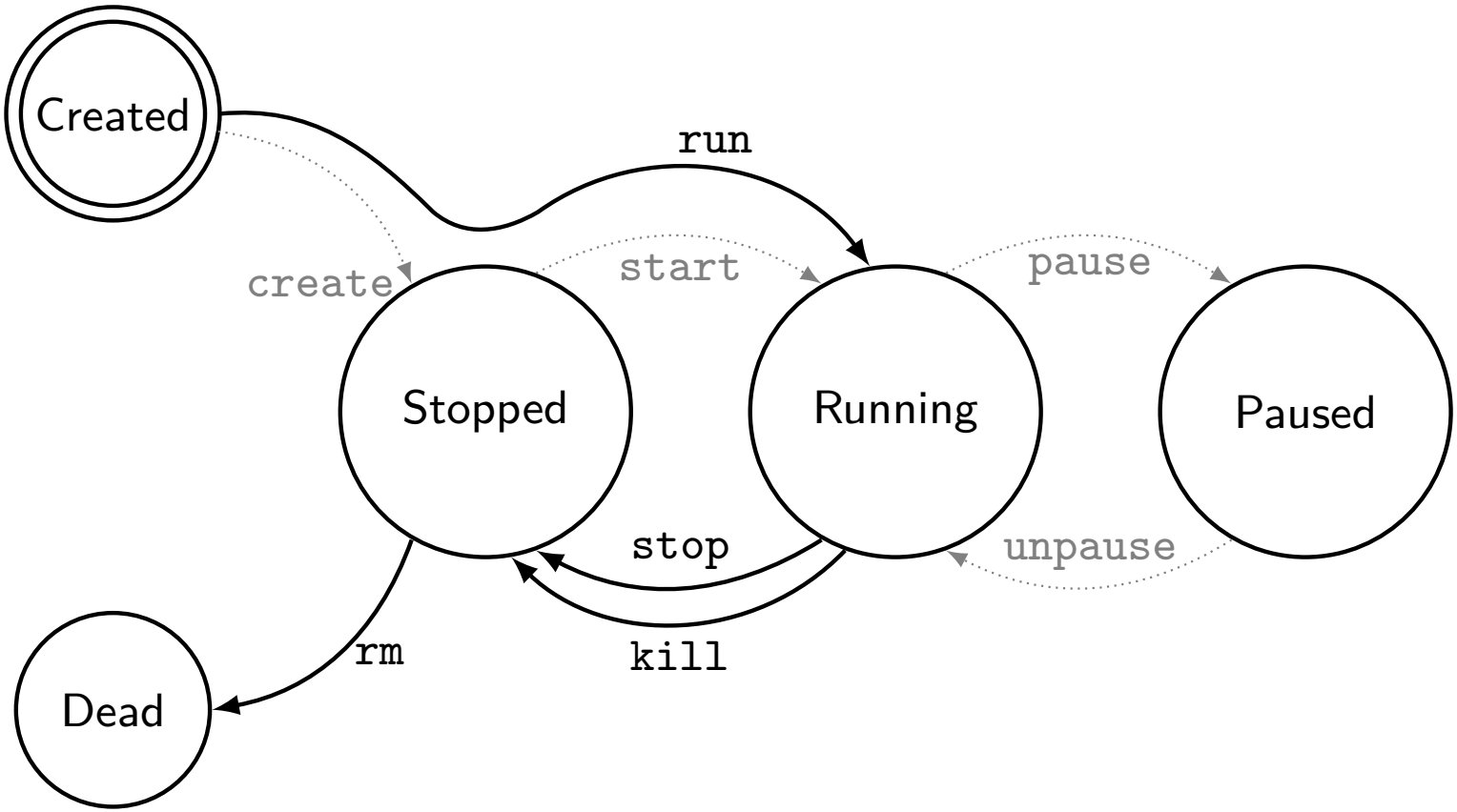
Container life cycle (continued)



Container life cycle (continued)



Container life cycle (continued)

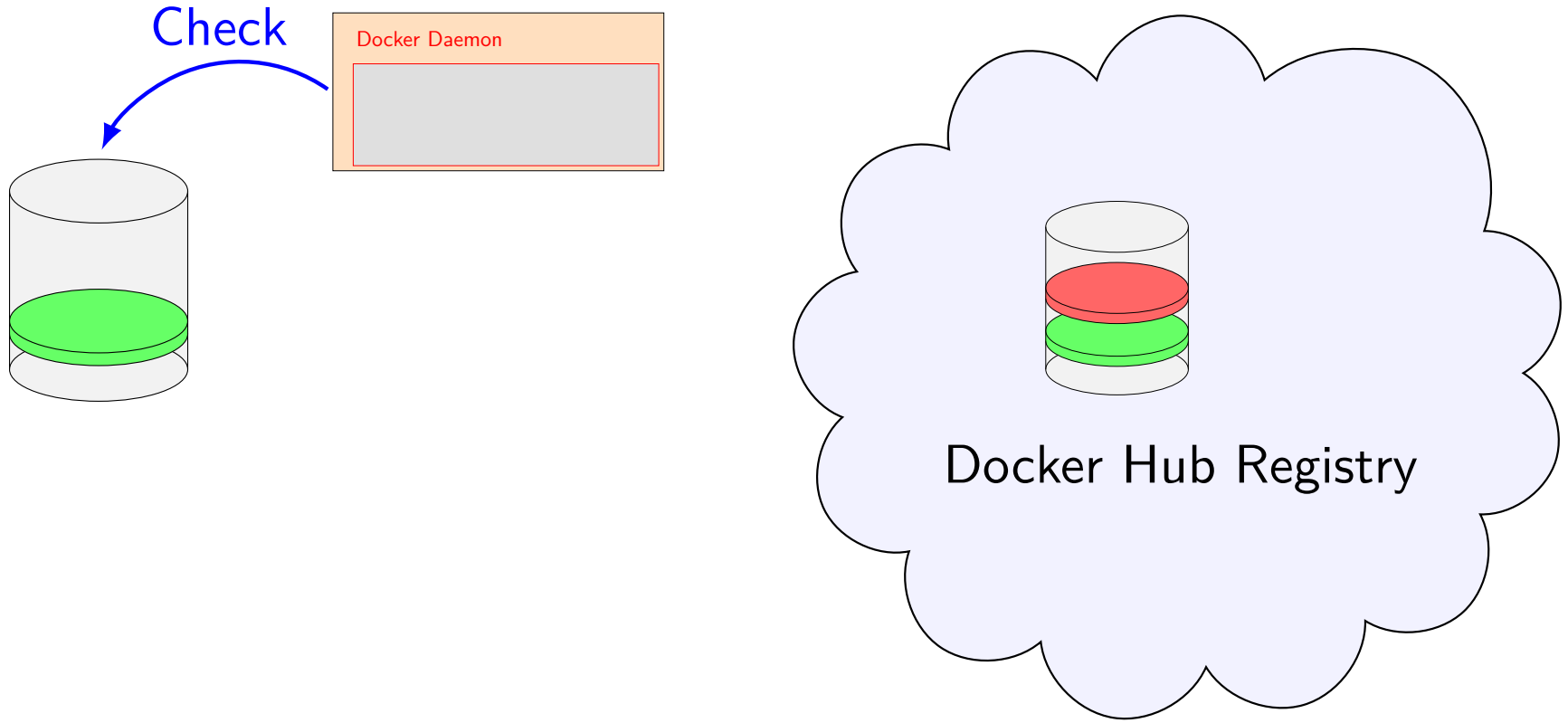


A word on life cycle

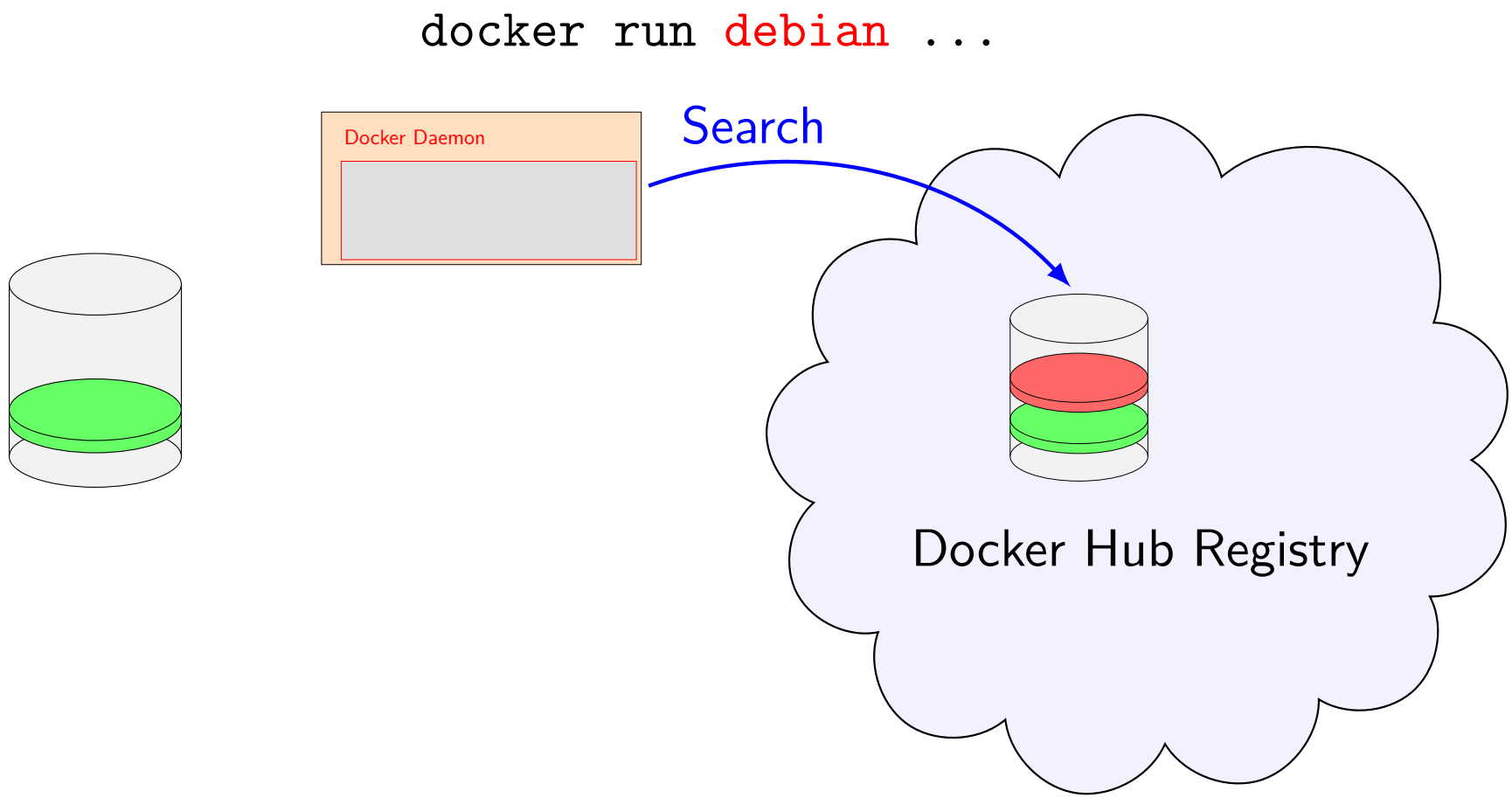
- Container file system is set up before the initial state (`created`)
  - It is cleaned up when going to the `Dead` state (with `rm`)
  - It is persistent across `stop`/`start`/`pause` operations
- The `kill` command sends a `SIGKILL` to the contained executable
- When running without a TTY, signals aren't forwarded
  - They are handled by the `docker` command, not by the contained executable
  - A `SIGINT` will therefore end the container with a `SIGKILL`

Docker Registry: local cache and registry

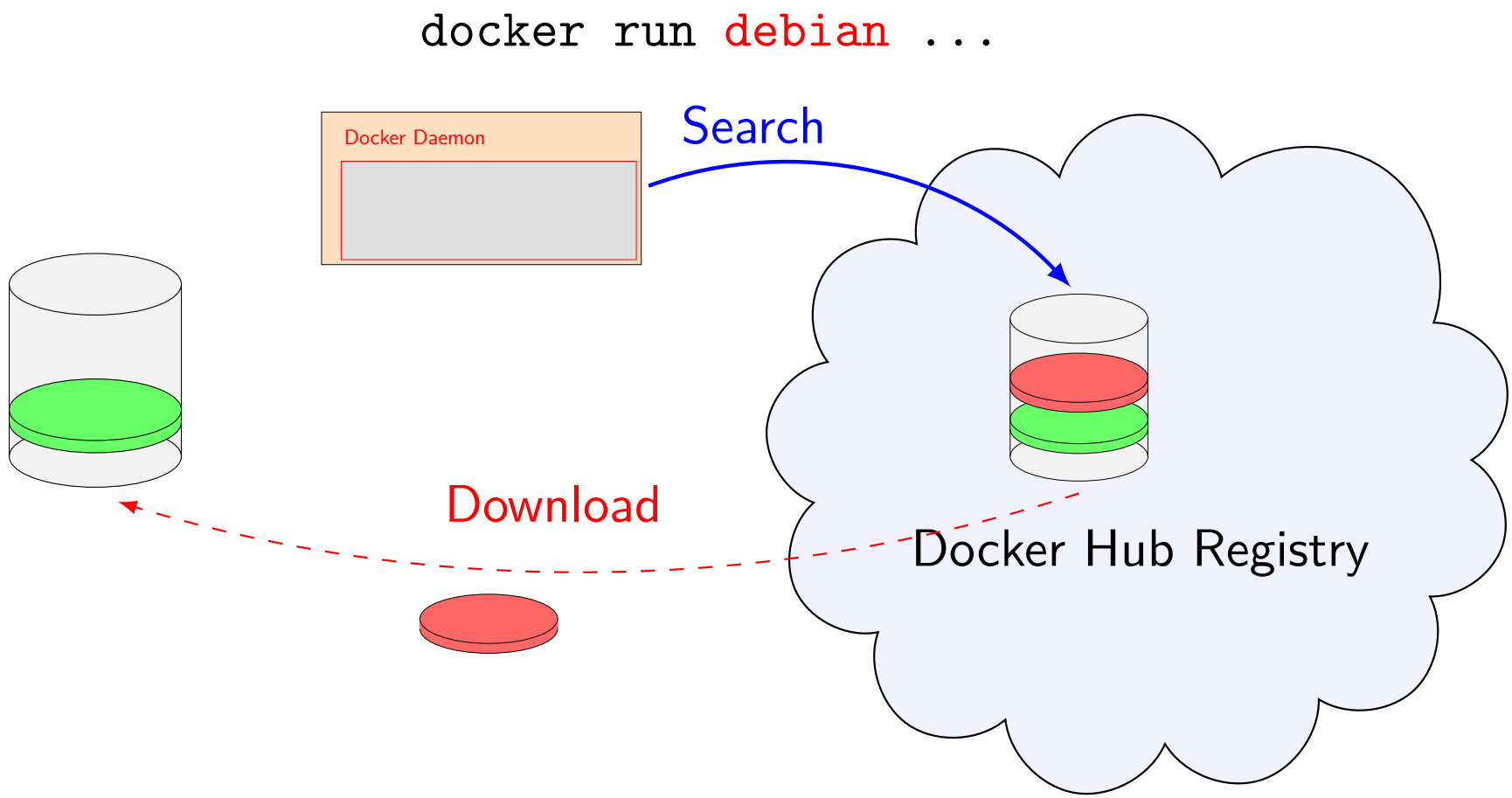
```
docker run debian ...
```



Docker Registry: local cache and registry

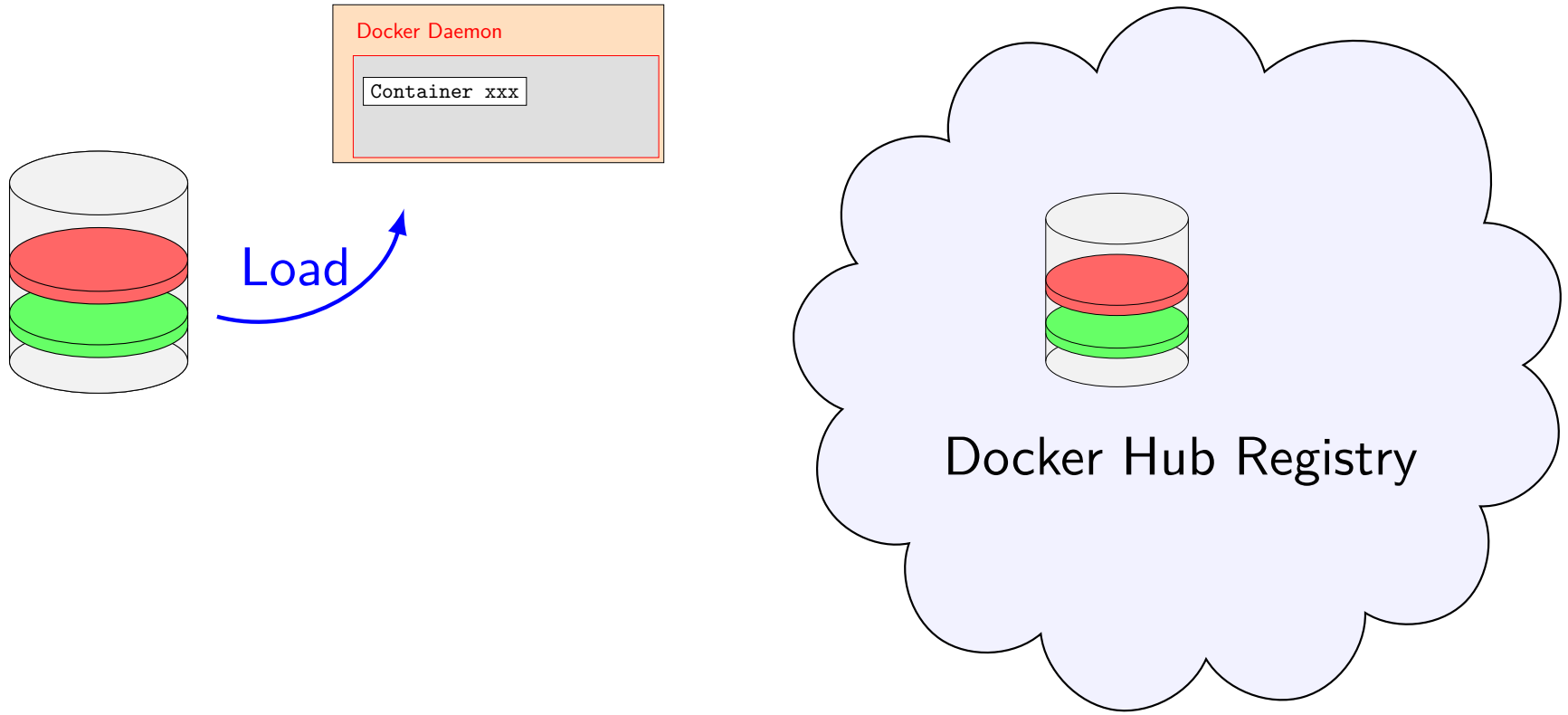


Docker Registry: local cache and registry



Docker Registry: local cache and registry

```
docker run debian ...
```



A journey through Docker Commands (1/6)

Step 1 Start a new container:

```
docker run -it ubuntu bash
```

A journey through Docker Commands (1/6)

Step 1 Start a new container:

```
docker run -it ubuntu bash
```

Step 2 Create a file in the container:

```
echo "Hello, World" > /root/greetings.txt
```

A journey through Docker Commands (1/6)

Step 1 Start a new container:

```
docker run -it ubuntu bash
```

Step 2 Create a file in the container:

```
echo "Hello, World" > /root/greetings.txt
```

Step 3 Print the hostname of the container (its ID):

```
hostname
```

A journey through Docker Commands (1/6)

Step 1 Start a new container:

```
docker run -it ubuntu bash
```

Step 2 Create a file in the container:

```
echo "Hello, World" > /root/greetings.txt
```

Step 3 Print the hostname of the container (its ID):

```
hostname
```

Step 4 Detach from the container:

Press `Ctrl+P`, `Ctrl+Q`

Step 5 Keep track the Container ID:

```
CID="ID_obtained_in_step_3"
```

A journey through Docker Commands (2/6)

Step 6 Copy the file from the container:

```
docker cp ${CID}:/root/greetings.txt
 ↪  ${HOME}/greetings.txt
```

A journey through Docker Commands (2/6)

Step 6 Copy the file from the container:

```
docker cp ${CID}:/root/greetings.txt
 ↪  ${HOME}/greetings.txt
```

Step 7 Edit/create a file on the host:

```
echo "Hello from host" > ${HOME}/host.txt
```

A journey through Docker Commands (2/6)

Step 6 Copy the file from the container:

```
docker cp ${CID}:/root/greetings.txt
 ↪  ${HOME}/greetings.txt
```

Step 7 Edit/create a file on the host:

```
echo "Hello from host" > ${HOME}/host.txt
```

Step 8 Send the file to the container

```
docker cp ${HOME}/host.txt ${CID}:/root/host.txt
```

A journey through Docker Commands (3/6)

Step 9 Reconnect the container:

```
docker attach $CID
```

Step 10 Check the new file:

```
cat /root/host.txt
```

A journey through Docker Commands (3/6)

Step 9 Reconnect the container:

```
docker attach $CID
```

Step 10 Check the new file:

```
cat /root/host.txt
```

Step 11 Edit a file inside the container:

```
echo "toto=1" >> /etc/sysctl.conf
```

Step 12 Re-detach the container

A journey through Docker Commands (4/6)

Step 13 List the modified files:

```
docker diff $CID
```

A journey through Docker Commands (4/6)

Step 13 List the modified files:

```
docker diff $CID
```

Step 14 Look what has been written to `stdout`/`stderr`:

```
docker logs $CID
```

A journey through Docker Commands (4/6)

Step 13 List the modified files:

```
docker diff $CID
```

Step 14 Look what has been written to `stdout`/`stderr`:

```
docker logs $CID
```

Step 15 Export the content:

```
docker export --output content.tar $CID
```

A journey through Docker Commands (5/6)

Step 16 Execute a detached process:

```
docker exec -d $CID sleep 1h
```

A journey through Docker Commands (5/6)

Step 16 Execute a detached process:

```
docker exec -d $CID sleep 1h
```

Step 17 View running processes:

```
docker exec $CID ps aux
```

A journey through Docker Commands (5/6)

Step 16 Execute a detached process:

```
docker exec -d $CID sleep 1h
```

Step 17 View running processes:

```
docker exec $CID ps aux
docker top $CID
```

A journey through Docker Commands (5/6)

Step 16 Execute a detached process:

```
docker exec -d $CID sleep 1h
```

Step 17 View running processes:

```
docker exec $CID ps aux
docker top $CID aux
```

A journey through Docker Commands (5/6)

Step 16 Execute a detached process:

```
docker exec -d $CID sleep 1h
```

Step 17 View running processes:

```
docker exec $CID ps aux
docker top $CID aux
```

Step 18 Execute an interactive process:

```
docker exec -it $CID bash
```

A journey through Docker Commands (6/6)

Step 19 Stop the container (from the host):

```
docker stop $CID
```

A journey through Docker Commands (6/6)

Step 19 Stop the container (from the host):

```
docker stop $CID
```

Step 20 See reclaimable space:

```
docker system df
```

A journey through Docker Commands (6/6)

Step 19 Stop the container (from the host):

```
docker stop $CID
```

Step 20 See reclaimable space:

```
docker system df
```

Step 21 Clean up:

```
docker container prune
docker volume prune
docker image prune
```

A journey through Docker Commands (6/6)

Step 19 Stop the container (from the host):

```
docker stop $CID
```

Step 20 See reclaimable space:

```
docker system df
```

Step 21 Clean up:

```
docker container prune
docker volume prune
docker image prune
```

} ▶ `docker system prune`

Last but not least

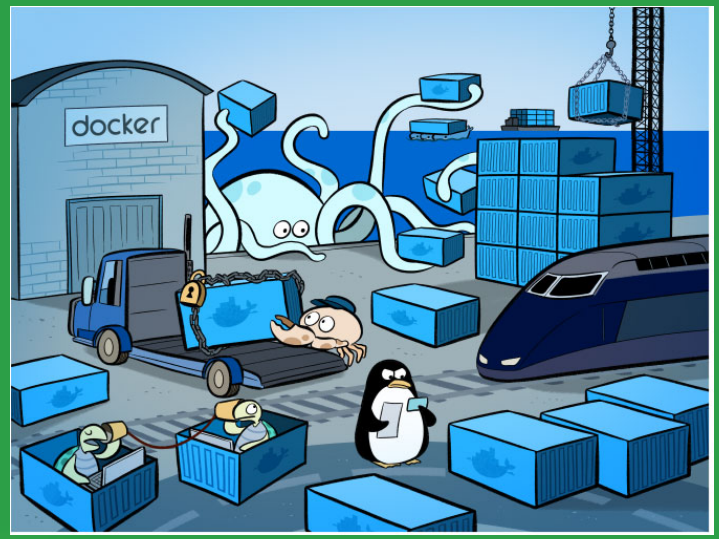
Step 22 Run a container and wait for it to finish:

```
CID=$(docker run --rm -d debian sleep 10)
docker wait $CID
```

Before we go...

Let Docker download images in background
(this can last some minutes)

```
docker pull debian:9.0
docker pull registry:2
docker pull nginx
docker pull hyper/docker-registry-web
```

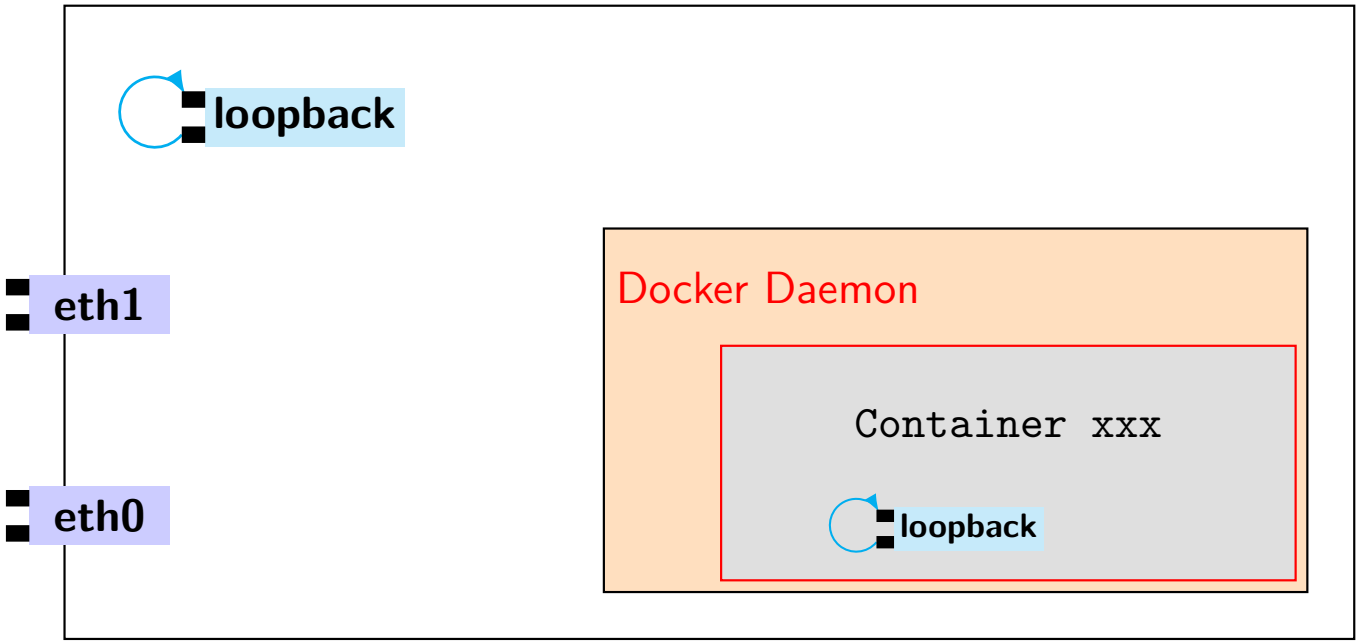


3
Basic interaction with the host

Network & Files

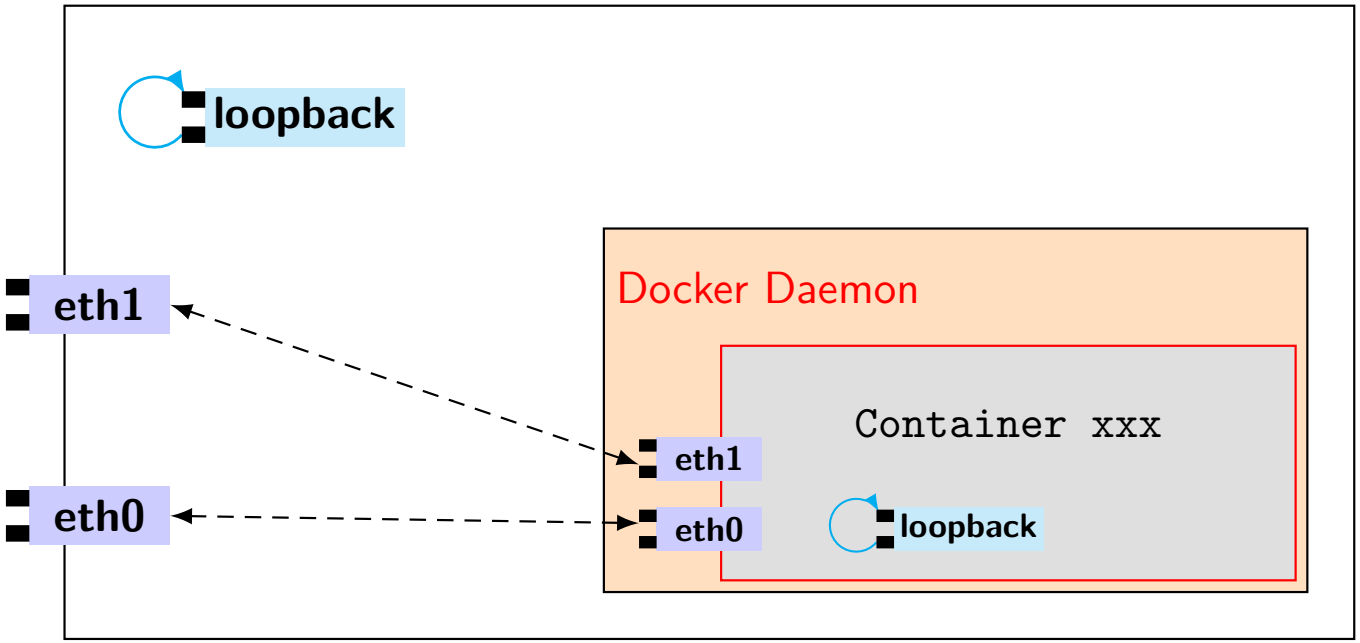
Docker default network configuration - none

`none` No network stack but `loopback`



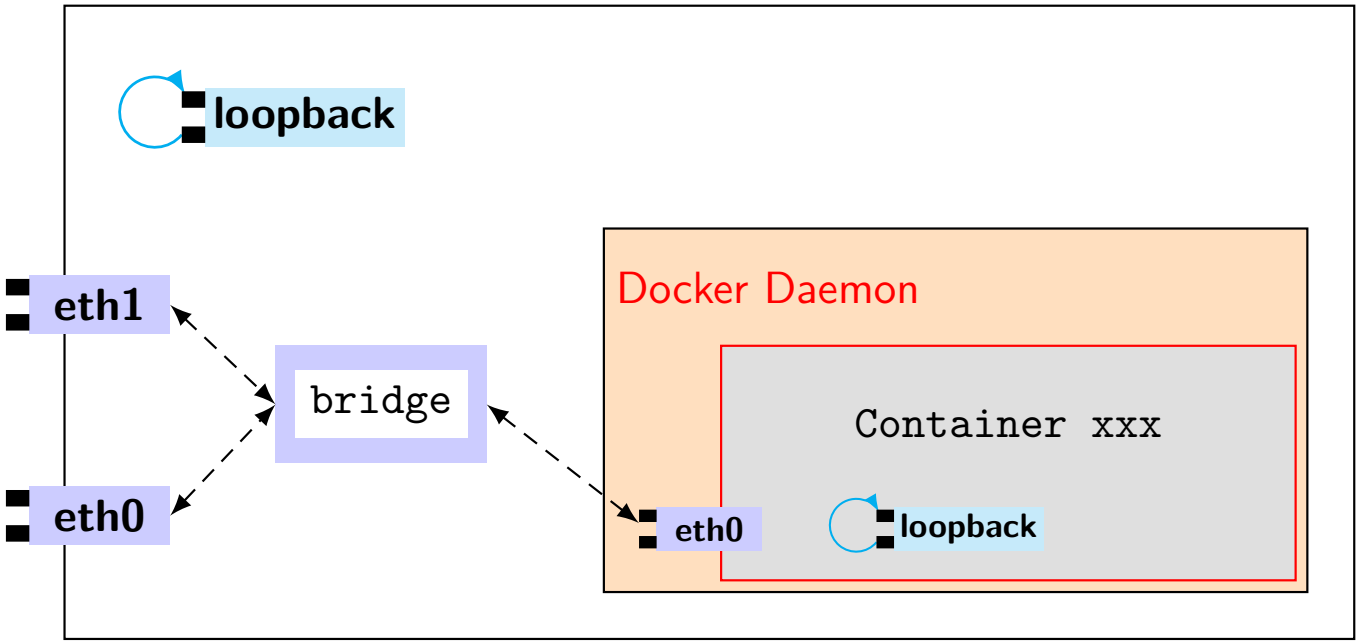
Docker default network configuration - `host`

`host` Host's network interfaces



Docker default network configuration - `bridge`

`bridge` Virtual switch handled by Docker (default behavior)



Docker networks - all configurations

- Default networks:

`none`	No network stack but `loopback`	
`host`	Host's network interfaces	
`bridge`	Virtual switch handled by Docker	(default)
`overlay`	A `bridge` network across hosts	(Swarm only)

- Custom networks:
  - `docker network create -d bridge my-net`
  - Only of type `bridge`, `overlay` or from a plugged-in type

- Multiple networks can be attached to a container

Docker networks - command setup

- Run a `debian` image with a specific network:
  - `docker run --rm -it debian ip addr`

Docker networks - command setup

- Run a `debian` image with a specific network:
  - `docker run --rm -it --network bridge debian ip addr`

Docker networks - command setup

- Run a `debian` image with a specific network:
  - `docker run --rm -it --network bridge debian ip addr`
    - Loopback and private IP
    - Access to external network (through the bridge to host's networks)

Docker networks - command setup

- Run a `debian` image with a specific network:
  - `docker run --rm -it --network bridge debian ip addr`
    - Loopback and private IP
    - Access to external network (through the bridge to host's networks)
  - `docker run --rm -it --network host debian ip addr`

Docker networks - command setup

- Run a `debian` image with a specific network:
  - `docker run --rm -it --network bridge debian ip addr`
    - Loopback and private IP
    - Access to external network (through the bridge to host's networks)
  - `docker run --rm -it --network host debian ip addr`
    - Loopback and host's IPs
    - Direct access to host's network interfaces

Docker networks - command setup

- Run a `debian` image with a specific network:
  - `docker run --rm -it --network bridge debian ip addr`
    - Loopback and private IP
    - Access to external network (through the bridge to host's networks)
  - `docker run --rm -it --network host debian ip addr`
    - Loopback and host's IPs
    - Direct access to host's network interfaces
  - `docker run --rm -it --network none debian ip addr`
    - Loopback only
    - No access to the outside world nor to the host

Publish a port: command line

- `-p`, `--publish`: gives access to a container port from the outside

`-p CC`	Host random port	$\Rightarrow$	Container port `CC`
`-p HH:CC`	Host port `HH`	$\Rightarrow$	Container port `CC`
`-p IP:HH:CC`	Same, but bound to host address IP		

Publish a port: command line

- `-p`, `--publish`: gives access to a container port from the outside

`-p CC`	Host random port	⇒	Container port `CC`
`-p HH:CC`	Host port `HH`	⇒	Container port `CC`
`-p IP:HH:CC`	Same, but bound to host address IP		

- `--expose`: defines a port to expose
  - *i.e.* made accessible by other containers
  - useful if Inter-Container-Communications (ICC) are disabled
  - equivalent to the `EXPOSE` Dockerfile command

Publish a port: example

- Run an `nginx` image:
  `docker run --rm -it -p 8080:80 nginx`

Publish a port: example

- Run an `nginx` image:
  `docker run --rm -it -p 8080:80 nginx`
  - Server available on `http://localhost:8080/`
  - Also from the host interfaces, if the firewall allows it

Publish a port: example

- Run an `nginx` image:
  `docker run --rm -it -p 8080:80 nginx`
  - Server available on `http://localhost:8080/`
  - Also from the host interfaces, if the firewall allows it

Figure: `nginx` is up & running

Docker volumes

- Kinds of volumes:

Bound volume	A host directory/file is mounted in the container
Data volume	Stored on host, in `/var/lib/docker/...`
Named volume	Volume created *a priori*, with `docker volume create`

Docker volumes

- Kinds of volumes:

Bound volume	A host directory/file is mounted in the container
Data volume	Stored on host, in `/var/lib/docker/...`
Named volume	Volume created *a priori*, with `docker volume create`

- Volume drivers: plug-ins to support new kinds of volumes
  - NetShare.io (NFS, CIFS, SMB), Nvidia, ...

Docker volumes: command line

- `--volume-driver`: the volume driver to use for this command line
  - Only one driver can be set per command line

Docker volumes: command line

- `--volume-driver`: the volume driver to use for this command line
  - Only one driver can be set per command line
- `-v`, `--volume`: defines a new volume

Docker volumes: command line

- `--volume-driver`: the volume driver to use for this command line
  - Only one driver can be set per command line
- `-v`, `--volume`: defines a new volume
  - `docker run -v /path ...`
    - Creates a *data* volume for the `/path` folder

Docker volumes: command line

- `--volume-driver`: the volume driver to use for this command line
  - Only one driver can be set per command line

- `-v`, `--volume`: defines a new volume
  - `docker run -v /path ...`
    - Creates a *data* volume for the `/path` folder
  - `docker run -v /host/path:/path ...`
    - Mounts a *bound* volume to `/path`
    - Most drivers also support a final `:ro` flag, to bind a read-only volume:
      `docker run -v /host/path:/path:ro ...`

Docker volumes: example

On the host, in a new folder:

- Create a simple HTML page: `./www/index.html`

```
<html>
<body><h1>Hello World, from Docker</h1></body>
</html>
```

- Create an `nginx` configuration: `./site.conf`

```
server {
    listen 80;
    root /www;
    autoindex on;
}
```

- Source files available on :

  `http://sed.inrialpes.fr/docker-tuto/index_docker.html`

Docker volumes: example

- Run the container with the following volumes:
  - `./site.conf` $\Rightarrow$ `/etc/nginx/conf.d/default.conf`
  - `./www/` $\Rightarrow$ `/www`

Docker volumes: example

- Run the container with the following volumes:
  - `./site.conf` ⇒ `/etc/nginx/conf.d/default.conf`
  - `./www/` ⇒ `/www`

```
docker run --rm \
  -p 8080:80 \
  -v $(pwd)/site.conf:/etc/nginx/conf.d/default.conf \
  -v $(pwd)/www:/www \
  nginx
```

Docker volumes: plug-ins

- Example: the NetShare.io plug-in
  - Plug-in to be installed separately;
    see `http://netshare.containx.io/`
  - Gives access to NFS & CIFS shared folders as volumes

- ```
  docker run \
      --volume-driver nfs \
      -v nfs-server/shared/path:/path ...
  ```
  - Note the lack of column ":" after the server name
  - No other kind of volume can be mounted on this line, unlike other NetShare volumes
  ```

Docker volumes: command line

How to use multiple volume drivers at once ?

Docker volumes: command line

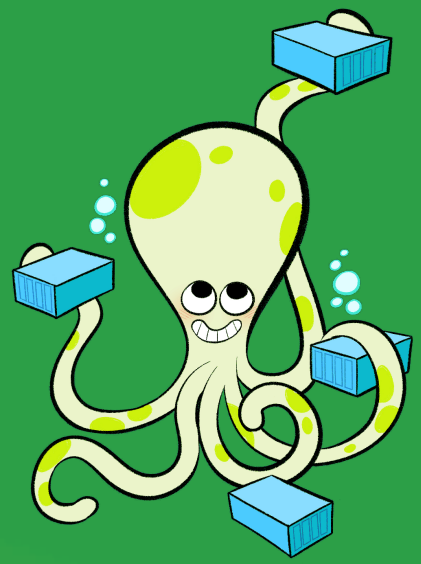
How to use multiple volume drivers at once ?

Solution: create a named volume

- `docker volume create -d nfs --name shared-data \`
  `-o share=nfs-server:/shared/path`
  - Note the `share=` format, equivalent to `fstab` options
- `docker run -v shared-data:/path ...`
  - No need for the `--volume-driver` option

4
Link containers together

Unity makes strength



Expose, Links & Networks

- Expose (`Dockerfile` or `run` argument)
  - Defines ports accessible by other containers, even without ICC
- Links (`run` argument, composition)
  - Indicates Docker that a container can communicate with another
  - Allows to give a network alias to access the container
- Networks
  - All containers of a network can communicate
  - No port restriction inside the network

Compositions: Docker Compose

- A Python script to manage sets of containers
  - The standalone version is recommended, see `https://docs.docker.com/compose/install`
  - `pip install docker-compose` on recent OSes
- Same capabilities as the `run` command
- Compositions written in YAML format

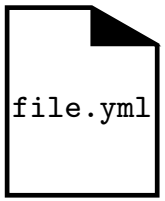
Sample composition

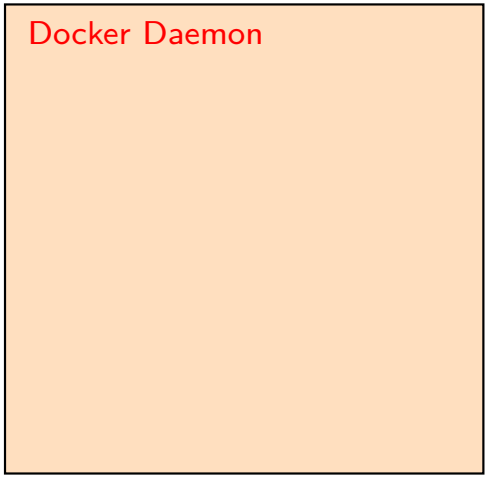
```
version: "3"

services:
  web:
    image: nginx
    ports:
     - "8080:80"
    links:
     - database:auth_db
    volumes:
     - ./site.conf:/etc/nginx/conf.d/default.conf
     - ./www:/www

  database:
    image: mysql
```

Principles





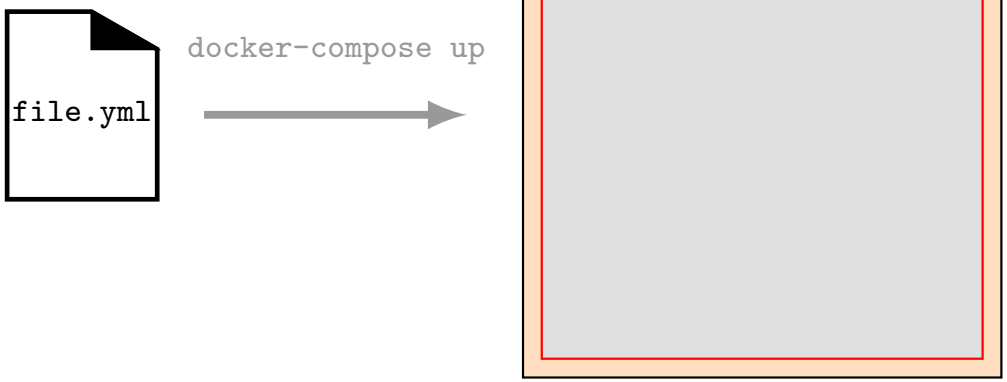
file.yml

```
version: "3"

services:
 web:
  image: nginx
  ports:
   - "8080:80"
  links:
   - database:auth_db
  volumes:
   - ./site.conf:[...]/default.conf
   - ./www:/www

 database:
  image: mysql
```

Principles



- `docker-compose up -d`

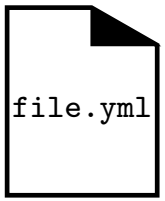
file.yml

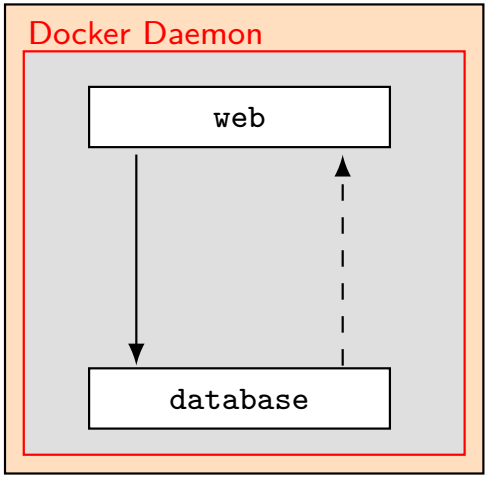
```
version: "3"

services:
 web:
  image: nginx
  ports:
   - "8080:80"
  links:
   - database:auth_db
  volumes:
   - ./site.conf:[...]/default.conf
   - ./www:/www

 database:
  image: mysql
```

Principles





- `docker-compose up -d`

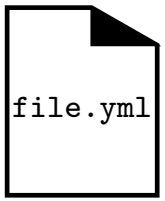
`file.yml`

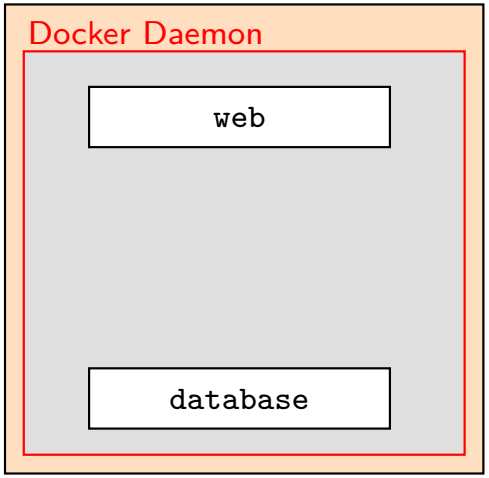
```
version: "3"

services:
 web:
  image: nginx
  ports:
   - "8080:80"
  links:
   - database:auth_db
  volumes:
   - ./site.conf:[...]/default.conf
   - ./www:/www

 database:
  image: mysql
```

Principles





- `docker-compose up -d`
- `docker-compose stop`

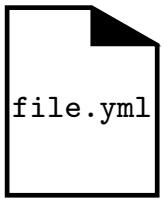
`file.yml`

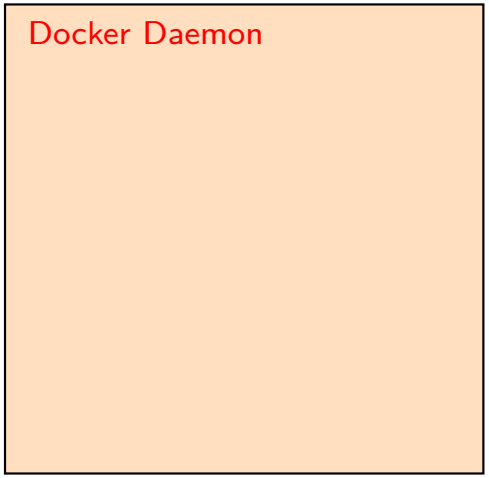
```
version: "3"

services:
 web:
  image: nginx
  ports:
   - "8080:80"
  links:
   - database:auth_db
  volumes:
   - ./site.conf:[...]/default.conf
   - ./www:/www

 database:
  image: mysql
```

Principles





file.yml

```
version: "3"

services:
 web:
  image: nginx
  ports:
   - "8080:80"
  links:
   - database:auth_db
  volumes:
   - ./site.conf:[...]/default.conf
   - ./www:/www

 database:
  image: mysql
```

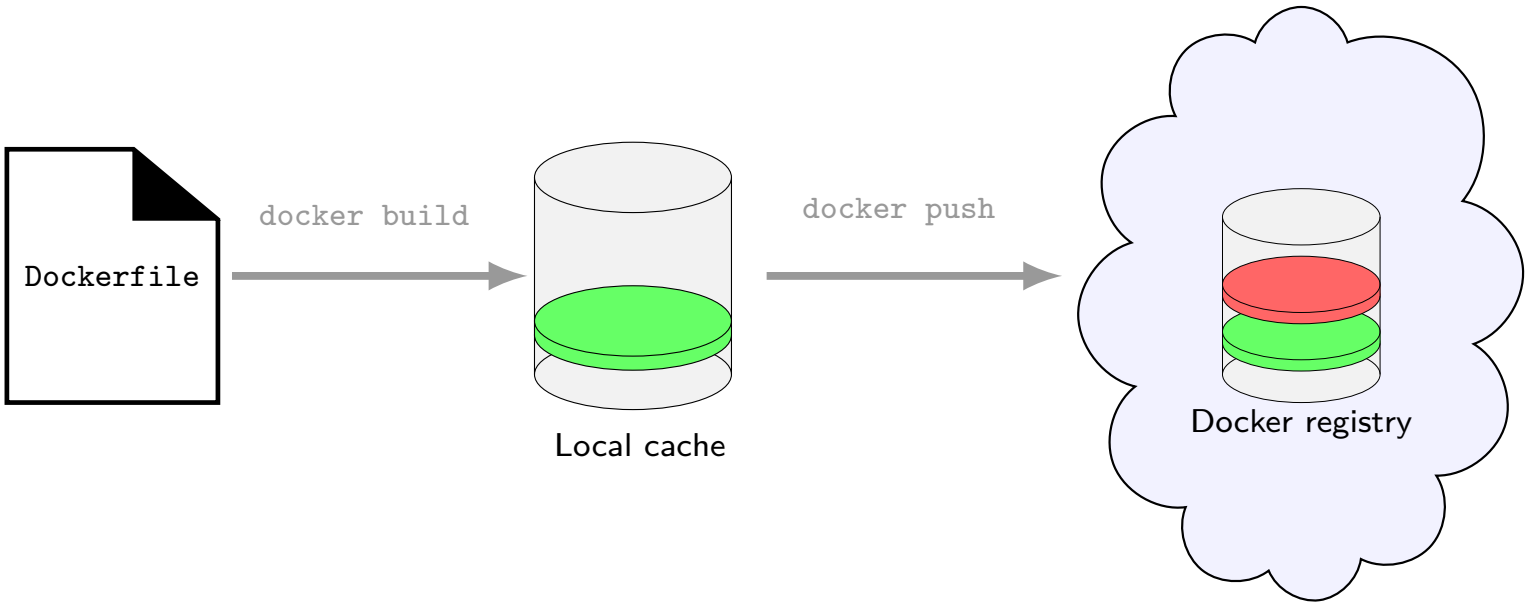
- `docker-compose up -d`
- `docker-compose stop`
- `docker-compose down`

5
Create a Docker image

Bring your own container

Principles

`Dockerfile`	File describing how the image is built
`docker build`	Command line to build the `Dockerfile`
Local cache	Local image store
`docker push`	Command line to send the image to a registry
Docker registry	Image store (public or private)



Dockerfile: first example

- Objective:
  - Provide a SOCKS5 proxy found on Gist
- Required environment:
  - `wget` to download the `socks5.py` script

Dockerfile: first example

- Objective:
  - Provide a SOCKS5 proxy found on Gist
- Required environment:
  - Python 3.4+
  - `wget` to download the `socks5.py` script

Dockerfile: first example

- Objective:
  - Provide a SOCKS5 proxy found on Gist
- Required environment:
  - Debian 9.0 (as it provides Python 3.4)
  - Python 3.4+
  - `wget` to download the `socks5.py` script

Dockerfile: first example

- Objective:
  - Provide a SOCKS5 proxy found on Gist
- Required environment:
  - Debian 9.0 (as it provides Python 3.4)
  - Python 3.4+
  - `wget` to download the `socks5.py` script
- Dockerfile available at:

  `http://sed.inrialpes.fr/docker-tuto/index_docker.html`

Dockerfile: first example

```
FROM debian:9.0
```

Parent image

Name: Debian (official)

Tag: 9.0

Dockerfile: first example

```
FROM debian:9.0
LABEL maintainer "thomas.calmant@inria.fr"
```

Meta information

- Maintainer, version, ...
- Visible in `docker inspect`

Dockerfile: first example

```
FROM debian:9.0
LABEL maintainer "thomas.calmant@inria.fr"
# Ensure a sane environment
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive
```

Environment variables

- Set for the whole container
- Can't reference current line

Dockerfile: first example

```
FROM debian:9.0
LABEL maintainer "thomas.calmant@inria.fr"
# Ensure a sane environment
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive

# Update the image & install some tools
RUN apt-get update --fix-missing && \
    apt-get -y dist-upgrade && \
    apt-get install -y \
            ca-certificates wget python3 && \
    apt-get clean
```

Dependencies setup

- Update the system first
- Install only what's necessary
- Regroup install commands
- Clean up caches immediately

Dockerfile: first example

```
FROM debian:9.0
LABEL maintainer "thomas.calmant@inria.fr"
# Ensure a sane environment
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive

# Update the image & install some tools
RUN apt-get update --fix-missing && \
    apt-get -y dist-upgrade && \
    apt-get install -y \
            ca-certificates wget python3 && \
    apt-get clean
# Download the SOCKS5 server & set it executable
RUN wget -O /opt/socks5.py \
                https://[...]/socks5.py && \
    chmod +x /opt/socks5.py && \
    sync
```

Software setup

- Avoid keeping temporary files
- Decompress while downloading
- Clean up immediately

Dockerfile: first example

```
FROM debian:9.0
LABEL maintainer "thomas.calmant@inria.fr"

# Ensure a sane environment
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive


# Update the image & install some tools
RUN apt-get update --fix-missing && \
    apt-get -y dist-upgrade && \
    apt-get install -y \
            ca-certificates wget python3 && \
    apt-get clean
# Download the SOCKS5 server & set it executable
RUN wget -O /opt/socks5.py \
                https://[...]/socks5.py && \
    chmod +x /opt/socks5.py && \
    sync
# Set the default entry point & arguments
ENTRYPOINT ["/usr/bin/python3", "/opt/socks5.py"]
CMD        ["-p", "1080"]
```

Behavior setup

- Set default program and arguments

Dockerfile: Build an image

Step 1 Download the Dockerfile:

`http://sed.inrialpes.fr/docker-tuto/socks5/Dockerfile`

Dockerfile: Build an image

Step 1 Download the Dockerfile:

`http://sed.inrialpes.fr/docker-tuto/socks5/Dockerfile`

Step 2 Build the image:

```
docker build -t aubergiste .
```

Dockerfile: Build an image

Step 1 Download the Dockerfile:

`http://sed.inrialpes.fr/docker-tuto/socks5/Dockerfile`

Step 2 Build the image:

```
docker build -t aubergiste .
```

- tag (name) of the image

Dockerfile: Build an image

Step 1 Download the Dockerfile:

`http://sed.inrialpes.fr/docker-tuto/socks5/Dockerfile`

Step 2 Build the image:

```
docker build -t aubergiste .
```

- tag (name) of the image
- context: folder where to find files referenced in `Dockerfile`

Dockerfile: Build an image

Step 1 Download the Dockerfile:

`http://sed.inrialpes.fr/docker-tuto/socks5/Dockerfile`

Step 2 Build the image:

```
docker build -t aubergiste .
```

- tag (name) of the image
- context: folder where to find files referenced in `Dockerfile`

Step 3 Run it:

```
docker run --rm -it -p 1080:1080 aubergiste
```

Dockerfile: Build an image

Step 1 Download the Dockerfile:

`http://sed.inrialpes.fr/docker-tuto/socks5/Dockerfile`

Step 2 Build the image:

```
docker build -t aubergiste .
```

- tag (name) of the image
- context: folder where to find files referenced in `Dockerfile`

Step 3 Run it:

```
docker run --rm -it -p 1080:1080 aubergiste
```

Step 4 Give it a parameter:

```
docker run --rm -it aubergiste --help
```

Dockerfile: Basic instructions

Description

`FROM`	Parent image
`LABEL`	Metadata to describe the image
`ARG`	Variable to be given at build time

Instructions

`ENV`	Sets environment variables
`RUN`	Executes shell commands
`SHELL`	Sets the shell executing `RUN` commands
`WORKDIR`	Sets the working directory

Behavior

`ENTRYPOINT`	Sets the command line to execute (`$SHELL` by default)
`CMD`	Sets the default arguments for the entry point

Dockerfile: More instructions

Files

`COPY`	Copies/Downloads a file to the image (**recommended**)
`ADD`	Copies/Downloads and auto-decompresses a file
`VOLUME`	Declares a folder as a data volume

Network

`EXPOSE`	Declares ports to expose to other containers

User management

`USER`	Switches to the given user. The user must have been created with `useradd`

Dockerfile: Change user

```
FROM debian:9.0
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive
```

Initial layers

- Shared with the previous image

Dockerfile: Change user

```
FROM debian:9.0
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive

ARG user=karadoc
ARG home=/kaamelott/kitchen
```

Build arguments

- With a default value

Dockerfile: Change user

```
FROM debian:9.0
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive

ARG user=karadoc
ARG home=/kaamelott/kitchen

# Create the user and its directory
RUN mkdir -p $home &&\
    useradd $user --home-dir $home && \
    chown -R $user: $home
```

Create the user and its directory

Dockerfile: Change user

```
FROM debian:9.0
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive

ARG user=karadoc
ARG home=/kaamelott/kitchen

# Create the user and its directory
RUN mkdir -p $home &&\
    useradd $user --home-dir $home && \
    chown -R $user: $home

# Switch to the new user
USER $user
```

Switch to the new user

- only a new `USER` command can switch back to `root`

Dockerfile: Change user

```
FROM debian:9.0
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    DEBIAN_FRONTEND=noninteractive

ARG user=karadoc
ARG home=/kaamelott/kitchen

# Create the user and its directory
RUN mkdir -p $home &&\
    useradd $user --home-dir $home && \
    chown -R $user: $home

# Switch to the new user
USER $user

# Change working directory
WORKDIR $home
RUN echo "alias ll='ls -l'" > ~/.bashrc
```

Run commands with the new user

Docker images in a nutshell

- Stored as layers of modifications
  - Layers are shared between images

Docker images in a nutshell

- Stored as layers of modifications
  - Layers are shared between images
- Named in the `<name>:<tag>` format
  - Default *tag*: `latest`
  - The name can be prefixed by the address of a custom registry

Docker images in a nutshell

- Stored as layers of modifications
  - Layers are shared between images
- Named in the `<name>:<tag>` format
  - Default *tag*: `latest`
  - The name can be prefixed by the address of a custom registry
- Stored in a Docker Registry
  - Either the official Docker Hub (hub.docker.com)
  - or a private instance of the `registry` image
  - or a compatible registry (Nexus plugin, ...)

Docker images in a nutshell

- Local cache: `/var/lib/docker/<driver>`
- Available drivers:

Overlay2	Replaces AUFS on Debian
AUFS	Historic, fallback on Debian flavor
Device Mapper	Historic, default on Red Hat flavor
BTRFS	Default on Suse, could replace Device Mapper
ZFS	"*Not recommended [...] unless you have substantial experience with ZFS on Linux*"

- Configuration:
  - `storage-driver` in `/etc/docker/daemon.json`

Docker Registry: where images are found

- Official registry: hub.docker.com
- Private registries
  - based on the official `registry` image
  - implement the registry REST API (Nexus plugin, ...)
- Registries must provide a valid certificate
  - self-signed certificates should be stored in `/etc/docker/certs.d/<registry>/ca.crt` to be fully accepted

Docker Registry: where images are found

- Official registry: hub.docker.com
- Private registries
  - based on the official `registry` image
  - implement the registry REST API (Nexus plugin, ...)
- Registries must provide a valid certificate
  - self-signed certificates should be stored in `/etc/docker/certs.d/<registry>/ca.crt` to be fully accepted
- User authentication using `docker login` and `docker logout`

Setup a Docker registry

Step 1 Download the composition setup at:

`http://sed.inrialpes.fr/docker-tuto/index_docker.html`

Step 2 Decompress the file and run the composition:

`docker-compose up -d`

(download can take a while)

Step 3 Wait for the server to come up: `https://localhost`

Docker image: commands

Step 4 Build an image:

```
docker build -t aubergiste:1.0 .
```

Docker image: commands

Step 4 Build an image:

```
docker build -t aubergiste:1.0 .
```

Step 5 Tag it as *latest*:

```
docker tag aubergiste:1.0 aubergiste
```

Docker image: commands

Step 4 Build an image:
`docker build -t aubergiste:1.0 .`

Step 5 Tag it as *latest*:
`docker tag aubergiste:1.0 aubergiste`

Step 6 See the content of the local cache:
`docker images`

Docker image: commands

Step 7 Tag the image for a private registry:
`docker tag aubergiste localhost/aubergiste`

Docker image: commands

Step 7 Tag the image for a private registry:
`docker tag aubergiste localhost/aubergiste`

Step 8 Upload it:
`docker push localhost/aubergiste`

Step 9 Remove the local reference:
`docker rmi aubergiste`

What about `docker commit`?

- Principle: save the current state of a container as a image
- Some use cases:
  - when an application setup is interactive
  - when the setup comes from a volume
  - when the setup is large (10GB+)
- Usage:

```
docker commit ${CID} <image>:<tag>
```

6
Security
(kind of)

Pøůịc @Le_Pouic · 1 mai
Il suffit d'enlever un seul cadenas pour pouvoir tout ouvrir.

(aussi connu sous "allégorie de la sécurité informatique en entreprise")
pic.twitter.com/sFI0vU846C

What Docker is about

- Docker isolates **processes** from the host

What Docker is about

- Docker isolates **processes** from the host
  - Untrusted applications should be executed with high isolation

What Docker is about

- Docker isolates **processes** from the host
  - Untrusted applications should be executed with high isolation
  - Avoid loosing the leash:
    - Avoid `--privileged`
    - Don't add capabilities to the container
    - Don't disable namespaces

What Docker is about

- Docker isolates **processes** from the host
  - Untrusted applications should be executed with high isolation
  - Avoid loosing the leash:
    - Avoid `--privileged`
    - Don't add capabilities to the container
    - Don't disable namespaces
- Docker **doesn't** isolate the **user** from the host
  - A user in the `docker` is `root` on the machine
  - Not suitable for children (and untrusted users)
  - "*With Great Power Comes Great Responsibility*"

What Docker is about

- Docker isolates **processes** from the host
  - Untrusted applications should be executed with high isolation
  - Avoid loosing the leash:
    - Avoid `--privileged`
    - Don't add capabilities to the container
    - Don't disable namespaces
- Docker **doesn't** isolate the **user** from the host
  - A user in the `docker` is `root` on the machine
  - Not suitable for children (and untrusted users)
  - "*With Great Power Comes Great Responsibility*"

```
docker run --rm -it -v /:/mnt/host debian
```

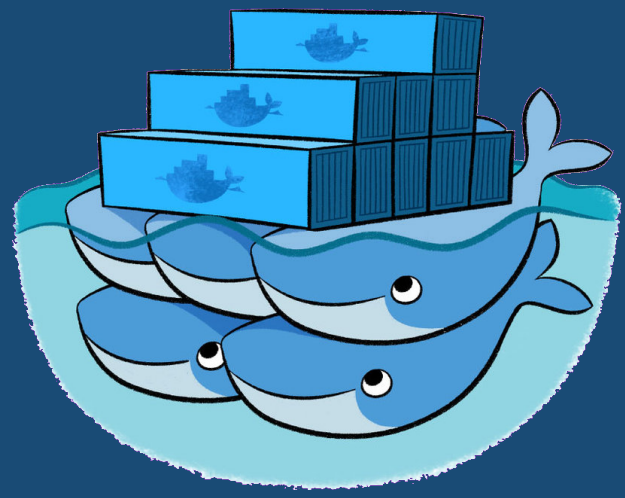
User namespace remap

- All actions from the container are seen as subuser's ones
- Privileged mode is disabled
- Configure the daemon: `/etc/docker/daemon.conf`
  - Activate *User Namespace Remap*: `userns-remap: default`
- Or, with a given sub user:
  - The user must exist in `/etc/passwd`
  - Configure the daemon: `userns-remap: bohort`
  - Set the `/etc/subuid`: `bohort:100000:65536`
  - Set the `/etc/subgid`: `bohort:100000:65536`

Why not?

- `docker run -it -d`
  ```
  --privileged --net=host
  -v /:/host
  -v /dev:/dev -v /run:/run
  -e sysimage=/host
  debian
  ```
- Inside the container:
  - `nsenter --mount=$sysimage/proc/1/ns/mnt --`
    `↪ /bin/bash`

7
Scale up with Swarm



What is Docker Swarm ?

- Docker on a multi-host cluster
  - Based on *overlay* networks
    (linking local *bridge* networks)

What is Docker Swarm ?

- Docker on a multi-host cluster
  - Based on *overlay* networks
    (linking local *bridge* networks)
- Adds the concept of *service*
  - Containers replicated or not on multiple machines
  - Restarted automatically
  - Migrated on host failure

What is Docker Swarm ?

- Docker on a multi-host cluster
  - Based on *overlay* networks
    (linking local *bridge* networks)
- Adds the concept of *service*
  - Containers replicated or not on multiple machines
  - Restarted automatically
  - Migrated on host failure
- At least one *manager*, no limit on *workers*
  - Managers act like workers
  - All nodes keep track of the Swarm state: the Swarm can fully restart if at least one node stays alive
  - `swarm` commands can only be run on managers

Setup a Swarm

- On the first manager host (*swarm leader*):
  - `docker swarm init`
  - `docker swarm join-token manager`
  - `docker swarm join-token worker`
- On other hosts (*swarm nodes*):
  - `docker swarm join --token SWMTKN-...\`
    `<manager-IP>:2377`

Nodes Handling

- Nodes inspection:
  - `docker node ls`
  - `docker node inspect <node>`
  - `docker node ps <node>`
  - `docker node rm <node>`

Nodes Handling

- Nodes inspection:
  - `docker node ls`
  - `docker node inspect <node>`
  - `docker node ps <node>`
  - `docker node rm <node>`
- Node mode switch:
  - `docker node promote <node>`
  - `docker node demote <node>`

Define a service

- Similar capabilities as the `run` command
- Useful commands:
  - `docker service create ...`
  - `docker service ls`
  - `docker service ps <service>`
  - `docker service rm <service>`

Define a service

- Similar capabilities as the `run` command
- Useful commands:
  - `docker service create ...`
  - `docker service ls`
  - `docker service ps <service>`
  - `docker service rm <service>`
- Sample:

```
docker service create --name postgres \
    --env POSTGRES_PASSWORD="toto" \
    --env POSTGRES_USER=hive \
    --env POSTGRES_DB=metastore \
    postgres:9.5
```

Docker Swarm: Stacks

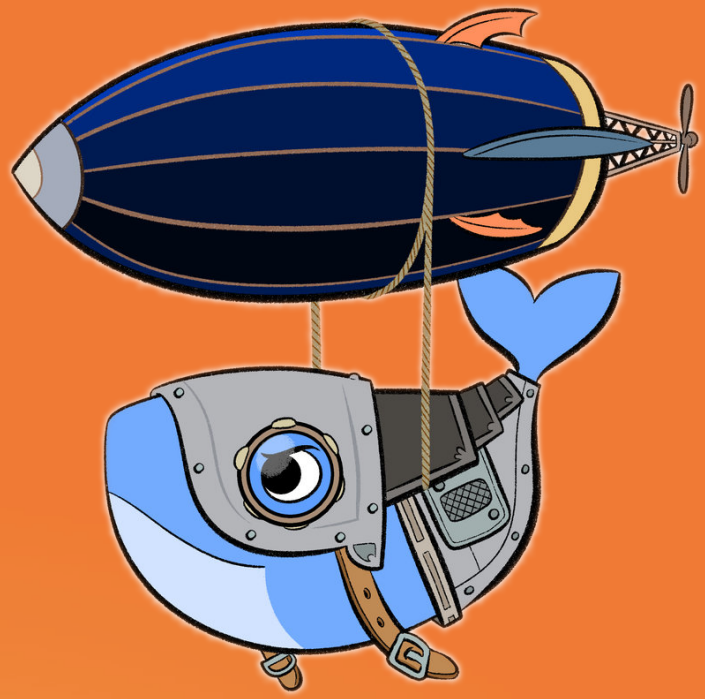
- Compatible with `docker-compose` V3 files
  - With some limitations: no `links` (mandatory use of networks)
  - And some new capabilities: `deploy` configuration
- `docker deploy --compose-file ./hdfs_stack.yml hdfs`

```
version: '3'
services:
  namenode:
    image: registry/hdfs-namenode
    env_file: ./hadoop.env
    environment:
      CLUSTER_NAME: tyrex
    ports:
      - "8020:8020"
      - "50070:50070"
    networks:
      - tls-net
    volumes:
      - /local/namenode:/dfs/name
    deploy:
      placement:
        constraints:
          - node.hostname == realhost

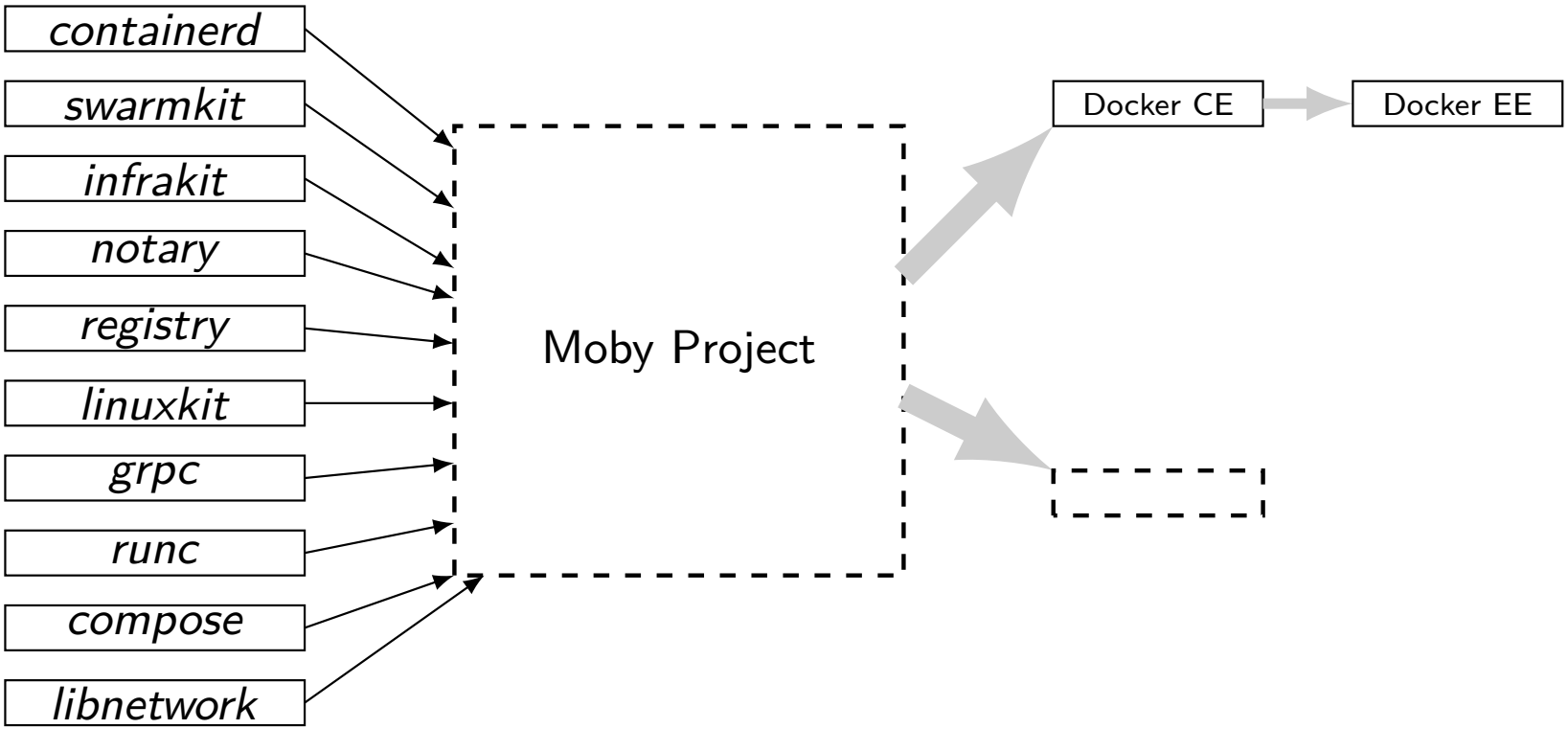
  datanode:
    image: registry/hdfs-datanode
    env_file: ./hadoop.env
    networks:
      - tls-net
    volumes:
      - /local/datanode:/dfs/data
    deploy:
      mode: global

networks:
  tls-net:
    external: true
```

8
Miscellaneous



Moby project



Docker on Nvidia

- Requires a working CUDA installation on the host
- Requires the CUDA driver and libraries in each container
- Provides a special volume allowing access to the GPUs
  - The `nvidia-docker` command wraps the `docker` one to always add this volume
  - Other volumes must be attached using a *Named Volume*

Docker on ARM

- Same Docker release as desktop
- Only works with `arm` images
  - Most are from `armhf` on the Docker Hub
  - `https://hub.docker.com/u/armhf/`
- Sample usage on a Raspberry Pi:
  - `http://blog.alexellis.io/getting-started-with-docker-on-raspberry-pi/`

Docker on Windows

- Requires Windows 10 Pro or Windows Server 2016
  - with the "`Containers`" and "`Hyper-V`" features
- Two base images are available:
  - `microsoft/windowsservercore`
  - `microsoft/nanoserver` (for 64 bits apps only)
- Isolation based on processes or Hyper-V
- `docker info`:

```
[...]
Server Version: 17.03.1-ce
Storage Driver: windowsfilter
Plugins:
 Network: l2bridge l2tunnel nat null overlay transparent
Default Isolation: hyperv
Kernel Version: 10.0 14393 (14393.953...)
Docker Root Dir: C:\ProgramData\Docker
[...]
```

Docker on Windows

```
FROM microsoft/windowsservercore

SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop';"]

# Install Python
RUN (new-object System.Net.WebClient).Downloadfile( \
        'https://www.python.org/ftp/python/3.5.3/python-3.5.3.exe', \
        'C:\python-setup.exe')
RUN start-process -filepath C:\python-setup.exe -passthru -wait \
        -argumentlist '/quiet InstallAllUsers=1 TargetDir=C:\Python35 ' \
        'CompileAll=1 PrependPath=1 Shortcuts=0 Include_tcltk=0'
RUN del C:\python-setup.exe

# Update environment
ENV PYTHONIOENCODING=utf-8:replace PYTHON_HOME="c:\Python35"
ENV PATH="${PYTHON_HOME};${PYTHON_HOME}\Scripts;C:\Windows\System32;${PATH}"

# Install requirements
RUN python -m pip install --upgrade pip
```

Thanks for your attention

Credits:

- CommitStrip
- Laurel
- xkcd

Thomas Calmant
thomas.calmant@inria.fr
SED/Tyrex
Montbonnot-Saint-Martin

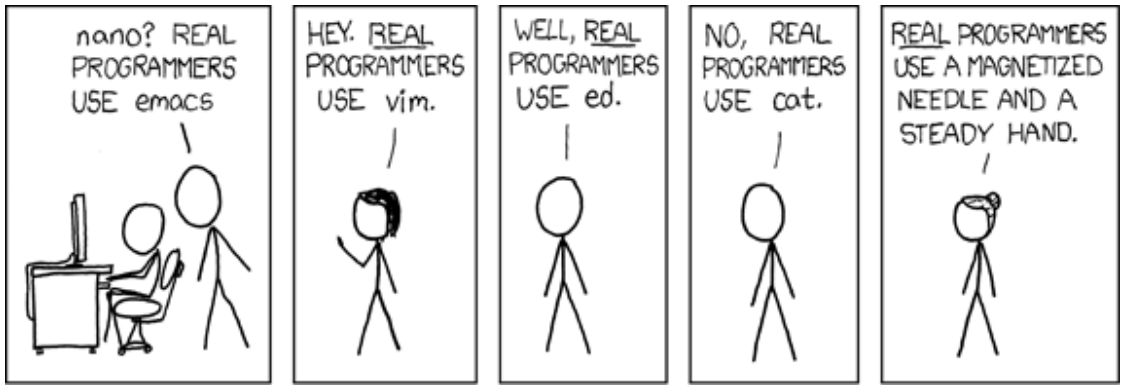

Hey guys, you're kind of quiet. Anyone want to troll us to wake us up?
Ah yeah. I've got an idea, but it's not really trolling...
Try anyway
Do you agree that Docker is just a kind of light-weight VM?
?!
NO
Pfff
You're crazy!
!
Docker isn't a VIRTUAL MACH
Just look at it for a second, God dammit!
You can do "ONE CONTAINER per PROCESS" or "ONE CONTAINER per APP"
for A SINGLE PROCESS ONLY!
Of course!
How could you SAY that?!
CommitStrip.com

Bearded man's cheat sheet

- *A posteriori* port forwarding:
  - `docker exec <CID> ip addr | grep 172.`
  - `iptables -t nat -A DOCKER -p tcp --dport 9000 -j DNAT --to-destination <CIP>:8080`

A word about `rkt`

- Started in 2014 to "*fix*" some Docker *flaws*
- Aims security (versus usability)
  - No central `root` daemon
- Compatible with the OpenContainer specification
  - ... so with Docker images
- Same conflict as "vim vs. emacs" or "etcd vs. consul"



Docker's Ecosystem