Introduction to Containerization with docker
1

A bit of context

NOT THAT KIND OF DOCKER...
The big questions

For administrators and packagers:

▸ How to ensure an application will work (nearly) everywhere ?
▸ How to avoid it messing with my system ?
▸ How to isolate the various components of my application ?
The big questions

For administrators and packagers:
- How to ensure an application will work (nearly) everywhere ?
- How to avoid it messing with my system ?
- How to isolate the various 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

<table>
<thead>
<tr>
<th>Transport</th>
<th>Isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ can be (un-)loaded/stacked efficiently</td>
<td>▶ OpenContainer Runtime Specification</td>
</tr>
<tr>
<td>▶ can be loaded on ships, trains, trucks, ...</td>
<td></td>
</tr>
<tr>
<td>▶ can be handled without being opened</td>
<td></td>
</tr>
</tbody>
</table>

OpenContainer Image Specification

ISO-standard sizes (5 classes)
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
- OpenContainer Image Specification

- are tracked with an identification number
- have ISO-standard sizes (5 classes)
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, DevOps-oriented), announced in a Lightning Talk at PyCon 2013
2015/06 Open Container Initiative (by Docker)
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, DevOps-oriented), announced in a Lightning Talk at PyCon 2013

2015/06  Open Container Initiative (by Docker)

2016/04  Singularity (HPC-oriented)
Virtualization vs. Containerization

- 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

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

1. A bit of context (we just did it)
2. Docker:
   - Playing with docker
   - Docker images & registry
   - Docker compositions
   - Security (kind of)
3. Singularity
   - Short introduction to singularity
   - Singularity vs. Docker
4. Miscellaneous & Bonus (if you’re good 😊)
2

Playing with docker

Because nothing beats the command line
Warm up

- Check if docker works:
  - `docker info`
  - `docker run hello-world`

- If it fails...
  - Check if docker is installed (docker-ce package)
    - `docs.docker.com/install/linux/docker-ce/debian/`
  - Check if your user is in the docker group:
    - `groups | grep docker`
  - If not:
    - Add yourself in: `sudo gpasswd -a $USER docker`
    - Restart your session (terminal won’t be enough)
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
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
Hands on: Running a container

- docker run debian

- 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
  - -a: also shows stopped containers

- docker rm -f <CID/name>
  - -f: stops the container if necessary
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)

- 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
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

- `docker rm -f <CID/name>`
  - Removes a stopped container
  - `-f` stops the container if necessary
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
Docker Registry: local cache and registry

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

Check

Docker Hub Registry

Docker Daemon
Docker Registry: local cache and registry

docker run debian ...
Docker Registry: local cache and registry

```
docker run debian ...
```
Docker Registry: local cache and registry

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

Created → Stopped → Running → Paused

Stopped → Running (run)

Running → Paused (stop)

Stopped → Dead (rm)

Dead → Created (rm)
Container life cycle

- Created
- Stopped
- Running
- Paused
- Dead

Actions:
- run
- stop
- rm
- kill
Container life cycle

- Created
- Stopped
- Running
- Paused
- Dead

- create
- start
- run
- stop
- rm
- kill
Container life cycle

Created → Run → Running → Pause → Paused
Created → Stopped → Stop → Dead
Created → Start → Running
Created → Run → Paused
Created → Run → Unpause
Created → Stop → Dead
Created → Stop → Kill
Created → Run → Start
Created → Run → Pause
Created → Run → Unpause

create → Created
run → Running
start → Running
run → Paused
pause → Paused
unpause → Paused
stop → Stopped
kill → Dead
rm → Dead
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
A journey through Docker Commands (1/6)

Step 1
(Host)
Start a new container:
docker run -it ubuntu bash
A journey through Docker Commands (1/6)

Step 1  Start a new container:
(Host) docker run -it ubuntu bash

Step 2  Create a file in the container:
(Docker) echo "Hello, World" > /root/greetings.txt
### A journey through Docker Commands (1/6)

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Start a new container:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Host)</td>
<td>docker run -it ubuntu bash</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2</th>
<th>Create a file in the container:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Docker)</td>
<td>echo &quot;Hello, World&quot; &gt; /root/greetings.txt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3</th>
<th>Print the hostname of the container (its ID):</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Docker)</td>
<td>hostname</td>
</tr>
</tbody>
</table>
A journey through Docker Commands (1/6)

Step 1
(Host) Start a new container:
docker run -it ubuntu bash

Step 2
(Docker) Create a file in the container:
echo "Hello, World" > /root/greetings.txt

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

Step 4
(Docker) Detach from the container:
Press Ctrl+P Ctrl+Q

Step 5
(Host) Keep track the Container ID:
CID="ID_obtained_in_step_3"
Step 6 (Host)
Copy the file from the container:
docker cp ${CID}:/root/greetings.txt greetings.txt

Step 7 (Host)
Edit/create a file on the host:
```bash
echo "Hello from host" > host.txt
```

Step 8 (Host)
Send the file to the container:
docker cp host.txt ${CID}:/root/host.txt
Step 6  
(Host)  
Copy the file from the container:  
docker cp ${CID}:/root/greetings.txt greetings.txt

Step 7  
(Host)  
Edit/create a file on the host:  
echo "Hello from host" > host.txt
Step 6  Copy the file from the container:
(Host)  docker cp ${CID}:/root/greetings.txt greetings.txt

Step 7  Edit/create a file on the host:
(Host)  echo "Hello from host" > host.txt

Step 8  Send the file to the container:
(Host)  docker cp host.txt ${CID}:/root/host.txt
A journey through Docker Commands (3/6)

Step 9  
(Host)  
Reconnect the container:  
docker attach $CID

Step 10  
(Docker)  
Check the new file:  
cat /root/host.txt
<table>
<thead>
<tr>
<th>Step</th>
<th>Command/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>9</strong></td>
<td>Reconnect the container:</td>
</tr>
<tr>
<td>(Host)</td>
<td><code>docker attach $CID</code></td>
</tr>
<tr>
<td><strong>10</strong></td>
<td>Check the new file:</td>
</tr>
<tr>
<td>(Docker)</td>
<td><code>cat /root/host.txt</code></td>
</tr>
<tr>
<td><strong>11</strong></td>
<td>Re-detach the container (Ctrl+P Ctrl+Q)</td>
</tr>
<tr>
<td>(Docker)</td>
<td></td>
</tr>
</tbody>
</table>
A journey through Docker Commands (4/6)

Step 12  List the modified files:
(Host)    docker diff $CID

Step 13  Look what has been written to stdout:
          docker logs $CID

Step 14  Export the content:
          docker export --output content.tar $CID
Step 12 (Host) List the modified files:
docker diff $CID

Step 13 (Host) Look what has been written to stdout/stderr:
docker logs $CID
A journey through Docker Commands (4/6)

Step 12  
(Host)  
List the modified files:  
docker diff $CID

Step 13  
(Host)  
Look what has been written to stdout/stderr:  
docker logs $CID

Step 14  
(Host)  
Export the content:  
docker export --output content.tar $CID
A journey through Docker Commands (5/6)

Step 15  (Host)  Execute a detached process:
        docker exec -d $CID sleep 1h
A journey through Docker Commands (5/6)

Step 15  Execute a detached process:
(Host)   docker exec -d $CID sleep 1h

Step 16  View running processes:
(Host)   docker exec $CID ps aux
A journey through Docker Commands (5/6)

Step 15  (Host)  Execute a detached process:
            docker exec -d $CID sleep 1h

Step 16  (Host)  View running processes:
            docker exec $CID ps aux
            docker top $CID
A journey through Docker Commands (5/6)

**Step 15**
(Host)
Execute a detached process:
```
docker exec -d $CID sleep 1h
```

**Step 16**
(Host)
View running processes:
```
docker exec $CID ps aux

docker top $CID aux

ds aux
```
A journey through Docker Commands (5/6)

Step 15  (Host)  Execute a detached process:
          docker exec -d $CID sleep 1h

Step 16  (Host)  View running processes:
          docker exec $CID ps aux
          docker top $CID aux
          ps aux

Step 17  (Host)  Execute an interactive process:
          docker exec -it $CID bash
A journey through Docker Commands (6/6)

Step 18  (Host)  Stop the container (from the host):
          docker stop $CID

Step 19  (Host)  See reclaimable space:
                docker system df

Step 20  (Host)  Clean up:
                docker container prune
                docker volume prune
                docker image prune

All in one:
            docker system prune
A journey through Docker Commands (6/6)

Step 18  Stop the container (from the host):
        (Host)        docker stop $CID

Step 19  See reclaimable space:
        (Host)        docker system df
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Stop the container (from the host)</td>
<td><code>docker stop $CID</code></td>
</tr>
<tr>
<td>19</td>
<td>See reclaimable space</td>
<td><code>docker system df</code></td>
</tr>
</tbody>
</table>
| 20     | Clean up:                                        | `docker container prune`
|        |                                                  | `docker volume prune`
|        |                                                  | `docker image prune` |
A journey through Docker Commands (6/6)

Step 18
(Host)
Stop the container (from the host):
docker stop $CID

Step 19
(Host)
See reclaimable space:
docker system df

Step 20
(Host)
Clean up:
docker container prune
docker volume prune
docker image prune

All in one:
docker system prune
Last but not least

Step 21 (Host)

Run a container and wait for it to finish:

CID1=$(docker run -d debian sleep 60)
CID2=$(docker run -d debian sleep 10)
docker wait $CID1 $CID2
Before we go...

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

docker pull python:3.7
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

Diagram:
- eth0
- eth1
- Docker Daemon
- Container xxx
- loopback
Docker default network configuration – **host**

**host**  Host’s network interfaces

![Diagram showing Docker default network configuration]

- Host’s network interfaces: `eth0`, `eth1`, `loopback`
- Docker Daemon connects with `eth1` and `loopback`
- Container `xxx` connects with `eth0` and `loopback`
Docker default network configuration – bridge

**bridge** Virtual switch handled by Docker (default behavior)
Docker networks – all configurations

▶ Kinds of 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
    --subnet 10.0.5.0/24
  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
  - 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
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
    - 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
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: example

- Run an nginx image:
  docker run --rm -it -p 8080:80 nginx
Publish a port: example

- Run an nginx image:
  
  ```bash
  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

![Welcome to nginx!](http://localhost:8080/)

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).
Commercial support is available at [nginx.com](http://nginx.com).

Thank you for using nginx.

**Figure**: nginx is up & running
Docker volumes: command line

- `--volume`: defines a new volume
Docker volumes: command line

- `-v`, `--volume`: defines a new volume

- `docker run -v /host/path:/path ...`
  - Mounts a *bound* volume to `/path`
  - Also support a final :`ro` flag, to bind a read-only volume:
    `docker run -v /host/path:/path:ro ...`
Docker volumes: command line

- **-v, --volume**: defines a new volume

- `docker run -v /host/path:/path ...`
  - Mounts a *bound* volume to `/path`
  - Also support a final `:ro` flag, to bind a read-only volume:
    `docker run -v /host/path:/path:ro ...`

- `docker run -v /path ...`
  - Creates a *data* volume for the `/path` folder
  - Volume will be kept even if the container is deleted
  - It will be visible in `docker volume ls`
  - It can be mounted as a named volume on another container
Docker volumes: example

On the host, in a new folder:

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

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

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

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

- Source files available on:

  http://sed.inrialpes.fr/docker-tuto/index_dockersingularity.html
Docker volumes: example

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

- Docker can be extended with Volume Drivers
- 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 volume create -d nfs --name shared-data
  -o share=nfs-server:/shared/path
```
- Creates a named volume with the NetShare driver
- NetShare accepts `fstab` options as configuration

```
  docker run -v shared-data:/path ...
```
4

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)
Objective:
- Provide a Jupyter notebook within a simple user workspace

Required environment:
- Python 3.7 (because we want to try its latest features)
- Jupyter, to work with notebooks
- A non-root user (karadoc)
Dockerfile: Jupyter notebook service

- **Objective:**
  - Provide a Jupyter notebook within a simple user workspace

- **Required environment:**
  - Python 3.7 (because we want to try its latest features)
  - Jupyter, to work with notebooks
  - A non-root user (karadoc)

- **Dockerfile is available at:**
  
  http://sed.inrialpes.fr/docker-tuto/index_dockersingularity.html
FROM python:3.7

Parent image

Name: Python (official)

Tag: 3.7
Dockerfile: Jupyter notebook service

```sh
FROM python:3.7
LABEL maintainer="SED RA <sed-gra@inria.fr>"
```

Meta information

- Maintainer, version, ...
- Visible in docker inspect
Dockerfile: Jupyter notebook service

FROM python:3.7
LABEL maintainer="SED RA <sed-gra@inria.fr>"

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

Environment variables

- Set for the whole container
- Can’t reference current line
Dockerfile: Jupyter notebook service

FROM python:3.7
LABEL maintainer="SED RA <sed-gra@inria.fr>"

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

# Update the image & install some tools
RUN apt update && apt -y dist-upgrade && \
    pip --no-cache-dir install jupyter

Dependencies setup

- Update the system first
- Install only what’s necessary
- Regroup install commands
- Clean up caches immediately
**Dockerfile: Jupyter notebook service**

```dockerfile
FROM python:3.7
LABEL maintainer="SED RA <sed-gra@inria.fr>"

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

# Update the image & install some tools
RUN apt update && apt -y dist-upgrade && 
    pip --no-cache-dir install jupyter

# Set arguments
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: Jupyter notebook service

FROM python:3.7
LABEL maintainer="SED RA <sed-gra@inria.fr>"

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

# Update the image & install some tools
RUN apt update && apt -y dist-upgrade && 
  pip --no-cache-dir install jupyter

# Set arguments
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
RUN mkdir $home/notebooks
WORKDIR $home/notebooks

Switch to the new user

▶ Only a new USER command can switch back to root
FROM python:3.7
LABEL maintainer="SED RA <sed-gra@inria.fr>"

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

# Update the image & install some tools
RUN apt update && apt -y dist-upgrade && 
pip --no-cache-dir install jupyter

# Set arguments
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
RUN mkdir $home/notebooks
WORKDIR $home/notebooks

# Set the default entry point & arguments
ENTRYPOINT ["jupyter", "notebook", "--no-browser"]
CMD ["--port=8888", "--ip='*'", "--NotebookApp.token=''"]
Dockerfile: Build an image

Step 1  Download the Dockerfile:
http://sed.inrialpes.fr/docker-tuto/docker/Dockerfile
Dockerfile: Build an image

Step 1  Download the Dockerfile:
http://sed.inrialpes.fr/docker-tuto/docker/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/docker/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/docker/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 3 Run it:

docker run --rm -it -p 8888:8888 aubergiste
Launch a browser on host: http://localhost:8888
Dockerfile: Build an image

Step 3 Run it:

docker run --rm -it -p 8888:8888 aubergiste
Launch a browser on host: http://localhost:8888

Step 4 Give it a parameter:

docker run --rm -it aubergiste --help
Dockerfile: Build an image

Step 3  Run it:
        docker run --rm -it -p 8888:8888 aubergiste
        Launch a browser on host: http://localhost:8888

Step 4  Give it a parameter:
        docker run --rm -it aubergiste --help

Step 5  Run a shell instead of a notebook:
        docker run --rm -it --entrypoint /bin/bash aubergiste
Dockerfile: Basic instructions

**Description**

<table>
<thead>
<tr>
<th>FROM</th>
<th>Parent image</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABEL</td>
<td>Metadata to describe the image</td>
</tr>
<tr>
<td>ARG</td>
<td>Variable to be given at build time</td>
</tr>
</tbody>
</table>

**Instructions**

<table>
<thead>
<tr>
<th>ENV</th>
<th>Sets environment variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>Executes shell commands</td>
</tr>
<tr>
<td>SHELL</td>
<td>Sets the shell executing RUN commands</td>
</tr>
<tr>
<td>WORKDIR</td>
<td>Sets the working directory</td>
</tr>
</tbody>
</table>

**Behavior**

<table>
<thead>
<tr>
<th>ENTRYPOINT</th>
<th>Sets the command line to execute ($SHELL by default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td>Sets the default arguments for the entry point</td>
</tr>
</tbody>
</table>
## Dockerfile: More instructions

### Files

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY</td>
<td>Copies/Downloads a file to the image (<a href="#">recommended</a>)</td>
</tr>
<tr>
<td>ADD</td>
<td>Copies/Downloads and auto-decompresses a file</td>
</tr>
<tr>
<td>VOLUME</td>
<td>Declares a folder as a data volume</td>
</tr>
</tbody>
</table>

### Network

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPOSE</td>
<td>Declares ports to expose to other containers</td>
</tr>
</tbody>
</table>

### User management

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>Switches to the given user.</td>
</tr>
<tr>
<td></td>
<td>The user must have been created with <code>useradd</code></td>
</tr>
</tbody>
</table>
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

- Stored in a Docker Registry
  - Either the official Docker Hub (hub.docker.com)
  - Or a private instance of the registry (Nexus plugin, ...)
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, ...)

Thomas Calmant - An Introduction to Docker September 2018 – 45
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
  - User authentication: `docker login`, `docker logout`
- Private registries, running the official registry image
- All registries must provide a **signed** certificate

```
Dockerfile
hub.docker.com
docker push hub.docker.com/name:tag
docker logout
docker build -t name:tag .
docker push myhost/name:tag
```
Setup a Docker registry

Step 1  Download the composition setup at:
http://sed.inrialpes.fr/docker-tuto/index_dockersingularity.html

Step 2  Decompress the file and run the composition from the
extracted folder:
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 (back to the folder with the Dockerfile):

docker build -t aubergiste:1.0 .
Docker image: commands

**Step 4** Build an image (back to the folder with the Dockerfile):

```
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 (back to the folder with the Dockerfile):
        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

**Step 8** Upload it:

docker push localhost/aubergiste

**Step 9** Remove the local reference:

docker rmi aubergiste

**Step 10** Stop the registry composition (from the composition folder):

docker-compose down
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
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 rm aubergiste

Step 10 Stop the registry composition (from the composition folder):
        docker-compose down
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:**
  ```bash
docker commit ${CID} <image>:<tag>
  ```
5
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
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
Principles

version: "3"
services:
nginx:
  image: nginx
  ports:
  - 443:443
  links:
  - registry:registry-srv
  volumes:
  - ./nginx/:/etc/nginx/conf.d

registry:
  image: registry:2
  environment:
    REGISTRY_STORAGE: /data
  volumes:
  - ./data:/data

Docker Daemon
Principles

```yaml
version: "3"
services:
  nginx:
    image: nginx
    ports:
      - 443:443
    links:
      - registry:registry-srv
    volumes:
      - ./nginx/:/etc/nginx/conf.d
  registry:
    image: registry:2
    environment:
      REGISTRY_STORAGE: /data
    volumes:
      - ./data:/data
```

- docker-compose up -d
Principles

- docker-compose up -d

```yaml
version: "3"
services:
  nginx:
    image: nginx
    ports:
      - 443:443
    links:
      - registry:registry-srv
    volumes:
      - ./nginx/:/etc/nginx/conf.d

  registry:
    image: registry:2
    environment:
      REGISTRY_STORAGE: /data
    volumes:
      - ./data:/data
```
Docker Compose Example:

```
docker-compose.yml

version: "3"

services:
  nginx:
    image: nginx
    ports:
      - 443:443
    links:
      - registry:registry-srv
    volumes:
      - ./nginx/:/etc/nginx/conf.d

  registry:
    image: registry:2
    environment:
      REGISTRY_STORAGE: /data
    volumes:
      - ./data:/data
```

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

- docker-compose up -d
- docker-compose stop
- docker-compose down
version: "3"
services:
  nginx:
    image: "nginx"
    ports:
      - "443:443"
    links:
      - registry:registry-srv
    volumes:
      - ./nginx/:/etc/nginx/conf.d
  registry:
    image: "registry:2"
    environment:
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
    volumes:
      - ./data:/data
6

Security

(kind of)
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`
  - Be careful not to overstep a real UID or GID
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

Docker isn't a VIRTUAL MACHINE!

Just look at it for a second, God dammit!

Pfff

You're crazy!

You can do "ONE CONTAINER per PROCESS" or "ONE CONTAINER per APP" or course!

How could you SAY that?!
7
A short introduction to singularity
Before it becomes a standard
What is it?

- HPC-oriented “isolation”
- Based on a single image file to ease transfers
- Code is executed with user’s rights
- Shares by default, constrains by arguments
- Aims to replace Virtual Machines, not Docker
  - Note that Docker and Singularity philosophies are opposite
Shares by default, you said?

- By default, singularity will share a lot from the host:
  - Current environment variables
  - Your home directory
  - Some system directories (/dev, /proc, /tmp, ...)
- This can lead to some tricky situations
  - Process crashing due to an invalid host-inherited environment variable
  - Installation right into your host home directory
    e.g. pip install --user -U setuptools
- Constraint arguments:
  - `-e/--cleanenv` Clean up environment variables
  - `-c/--contain` Use virtual folders (except part of /dev)
    Environment is not cleaned.
  - `-C/--containall` Both -e and -c, plus namespaces isolation
Host sharing/isolation arguments

- **Networking:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Behaviour</th>
<th>Docker equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>Use host network</td>
<td>--net=host</td>
</tr>
<tr>
<td>-n</td>
<td>No network (loopback only)</td>
<td>--net=none</td>
</tr>
</tbody>
</table>

- **Mount points:**
  - `-B /opt`: mount host `/opt` as `/opt` in container
  - `-B /opt:/inner`: mount host `/opt` as `/inner` in container
  - Multiple shares at once: `-B /etc/my-app,/opt:/inner`
Mount points – Home directory

The Home directory is treated with a specific argument:

- `-H $HOME/lower`
  - Mounts $HOME/lower as home folder
  - Path will be the same inside the container
  - Parent hierarchy won’t be mounted.

- `-H $HOME/lower:/home/toto`
  - Mounts $HOME/lower as home folder
  - Makes it appear as /home/toto in the container
## Container recipe

Single file (no default name) separated into multiple sections:

### Header

- **Bootstrap:** Kind of source image  
  (docker, shub, debootstrap, busybox, ...)
- **From:** Name of the source image  
  (content depends on Bootstrap)

### Metadata

- `%help` A help message on how to use the image
- `%labels` Labels to describe/tag the image
## Container recipe

### Content Setup *(executed with root rights)*

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%setup</code></td>
<td>Script executed <strong>on the host</strong></td>
</tr>
<tr>
<td><code>%files</code></td>
<td>List of host files to copy inside the image</td>
</tr>
</tbody>
</table>

### Container setup

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%environment</code></td>
<td>Environment variables in the container</td>
</tr>
<tr>
<td><code>%post</code></td>
<td>Commands executed to construct the image (inside a temporary container)</td>
</tr>
<tr>
<td><code>%runscript</code></td>
<td>Commands executed on singularity run</td>
</tr>
<tr>
<td><code>%test</code></td>
<td>Commands executed at the end of build to check the image</td>
</tr>
</tbody>
</table>
Container recipe – Notebook sample

Bootstrap: docker
From: python:3.7

Parent image
- From a Docker image
- python:3.7 (Docker official image)
Container recipe – Notebook sample

Bootstrap: docker
From: python:3.7

%labels
AUTHOR sed-gra@inria.fr

Meta information

▶ Maintainer, version, ...
▶ Visible in singularity inspect
Container recipe – Notebook sample

Bootstrap: docker
From: python:3.7

%labels
AUTHOR sed-gra@inria.fr

%files
run_jupyter.sh /opt/run_jupyter.sh

Files to copy in the image

► Copies are done before running commands
► Files can be generated on host in the %setup section
Container recipe – Notebook sample

Bootstrap: docker
From: python:3.7

%labels
AUTHOR sed-gra@inria.fr

%files
run_jupyter.sh /opt/run_jupyter.sh

%environment
export LANG=C.UTF-8
export LC_ALL=C.UTF-8

Environment variables

▶ In fact, a shell file sourced at start-up
▶ Don’t forget to EXPORT them
Container recipe – Notebook sample

Bootstrap: docker
From: python:3.7

%labels
AUTHOR sed-gra@inria.fr

%files
run_jupyter.sh /opt/run_jupyter.sh

%environment
export LANG=C.UTF-8
export LC_ALL=C.UTF-8

%post
apt update && apt -y dist-upgrade
pip install jupyter

Commands executed in the image
▶ A shell file executed in a temporary folder
Container recipe – Notebook sample

Bootstrap: docker
From: python:3.7

%labels
AUTHOR sed-gra@inria.fr

%files
run_jupyter.sh /opt/run_jupyter.sh

%environment
export LANG=C.UTF-8
export LC_ALL=C.UTF-8

%post
apt update && apt -y dist-upgrade
pip install jupyter
chmod ugo+x /opt/run_jupyter.sh

%runscript
mkdir -p $HOME/notebooks
/opt/run_jupyter.sh --notebook-dir=$HOME/notebooks --ip="*" --port 8888

Script to be sourced on
▶ singularity run
Container recipe – Apps

- *Apps* are a way to use the same image for multiple pre-defined usages
- Listed with `singularity apps <img>`
- Defined alongside base image sections
- Ran with `singularity run --app <app> <img file>`
  - `singularity run jupyter.img`
  - `singularity run --app console jupyter.img`
  - `singularity run --app qtconsole jupyter.img`
## Application sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%apphelp</td>
<td>Description of the application</td>
</tr>
<tr>
<td>%applabels</td>
<td>Metadata of the application</td>
</tr>
<tr>
<td>%appenv</td>
<td>Environment variables for the application</td>
</tr>
<tr>
<td>%appfiles</td>
<td>Host files to copy inside image</td>
</tr>
<tr>
<td>%appinstall</td>
<td>Commands executed inside the image</td>
</tr>
<tr>
<td>%apprun</td>
<td>Commands executed on <code>run --app &lt;app&gt;</code></td>
</tr>
</tbody>
</table>

- **No %appsetup section**
- Use relative path when copying files for an *app*
- Access it using the `$SCIF_APPROOT` environment variable
Container recipe – App Example

%appfiles console
sample.conf

%appinstall console
pip install readline

%apprun console
echo "Starting in console mode..."
cat $SCIF_APPROOT/sample.conf
jupyter console
Singularity Basic commands

Files available at
http://sed.inrialpes.fr/docker-tuto/index_dockersingularity.html

# Build the image file
sudo singularity build jupyter.img Jupyter.singularity

# Basic
singularity run jupyter.img
# Highly recommended
singularity run -e jupyter.img
# Run a shell in the image
singularity shell -e jupyter.img
# Run an app
singularity run -e --app console jupyter.img
Singularity Container images

- Singularity uses a single file as a container image
- Supported image formats:
  - SquashFS: the current default format
    - Read-only
  - ext3: the previous default format
    - Possible read-write mode
  - sandbox: based on a local directory instead of a single file
    - Writeable
    - Can be seen as a chroot directory
  - .tar, .tar.gz, .tar.bz2: a compressed sandbox
    - Read-only
8

Singularity — Docker

The Persuaders
Most visible differences

<table>
<thead>
<tr>
<th>Singularity</th>
<th>Docker</th>
</tr>
</thead>
<tbody>
<tr>
<td>No daemon (uses SUID)</td>
<td>Unique daemon per host</td>
</tr>
<tr>
<td>Share by default</td>
<td>Constrain by default</td>
</tr>
<tr>
<td>Processes run with user’s rights</td>
<td>Processes run with inner rights</td>
</tr>
<tr>
<td>Sees host with user’s rights</td>
<td>Sees host with root rights</td>
</tr>
<tr>
<td>Single file images</td>
<td>Multi-layer images</td>
</tr>
<tr>
<td>Targets shared computer</td>
<td>Targets service-hosting servers</td>
</tr>
</tbody>
</table>
Work with NVidia GPUs

- Requires the NVIDIA drivers to be installed on the host

- On Docker:
  - Official Open Source plugin from NVIDIA:
    github.com/NVIDIA/nvidia-docker
  - Install the nvidia-docker2 package
  - Run containers with the --runtime=nvidia argument

- On Singularity:
  - Support is included in Singularity (beta)
  - Add the --nv flag when starting the container
Emulate Singularity with Docker

The following command is equivalent to:

```
singularity shell docker://debian
```

docker run \
  -it --rm \
  --pid=host --ipc=host \
  --net=host --uts=host \
  -v /tmp:/tmp \
  -v /etc/passwd:/etc/passwd:ro \
  -v "${HOME}}:${HOME}" -w "${HOME}" \
  --user="$(id -u):$(id -g)" \
  --env-file=<(bash -c set) \
  --entrypoint "/bin/bash" \
  debian
Run Singularity inside Docker

Because... why not?

Dockerfile:

- Debian + Backport repository + singularity-container
- Executed with a new user
- User can do `sudo singularity` without password

Execution:

```
docker run -it --rm --privileged \
    -v $(pwd):/src \
    singularity \
    sudo singularity build /src/out.img /src/Singularity
```
9
Miscellaneous
Singularity Image Registry

- Open Source registry available on GitHub
  https://github.com/singularityhub/sregistry

- Available as a Docker composition:
  1. `git clone` https://github.com/singularityhub/sregistry.git
  2. `cp shub/settings/dummy_secrets.py` to `shub/settings/secrets.py`
  3. Edit `secrets.py` (at least the `SECRET_KEY` variable)
  4. If necessary, edit `shub/settings/config.py`
  5. Run `docker-compose up -d`
  6. Registry is available at `http://localhost`
Containers on ARM

- Both Docker & Singularity have packages for ARM
- Only works with arm images
  - Most are from armhf on the Docker Hub
  - https://hub.docker.com/u/armhf/
- Sample Docker 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 (in multiple versions):
  - microsoft/windowsservercore
  - microsoft/nanoserver (for 64 bits apps only)
- Many images now have a Windows version
  - Python, Node.js, ...
- `docker info`:
  ```
  [...] 
  Server Version: 18.06.1-ce 
  Storage Driver: windowsfilter 
  Default Isolation: hyperv 
  Kernel Version: 10.0 17134 (17134.1.amd64fre.rs4_release.180410-1804) 
  Docker Root Dir: C:\ProgramData\Docker 
  [...] 
  ```
Thanks for your attention

Credits:
▶ CommitStrip
▶ Laurel
▶ xkcd

Thomas Calmant
thomas.calmant@inria.fr
SED/Tyrex
Montbonnot-Saint-Martin
10

Bonus slides

There’s always more
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”
Why not unlocking security?

- docker run -it -d
  --privileged --net=host
  -v /:/host
  -v /dev:/dev -v /run:/run
  -e sysimage=/host
debian

- Inside the container:

  nsenter --mount=/host/proc/1/ns/mnt -- /bin/bash
Some snippets

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
11
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>`

- Node mode switch:
  - `docker node promote <node>`
  - `docker node demote <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:

  ```bash
  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"
  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
```