Introduction to containerization with Singularity

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

1 Warm Up

Check your SSH connection to Inria resources

SSH connection and Slides download

- Choose a Wi-Fi network
  - If you're on INRIA-grenoble or interne-inrialpes, try:
    `ssh yourInriaLogin@access1-cp.inrialpes.fr`
  - If you're on INRIA-guest or eduroam, try:
    `ssh yourInriaLogin@bastion.inrialpes.fr`
- If `ssh` works: you can end your nap
- Otherwise: you need to register your SSH keys
  (see slide 4)
- Link to the slides:
  sed.inrialpes.fr/docker-tuto/pres-20181204.pdf

Generate & register your SSH keys

- Generate ssh keys:
  `ssh-keygen -t rsa`
- Create an issue on the help desk: helpdesk.inria.fr
  - Submit a request to add your new `$HOME/.ssh/id_rsa.pub` file to access the `bastion` server
  - Wait the IT service to handle your request
  - Check again if you can connect to `bastion`
- Add this to your `$HOME/.ssh/config` file
  (create it if necessary):

```
Host access*-cp
    ProxyCommand ssh yourInriaLogin@bastion.inrialpes.fr
↪   "/usr/bin/nc %h %p"
```

Agenda

- The philosophy of Docker & Singularity
- “Safety”
- Playing with Singularity
- Singularity eco-system
- Running Singularity on the cluster

2
Philosophy of Docker & Singularity

Different behaviour for different usage

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 (DevOps-oriented)

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)

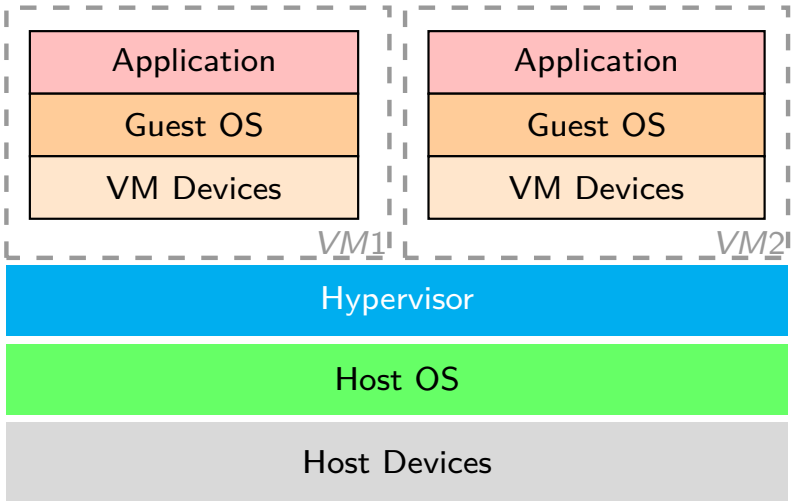
2013/03 Docker (DevOps-oriented)

2015/06 Open Container Initiative (by Docker)

2016/04 Singularity (HPC-oriented)

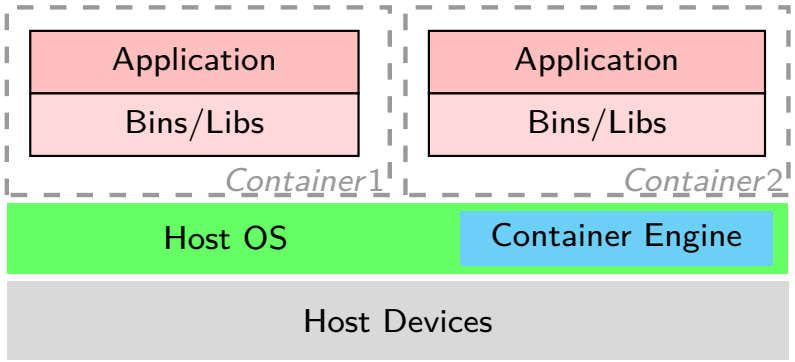
Virtualization vs. Containerization

Type II Virtual Machine



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

Containerization



- 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

Container engines: Docker *vs.* Singularity

Docker

- Long-lived services oriented
- Targets execution on dedicated/managed servers
- Managed by an administrator
- Maximum isolation by default, unlocked by arguments

Singularity

- Wall-timed process oriented
- Targets execution on shared servers
- Containers executed as a user process
- Lowest isolation by default, constrained by arguments

3
Containers and Safety
Define “safe”

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` **group** is `root` on the machine
  - Not suitable on shared resources (like HPC clusters)
  - *User Namespace Remap* can *partially* solve this issue

What Singularity is about

- Singularity executes a process in a pre-defined file system (`chroot`-like)
- Processes are started with **executor's** rights

What Singularity is about

- Singularity executes a process in a pre-defined file system (`chroot`-like)
- Processes are started with **executor's** rights
- Singularity shares **by default**:
  - Folders:
    - Executor's home directory (`$HOME`),
    - Current working directory (`$PWD`),
    - `/tmp`, `/dev`, `/proc`, `/sys`
  - Namespaces:
    - Network, PID, UTS and IPC

What Singularity is about

- Singularity executes a process in a pre-defined file system (`chroot`-like)
- Processes are started with **executor's** rights
- Singularity shares **by default**:
  - Folders:
    - Executor's home directory (`$HOME`),
    - Current working directory (`$PWD`),
    - `/tmp`, `/dev`, `/proc`, `/sys`
  - Namespaces:
    - Network, PID, UTS and IPC
- This can be enforced using arguments (wait for next chapter)
- Executor will never have `root` rights inside a container
  - except if running it with `sudo`
    (**which you'll never do, obviously**)

4
Basic commands
Docker/Singularity equivalences

Warm up

Singularity

- Check if `singularity` 3.0 is installed:
  - `singularity --version`
- Try to run the Singularity "lol cow" image from Docker:
  - `singularity run docker://godlovedc/lolcow`

Docker

- Check if `docker` works:
  - `docker info`
  - `docker run -t godlovedc/lolcow`
- If not:
  - check if the `docker-ce` package is installed
  - add yourself in the **docker** group (and restart your session)

Run a container

- Running a container with the default entry point of an image

Singularity

```
singularity run docker://python:3.7
```

Docker

```
docker run -it python:3.7
```

- In both interpreters, check your user name:

```
>>> import getpass
>>> print("User:", getpass.getuser())
```

Run a shell

- Running `bash` instead of the default entry point of an image

Singularity

```
singularity shell docker://python:3.7
```

Docker

```
docker run -it --entrypoint bash python:3.7
```

- In Docker
  - you don't have any access to the host
- In Singularity
  - you have R/W access to your home directory
  - you are in the host current working directory

Run two processes in the same container – Docker

On a terminal

```
docker run -it python:3.7
```

```
>>> import socket
>>> print("Container ID:", socket.gethostname())
```

On another terminal

```
docker exec -it <container ID> bash
```

Run two processes in the same container – Singularity

Setup

- Singularity run/shell commands doesn't provide this capability
- The container must be prepared as an *instance*:

```
singularity instance start docker://python:3.7 my-instance
```

First process

```
singularity run instance://my-instance
```

Second process

```
singularity exec instance://my-instance bash
```

Stop the instance

```
singularity instance stop my-instance
```

5 Singularity isolation arguments

Before being sorry for losing data

Constraint arguments

Namespace containements

`-i/--ipc`	New IPC namespace (D-Bus, ...)
`-n/--net`	New network namespace (see next slides)
`-p/--pid`	New PID namespace
`--uts`	New UTS (hostname) namespace
`-u/--userns`	New user namespace

Unlocking options

`--allow-setuid`	Allow to run binaries with the *sticky bit*

Constraint arguments

Most used containment options

`--no-home`	Don't mount the user's home directory
`--no-privs`	Drop `root` privileges inside the container
`-e/--cleanenv`	Clear host environment variables
`-c/--contain`	Use virtual folders (except part of `/dev`) Environment is not cleaned.
`-C/--containall`	Namespaces isolation, virtual folders and clean environment

Environment containment

```
export ANSWER=42
singularity exec docker://debian env | grep ANSWER
singularity exec -e docker://debian env | grep ANSWER
```

Networking

- By default, host network namespace is shared
  - Docker equivalent of `--net=host`
- Any configuration or constraint **requires** to execute Singularity as `root`
  - **Note:** don't forget to use `--no-privs`

Network containement arguments

`-n/--net`	Use a new network namespace
`--network`	Kinds of networks to setup in the new namespace
`--network-args`	Network plugins configuration (*port mapping*, ...)
`--dns`	Set the DNS servers inside the container
`--hostname`	Set a custom host name inside the container

Networking

Simple Netcat server

```
singularity run docker://subfuzion/netcat -l 8080
```

Isolated server (new network namespace)

```
sudo singularity run --net \
    docker://subfuzion/netcat -l 8080
```

Mapped server (Host 80 → Container 8080)

```
sudo singularity run --no-privs --net \
    --network-args "portmap=80:8080/tcp" \
    docker://subfuzion/netcat -l 8080
```

Test each with: `nc localhost 8080` (80 for the latest one)

Mount points

- Access to host files/folders with executor's rights
- 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`
- No equivalent to *Docker named volumes*

Mount points – Home directory

The Home directory is treated with a specific argument:

- `-H $HOME/lower`
  - Mounts `$HOME/lower` as home directory
  - Path will be the same inside the container
  - Parent hierarchy won't be mounted.
- `-H $HOME/lower:/home/toto`
  - Mounts `$HOME/lower` as home directory inside the container
  - Makes it appear as `/home/toto` inside the container
- `--no-home`
  - Doesn't mount the user's home directory
  - User will only be able to write on `/tmp` and explicit mounts

Mount points

Docker

```
docker run -it -v /scratch:/scratch \
    -v /data:/host-data debian
```

Singularity

```
singularity run \
    -B /scratch,/data:/host-data \
    docker://debian
```

Or, closer to Docker behaviour:

```
singularity run \
    -B /scratch,/data:/host-data -C \
    docker://debian
```

Work with NVidia GPUs

Requirements

NVIDIA drivers must be installed on the host

Docker

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

Singularity

- Support is included in Singularity (*beta*)
- Add the `--nv` flag when starting the container

6
Singularity images

Running Docker images with Singularity

Run an image from Docker

- From a Docker registry:

  ```
  singularity run docker://debian
  ```

- From a local Docker image

  Note: tag is **mandatory**:

  ```
  singularity run docker-daemon:my-local-image:latest
  ```

- From a `docker save .tar` file:

  ```
  docker save subfuzion/netcat > netcat.tar
  singularity run docker-archive:./netcat.tar
  ```

Running Docker images with Singularity

Convert a Docker image to Singularity

- From a Docker registry:

```
singularity build netcat.sif docker://subfuzion/netcat
```

- From a local Docker image

  Note: tag is **still mandatory**:

```
singularity build netcat.sif \
    docker-daemon:subfuzion/netcat:latest
```

- From a `docker save .tar` file:

```
docker save subfuzion/netcat > netcat.tar
singularity build netcat.sif \
    docker-archive:./netcat.tar
```

Singularity recipe *vs.* `Dockerfile`: Main differences

Docker

- Dockerfile containing instructions/commands
- Each instruction is executed in a temporary container
  (each making a new image layer)

Singularity

- Recipe divided into *sections*
  (most of them becoming script files in the image)
- Commands are executed both
  - on host with `root` rights
  - in an isolated directory (will become the image)

Singularity recipe

Header

`Bootstrap:`	Kind of source image (`docker`, `shub`, `debootstrap`, `busybox`, `yum`, ...)
`From:`	Name of the source image (value depends on `Bootstrap`)

Metadata

`%help`	A help message on how to use the image (shown with `singularity run-help`)
`%labels`	Labels to describe/tag the image (shown with `singularity inspect`)

Quick overview

Docker

```
FROM alpine
LABEL maintainer="Tony Pujals <tony.pujals@amazon.com>"
RUN apk add --no-cache netcat-openbsd
ENTRYPOINT [ "nc" ]
CMD [ "-l", "8080" ]
```

Singularity

```
Bootstrap: library
From: alpine
%labels
AUTHOR="Tony Pujals <tony.pujals@amazon.com>"
%post
    apk add --no-cache netcat-openbsd
%runscript
    if [ -z "$*" ]; then nc -l 8080
    else nc $*; fi
```

Singularity recipe

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

`%setup`	Script executed **on the host**
`%files`	List of host files to copy inside the image

Container setup

`%post`	Commands executed to construct the image (inside a temporary container)
`%environment`	Environment variables in the container (will override user environment) (not available during %post)
`%runscript`	Commands executed on `singularity run`
`%test`	Commands executed at the end of `build` to check the image

Sample recipe: TensorFlow – Metadata

```
Bootstrap: docker
From: python:3.6

%labels
AUTHOR sed-gra@inria.fr
```

Sample recipe: TensorFlow – Files

```
%setup
mkdir -p ${SINGULARITY_ROOTFS}/opt/scripts
mkdir -p ${SINGULARITY_ROOTFS}/opt/datasets/fashion-mnist

%files
basic_classification.py     /opt/scripts/
train-labels-idx1-ubyte.gz  /opt/datasets/fashion-mnist
train-images-idx3-ubyte.gz  /opt/datasets/fashion-mnist
t10k-labels-idx1-ubyte.gz   /opt/datasets/fashion-mnist
t10k-images-idx3-ubyte.gz   /opt/datasets/fashion-mnist
```

Sample recipe: TensorFlow – Installation

```
%environment
export LANG=C.UTF-8 LC_ALL=C.UTF-8
export PYTHONPATH=/opt/scripts

%post
pip install tensorflow  # CPU only
pip install keras jupyter notebook matplotlib
chmod a+x /opt/scripts/basic_classification.py

%runscript
python3 /opt/scripts/basic_classification.py
```

Sample recipe: TensorFlow

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

Compilation

```
sudo singularity build tensorflow.sif
↪  tensorflow.singularity
```

Execution

```
singularity run tensorflow.sif
singularity shell tensorflow.sif
```

Note on compilation

- If you have a no space left in /tmp:
  `mkdir /scratch/tmp ; export TMPDIR=/scratch/tmp`

Singularity 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 recipe – Apps

Application sections

`%apphelp`	Description of the application
`%applabels`	Metadata of the application
`%appenv`	Environment variables for the application
`%appfiles`	Host files to copy inside image
`%appinstall`	Commands executed inside the image
`%apprun`	Commands executed on `run --app <app>`

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

Sample recipe: TensorFlow with apps

```
# ...

%runscript
python3 /opt/scripts/basic_classification.py

%apprun console
jupyter console

%apprun notebook
jupyter notebook --ip="127.0.0.1" --NotebookApp.token='' --no-browser
```

Sample recipe: TensorFlow with apps

Run apps from an image

- `singularity run tensorflow.sif`
- `singularity run --app console tensorflow.sif`
- `singularity run --app notebook tensorflow.sif`

Sample usage: Mount points

1. Download the *Movie Script Keywords Classification* files (3 files) from sed.inrialpes.fr/docker-tuto/
2. Put them in a local folder
3. Run singularity, mounting dataset files in the following folders
4. Execute `text_classification.py` from your home directory

```
           imdb.npz   ⇒   /opt/datasets/imdb.npz
imdb_word_index.json  ⇒   /opt/datasets/imdb_word_index.json
```

Run the script in the image context

```
singularity exec -B imdb.npz:/opt/datasets/imdb.npz,... \
    tensorflow.sif \
    $HOME/text_classification.py
```

7
Singularity eco-system
Singularity Hub & private registry

Docker-oriented Image registries

Docker Hub

- Public repository: hub.docker.com
- **Note:** you must **always** look at the Dockerfile when choosing an image

Private local registry

- Private instance of the official `registry` image

GitLab at INRIA

- Activate the (black box) *registry* feature on your projects
- Access depends on the project visibility

Singularity-oriented Image registries

Singularity Hub

- Public repository: www.singularity-hub.org
- **Note:** you must **always** look at the recipe when choosing an image

Singularity Library

- Public repository: cloud.sylabs.io/library
- **Note:** you must **always** look at the recipe when choosing an image

Private Singularity registry

- OpenSource registry: singularityhub.github.io/sregistry/
- To be executed with `docker-compose`

Analyze a SIF image

How to deal with an unknown SIF file

- If the source is really weird
  - Delete the file.
- If the source is kind of trustworthy:
  - Run it in `shell` mode:
    `singularity shell weird.sif`
  - Find the content of the recipe:
    `cat /.singularity.d/Singularity`

Important

- SIF files are as safe as Docker images (read-only, signed...)
- Other formats (ext3, SquashFS...) can be dangerous:
  - Many tools exist to edit those images
  - Note that `run`, `shell` and `exec` are in fact scripts in the image

8
Inria Cluster and Singularity

Inria Cluster and Singularity

HPC Clusters at Inria Rhône-Alpes

- Click to access : HPC Clusters service presentation

How to test Tensorflow singularity image on Inria Clusters

- Image on your computer, using `scp`:

```
scp ./yourimage.sif
 ↪  access1-cp:/services/scratch/yourteam/yourname/
```

- Image on your home directory, from front-end:

```
ssh access[1|2]-cp
# ...
cp ~/yourimage.sif
 ↪  /services/scratch/yourteam/yourname/
```

Submit job for CPU nodes

- Interactive mode:

```
oarsub -I -l "cpu=1,walltime=00:30:00" \
  -p "cluster='mistis|SIC|...'"
```

- Batch mode:

```
oarsub -l "cpu=1,walltime=00:30:00" \
  -p "cluster='mistis|SIC|...'" \
  singularity run
↪  /services/scratch/yourteam/yourname/yourimage.sif
```

Submit job for GPU nodes

- Interactive mode

```
oarsub -I \
  -l "{gpu='YES'}/host=1/gpudevice=1,walltime=00:30:00" \
  -p "cluster='mistis|kinovis|.."
```

- Batch mode

```
oarsub \
  -l "{gpu='YES'}/host=1/gpudevice=1,walltime=00:30:00" \
  -p "cluster='mistis|kinovis|..." \
  singularity run --nv
↪  /services/scratch/yourteam/yourname/yourimage.sif
```

When reusing the image multiple times

Sometimes, you will use the same image multiple time, either:

- restarting the same (random-based) code multiple times
- running different *apps* sequentially

It is better in that case to copy the image file to the local scratch, to speed up access to the image content.

- On the OAR node:

```
mkdir -p /local_scratch/data/yourteam/yourname

cp /services/scratch/yourteam/yourname/yourimage.sif
 ↪  /local_scratch/data/yourteam/yourname

singularity run
 ↪  /local_scratch/data/yourteam/yourname/yourimage.sif
```

Muti-hosts, multi-calls (1/2)

```
#!/bin/bash
# Compute OAR hosts
hosts=($(sort -u $OAR_RESOURCE_FILE))
master=${hosts[0]}
slaves=${hosts[@]:1}

# Copy files
for host in ${hosts[@]}
do
    oarsh $host /bin/bash <<EOF&
mkdir -p /local_scratch/data/yourteam/yourname
cp /services/scratch/yourteam/yourname/yourimage.sif
 ↪   /local_scratch/data/yourteam/yourname
EOF
done
# Wait for the copies to end
wait
```

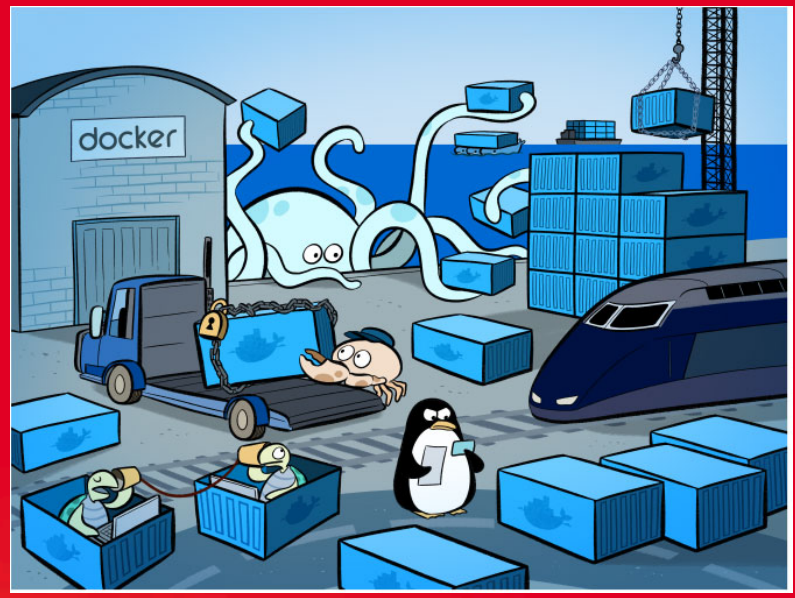
Muti-hosts, multi-calls (2/2)

```
# Run master script
oarsh -n $master "singularity run -B /services --app master
 ↪   /local_scratch/data/yourteam/yourname/yourimage.sif"
# Run slave scripts
for slave in ${slaves[@]}
do
    oarsh -n $slave "singularity run -B /services --app slave
 ↪   /local_scratch/data/yourteam/yourname/yourimage.sif $master" &
done
wait
for host in ${slaves[@]}
do
  oarsh -n $slave "singularity run -B /services --app consolidate
 ↪   /local_scratch/data/yourteam/yourname/yourimage.sif" &
done
wait
# Clean up
for host in ${hosts[@]}
do
    oarsh -n $host "rm
 ↪   /local_scratch/data/yourteam/yourname/yourimage.sif"
done
```

Thanks for your attention

Credits:

- Soraya Arias
- Jean-François Scariot
- Vincent Blanc

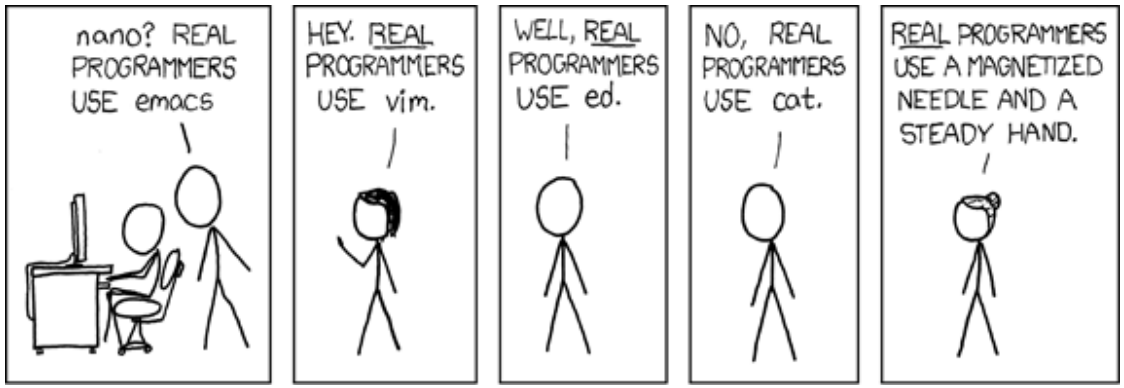


Thomas Calmant
thomas.calmant@inria.fr
SED/Tyrex
Montbonnot-Saint-Martin

9
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"



Docker configuration: 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

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 ...
  ```

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
[...]
```