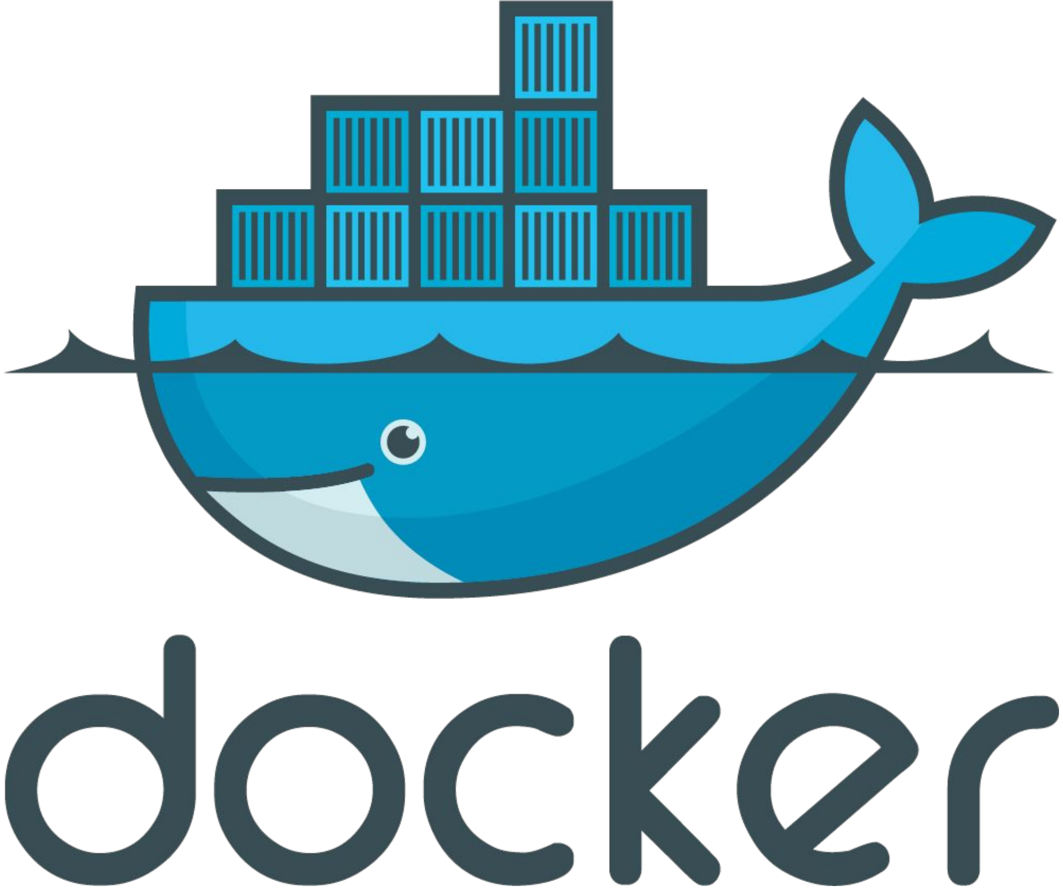


Introduction to Containers

By: Nitish Jadia

Outline

- Container recipe
- Why should I care?
- A quick docker demo
- Building blocks
- Security

Outline

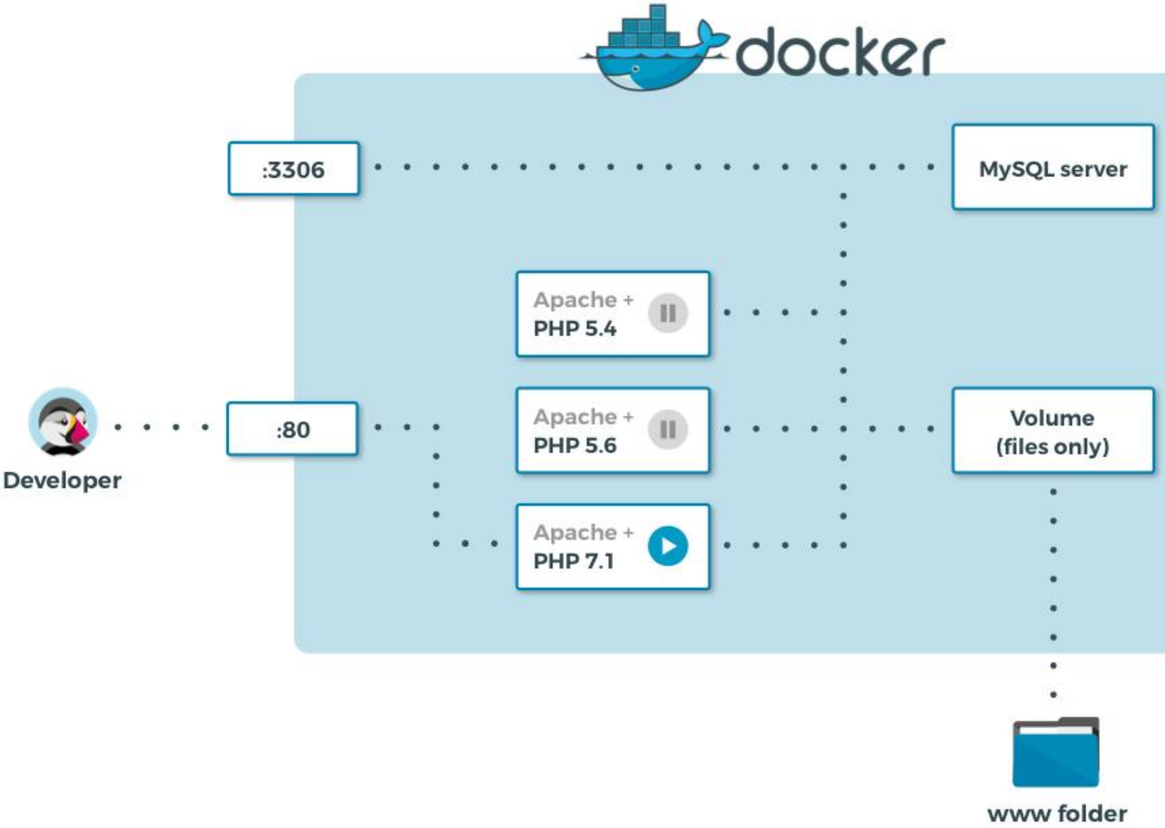
- **Container recipe**
- Why should I care?
- A quick docker demo
- Building blocks
- Security

Containers? These?

source: https://medium.com

Before containers ...things were messy

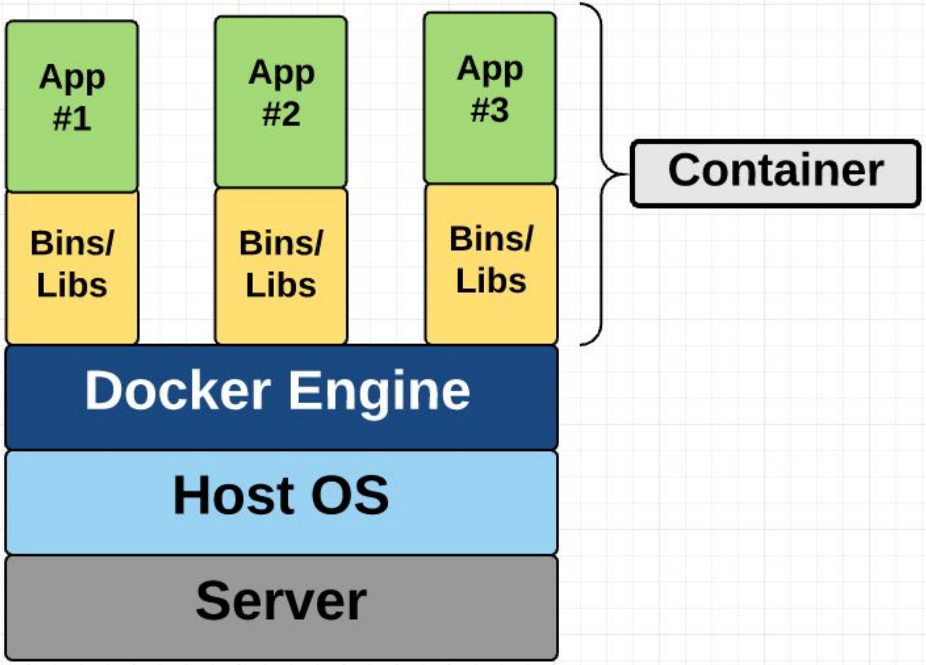
source: https://en.wikipedia.org/

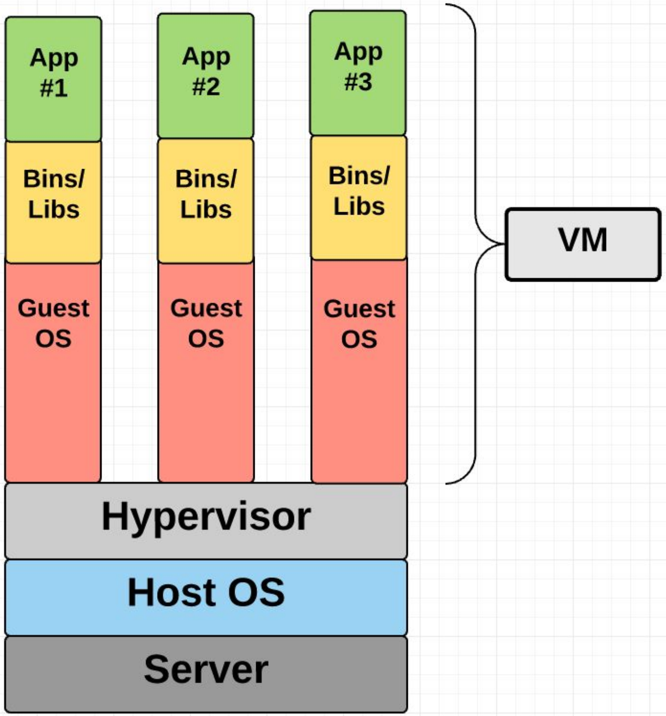


source: http://build.prestashop.com/assets/images/2017/02/prestashop-docker.jpg

Organise your server with containers

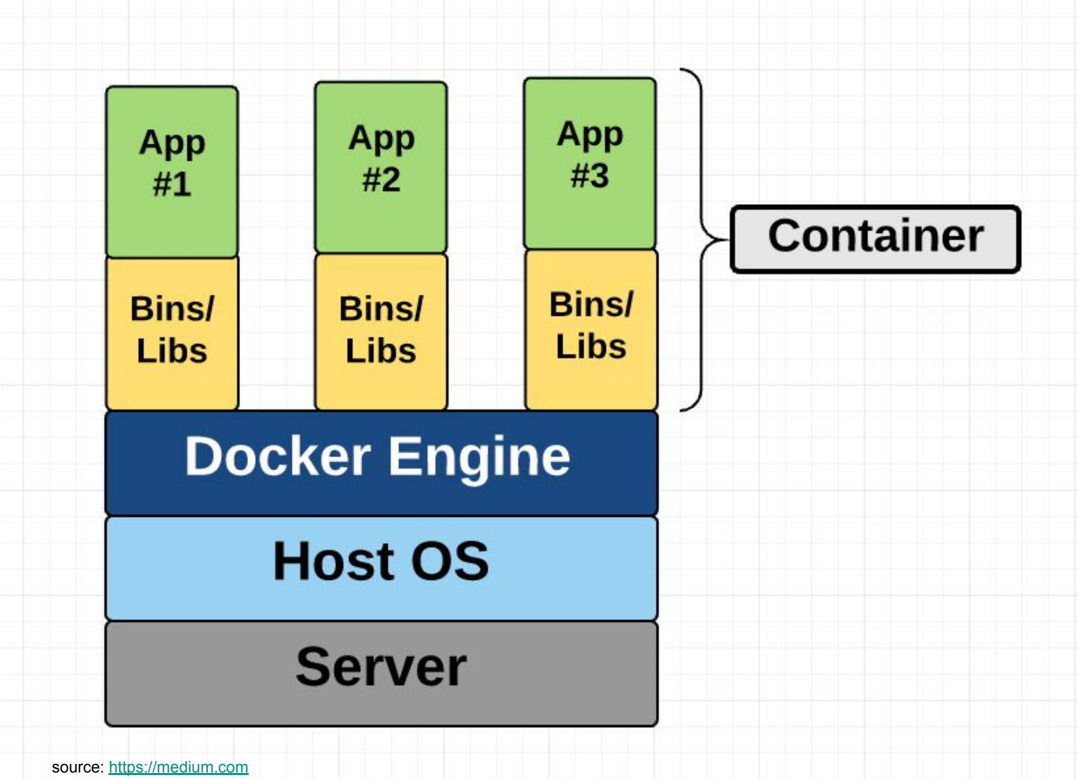
Containers





source: https://medium.com

Containers



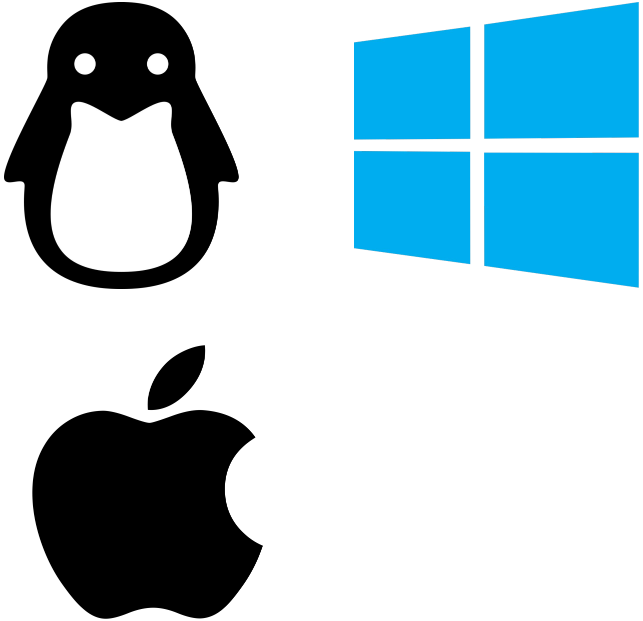
source: https://medium.com

- All containers share the same kernel of the host system.
  Pro: Extremely reduced performance overhead.
- Better utilization of resources due to shared kernel.
- Lightweight and uses less space on disk.
- Portable and better dependency management

Outline

- Container recipe
- **Why should I care?**
- A quick docker demo
- Building blocks
- Security

Deploy anywhere and anything



- webapps
- backends
- SQL, NoSQL
- big data
- load balancing
- … and more

… but, it was working on my machine.

Deploy reliably & consistently

- If it works locally, it will work on the server
- With exactly the same behavior
- Regardless of versions
- Regardless of distros
- Regardless of dependencies
- Typical laptop runs 10-100 containers easily
- Typical server can run 100-1000 containers

Outline

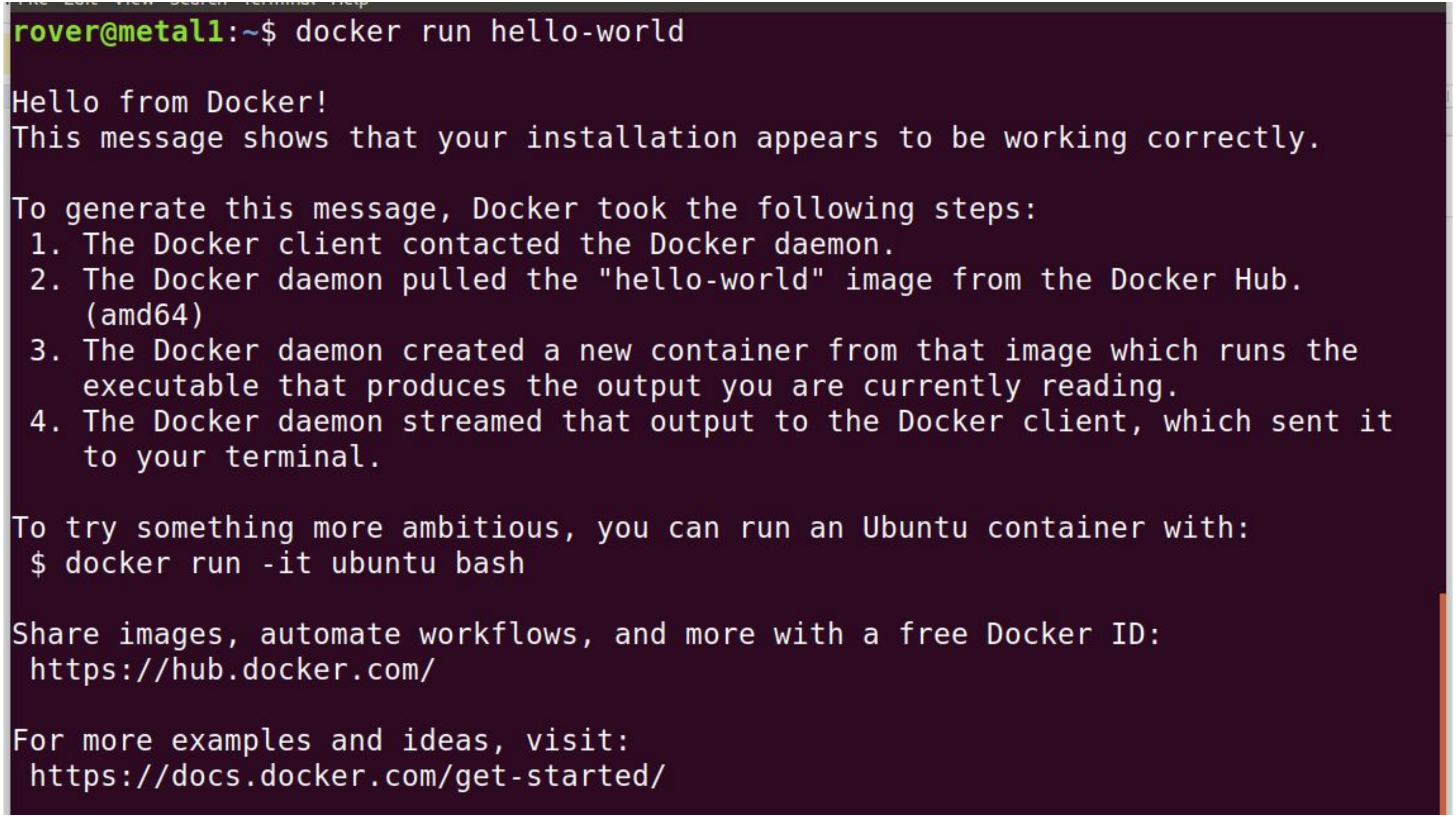
- Container recipe
- Why should I care?
- **A quick docker demo**
- Building blocks
- Security

Demo

- Docker installation
- Docker CLi
  - Basic commands
  - Pull images
  - Deploy containers
- Docker hub

Is docker running?

```
docker run hello-world
```

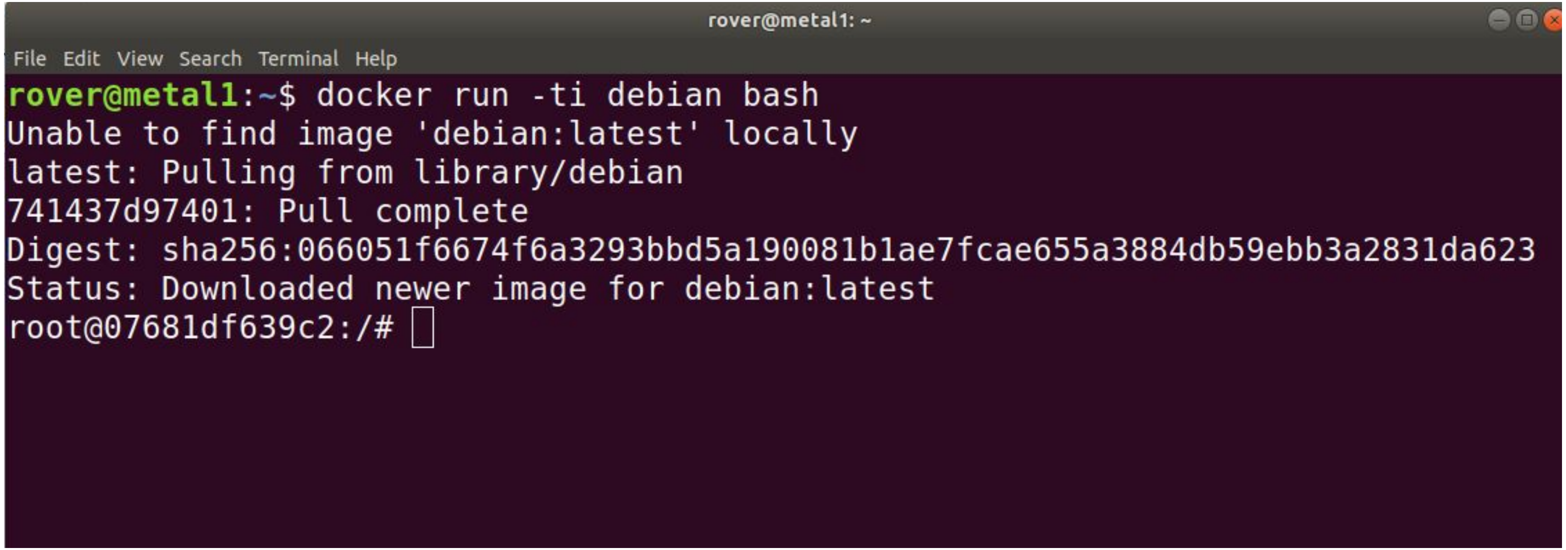


```
rover@metal1:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

docker run

```
docker run -ti debian bash
```

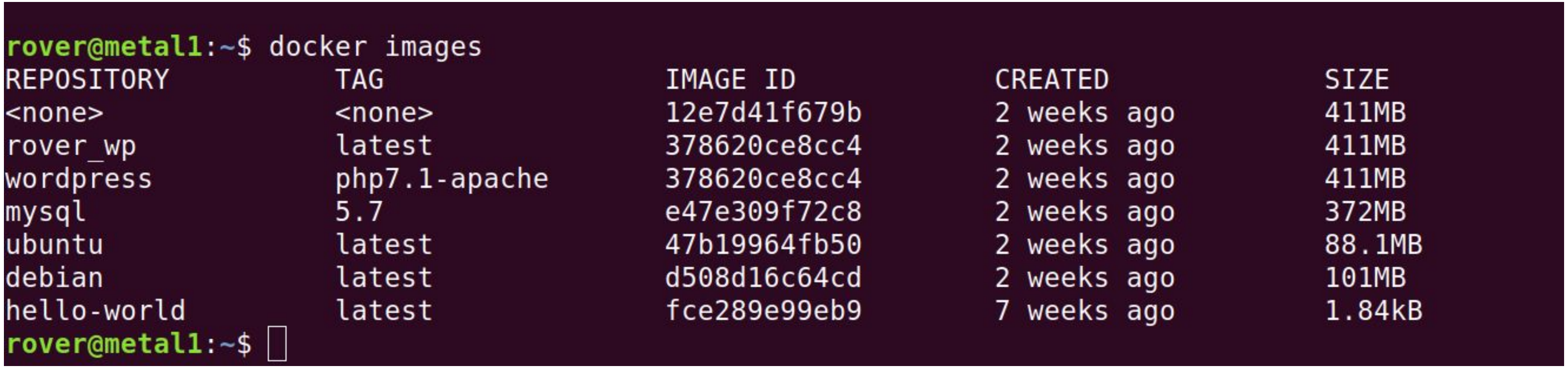
-ti -> terminal interactive



```
rover@metal1:~$ docker run -ti debian bash
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
741437d97401: Pull complete
Digest: sha256:066051f6674f6a3293bbd5a190081b1ae7fcae655a3884db59ebb3a2831da623
Status: Downloaded newer image for debian:latest
root@07681df639c2:/# 
```

Check list of images

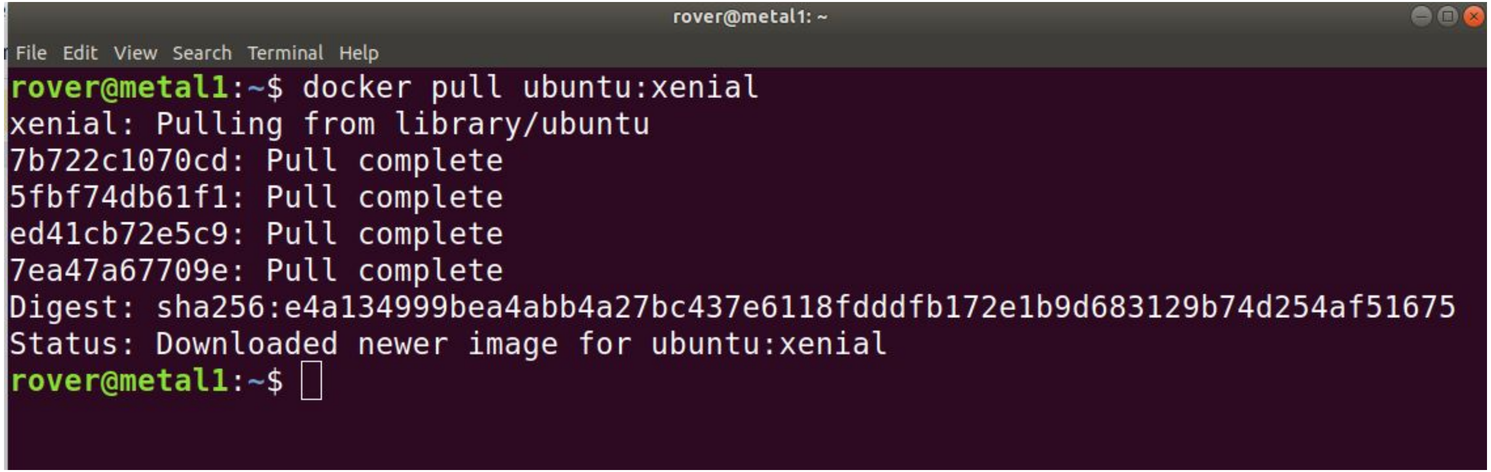
```
docker images
```



```
rover@metal1:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              12e7d41f679b        2 weeks ago         411MB
rover_wp            latest              378620ce8cc4        2 weeks ago         411MB
wordpress           php7.1-apache       378620ce8cc4        2 weeks ago         411MB
mysql               5.7                 e47e309f72c8        2 weeks ago         372MB
ubuntu              latest              47b19964fb50        2 weeks ago         88.1MB
debian              latest              d508d16c64cd        2 weeks ago         101MB
hello-world         latest              fce289e99eb9        7 weeks ago         1.84kB
rover@metal1:~$ 
```

Pulling images

```
docker pull ubuntu:xenial
```



```
rover@metal1:~$ docker pull ubuntu:xenial
xenial: Pulling from library/ubuntu
7b722c1070cd: Pull complete
5fbf74db61f1: Pull complete
ed41cb72e5c9: Pull complete
7ea47a67709e: Pull complete
Digest: sha256:e4a134999bea4abb4a27bc437e6118fdddfb172e1b9d683129b74d254af51675
Status: Downloaded newer image for ubuntu:xenial
rover@metal1:~$ 
```

Terminology

Images - The blueprints of our application which form the basis of containers. In the demo above, we used the `docker pull` command to download the busybox or ubuntu image.

Containers - Created from Docker images and run the actual application. We create a container using `docker run` which we did using the busybox image that we downloaded. A list of running containers can be seen using the `docker ps` command.

Docker Daemon - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system to which clients talk to.

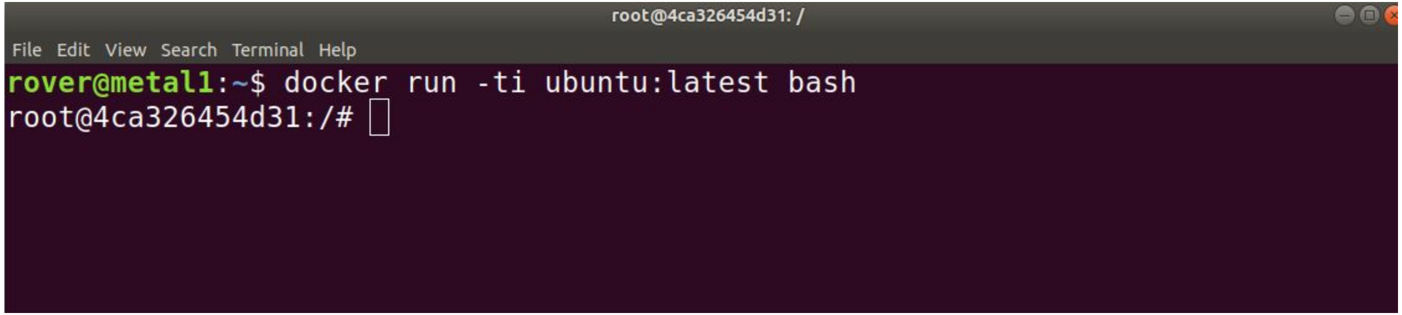
Docker Client - The command line tool that allows the user to interact with the daemon. More generally, there can be other forms of clients too - such as Kitematic which provide a GUI to the users.

Docker Hub - A registry of Docker images. You can think of the registry as a directory of all available Docker images. If required, one can host their own Docker registries and can use them for pulling images.

Run docker container from images

```
docker run -ti ubuntu:latest bash
```

ubuntu -> image
latest -> tag (optional, by default it's latest)
bash -> what do we want to do with the image.

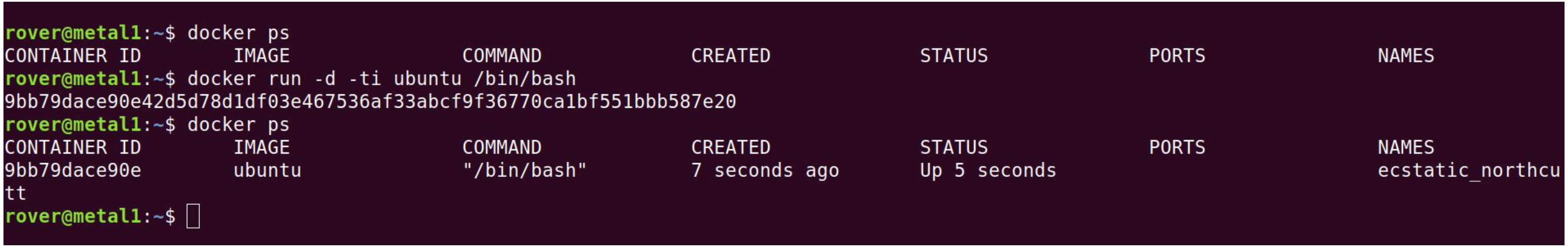


```
rover@metal1:~$ docker run -ti ubuntu:latest bash
root@4ca326454d31:/# 
```

Leave container running in background(detatch)

```
docker run -d -ti ubuntu /bin/bash
```

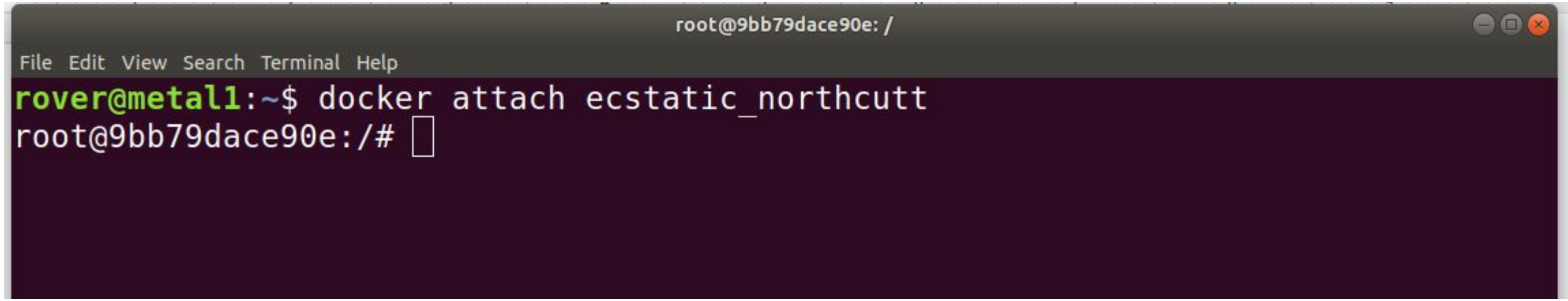
-d -> detaches the container



```
rover@metal1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
rover@metal1:~$ docker run -d -ti ubuntu /bin/bash
9bb79dace90e42d5d78d1df03e467536af33abcf9f36770ca1bf551bbb587e20
rover@metal1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
9bb79dace90e        ubuntu              "/bin/bash"         7 seconds ago       Up 5 seconds                            ecstatic_northcu
tt
rover@metal1:~$ 
```

Attach to a container

```
docker attach ecstatic_northcutt
```



```
rover@metal1:~$ docker attach ecstatic_northcutt
root@9bb79dace90e:/# 
```

Additional commands

- `docker ps -a`
- `docker info`
- `docker restart zeolous_darwin`
- `docker inspect blisful_saha`
- `docker inspect blisful_saha | grep -i ip`

Dockerfile

- A Dockerfile is a simple text-file that contains a list of commands that the Docker client calls while creating an image.
- It's a simple way to automate the image creation process.

Let's create a Dockerfile:

```
mkdir build

cd build

vim Dockerfile
```

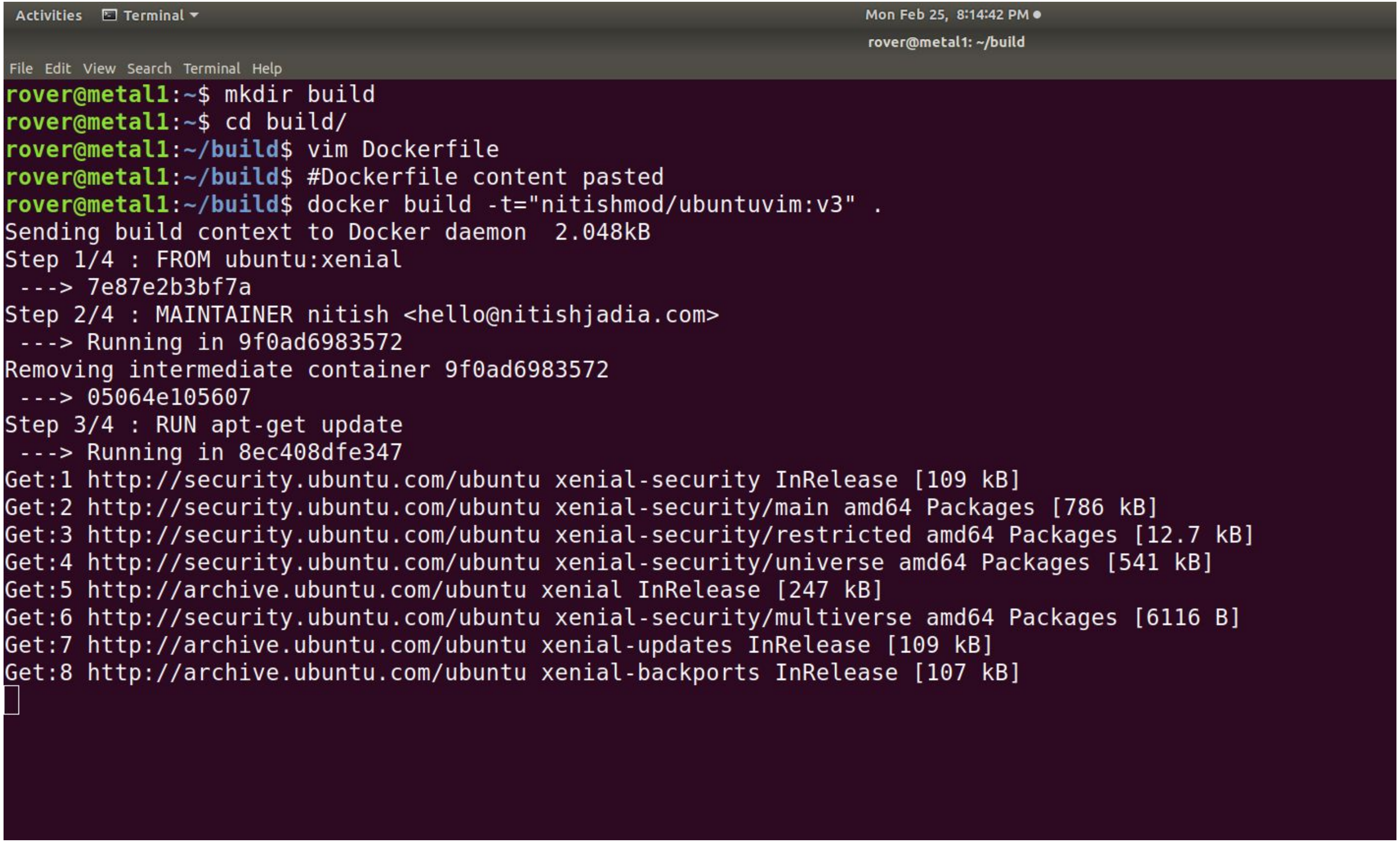
Dockerfile

```
#This is a custom ubuntu image with vim already installed

FROM ubuntu:xenial

MAINTAINER nitish <hello@nitishjadia.com>

RUN apt-get update

RUN apt-get install -y vim
```

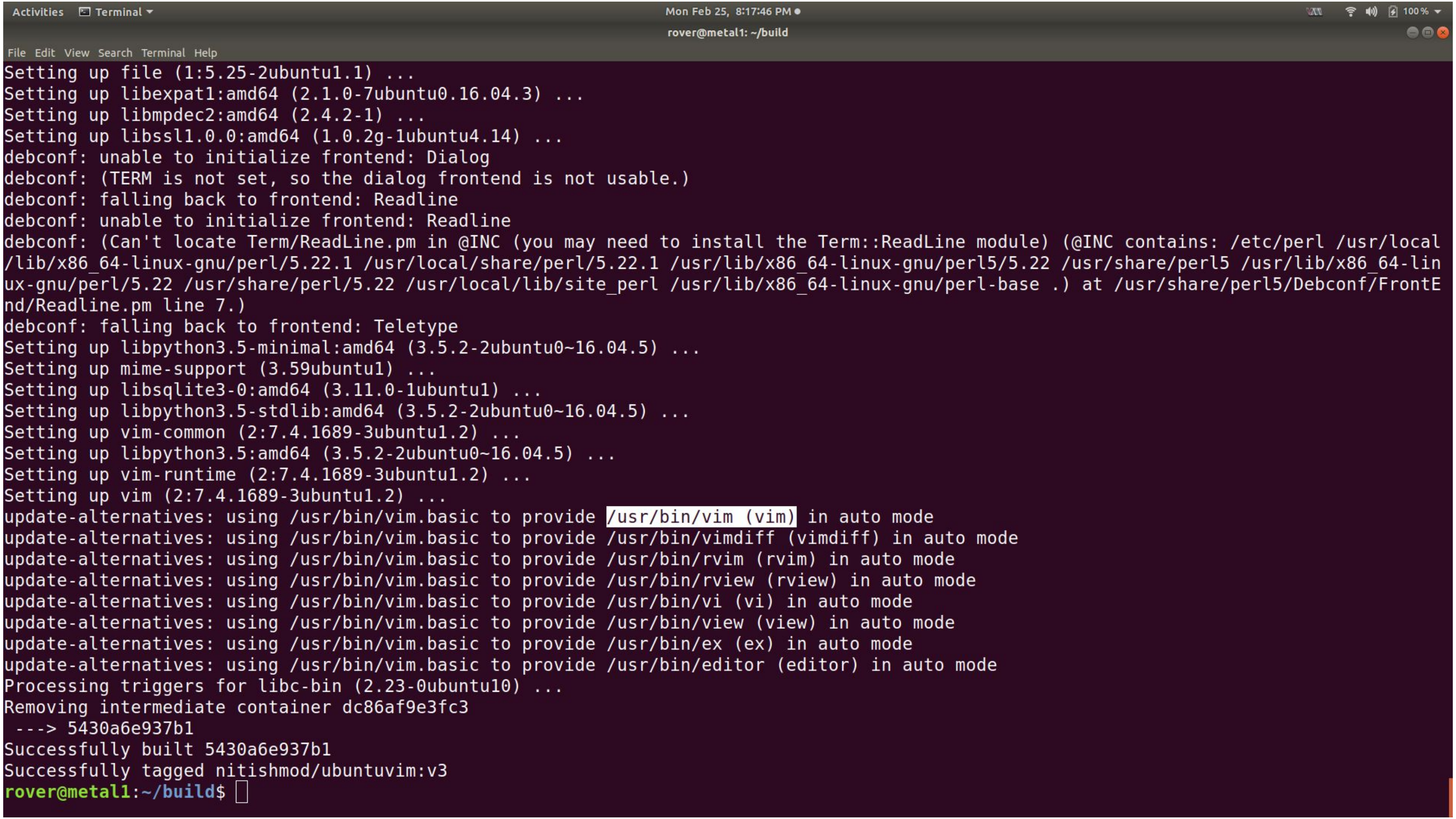
Dockerfile

```
docker build -t="nitishmod/ubuntuvim:v3" .
```

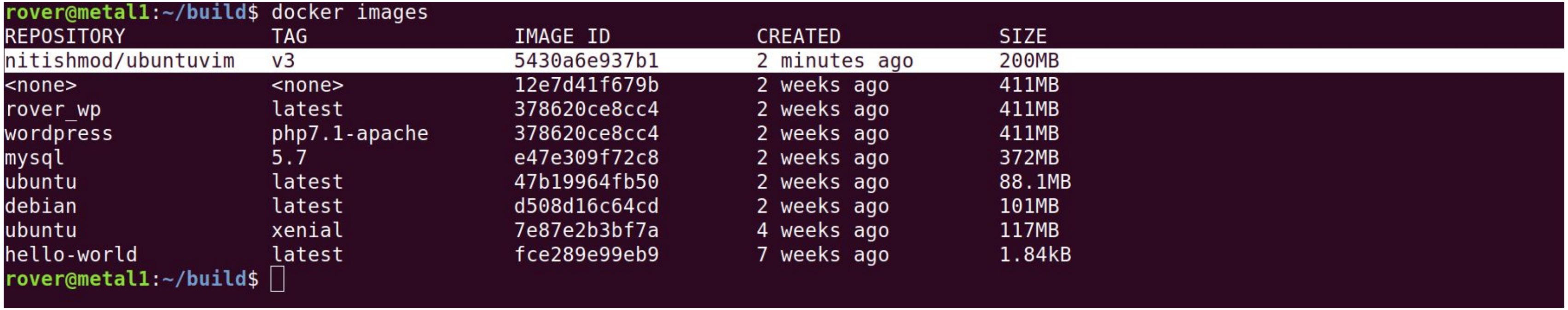
-t -> title
. -> dot, because Dockerfile is in the same folder.



```
rover@metal1:~$ mkdir build
rover@metal1:~$ cd build/
rover@metal1:~/build$ vim Dockerfile
rover@metal1:~/build$ #Dockerfile content pasted
rover@metal1:~/build$ docker build -t="nitishmod/ubuntuvim:v3" .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu:xenial
 ---> 7e87e2b3bf7a
Step 2/4 : MAINTAINER nitish <hello@nitishjadia.com>
 ---> Running in 9f0ad6983572
Removing intermediate container 9f0ad6983572
 ---> 05064e105607
Step 3/4 : RUN apt-get update
 ---> Running in 8ec408dfe347
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [786 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.7 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [541 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [6116 B]
Get:7 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:8 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
```



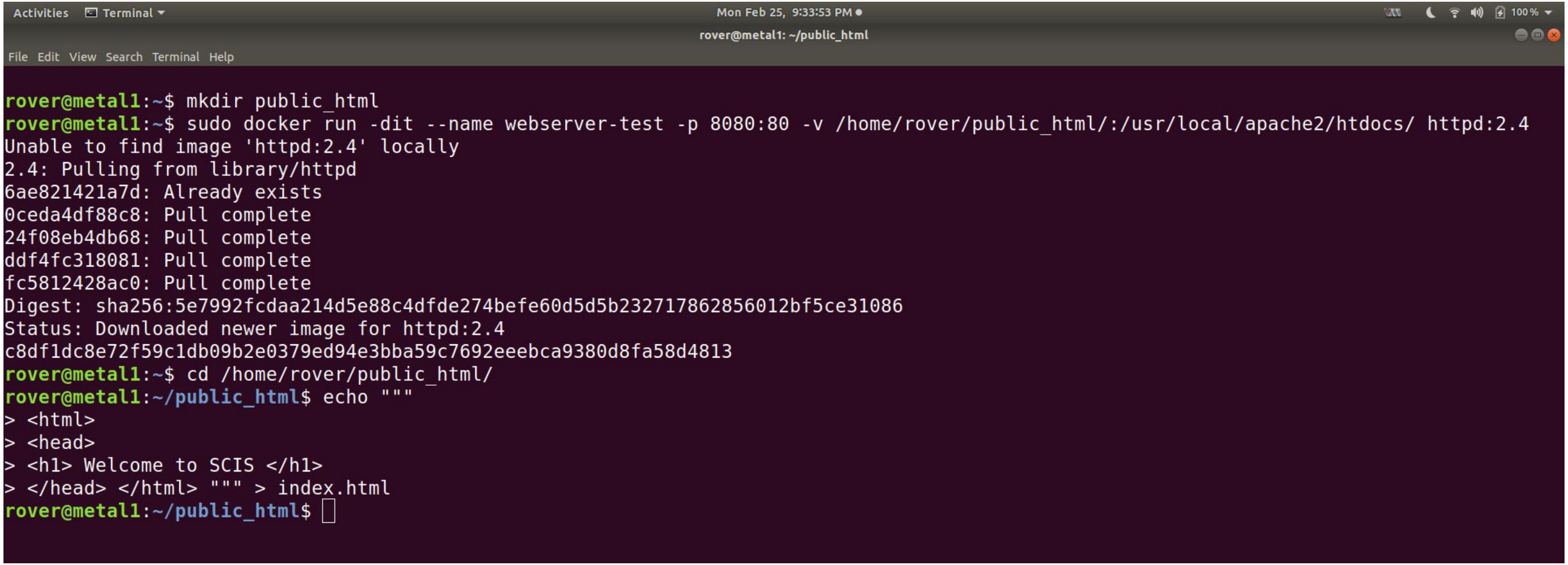
```
Setting up file (1:5.25-2ubuntu1.1) ...
Setting up libexpat1:amd64 (2.1.0-7ubuntu0.16.04.3) ...
Setting up libmpdec2:amd64 (2.4.2-1) ...
Setting up libssl1.0.0:amd64 (1.0.2g-1ubuntu4.14) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC contains: /etc/perl /usr/local
/lib/x86_64-linux-gnu/perl/5.22.1 /usr/local/share/perl/5.22.1 /usr/lib/x86_64-linux-gnu/perl5/5.22 /usr/share/perl5 /usr/lib/x86_64-lin
ux-gnu/perl/5.22 /usr/share/perl/5.22 /usr/local/lib/site_perl /usr/lib/x86_64-linux-gnu/perl-base .) at /usr/share/perl5/Debconf/FrontE
nd/Readline.pm line 7.)
debconf: falling back to frontend: Teletype
Setting up libpython3.5-minimal:amd64 (3.5.2-2ubuntu0~16.04.5) ...
Setting up mime-support (3.59ubuntu1) ...
Setting up libsqlite3-0:amd64 (3.11.0-1ubuntu1) ...
Setting up libpython3.5-stdlib:amd64 (3.5.2-2ubuntu0~16.04.5) ...
Setting up vim-common (2:7.4.1689-3ubuntu1.2) ...
Setting up libpython3.5:amd64 (3.5.2-2ubuntu0~16.04.5) ...
Setting up vim-runtime (2:7.4.1689-3ubuntu1.2) ...
Setting up vim (2:7.4.1689-3ubuntu1.2) ...
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vim (vim) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vimdiff (vimdiff) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rvim (rvim) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rview (rview) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vi (vi) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/view (view) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/ex (ex) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/editor (editor) in auto mode
Processing triggers for libc-bin (2.23-0ubuntu10) ...
Removing intermediate container dc86af9e3fc3
 ---> 5430a6e937b1
Successfully built 5430a6e937b1
Successfully tagged nitishmod/ubuntuvim:v3
rover@metal1:~/build$ 
```



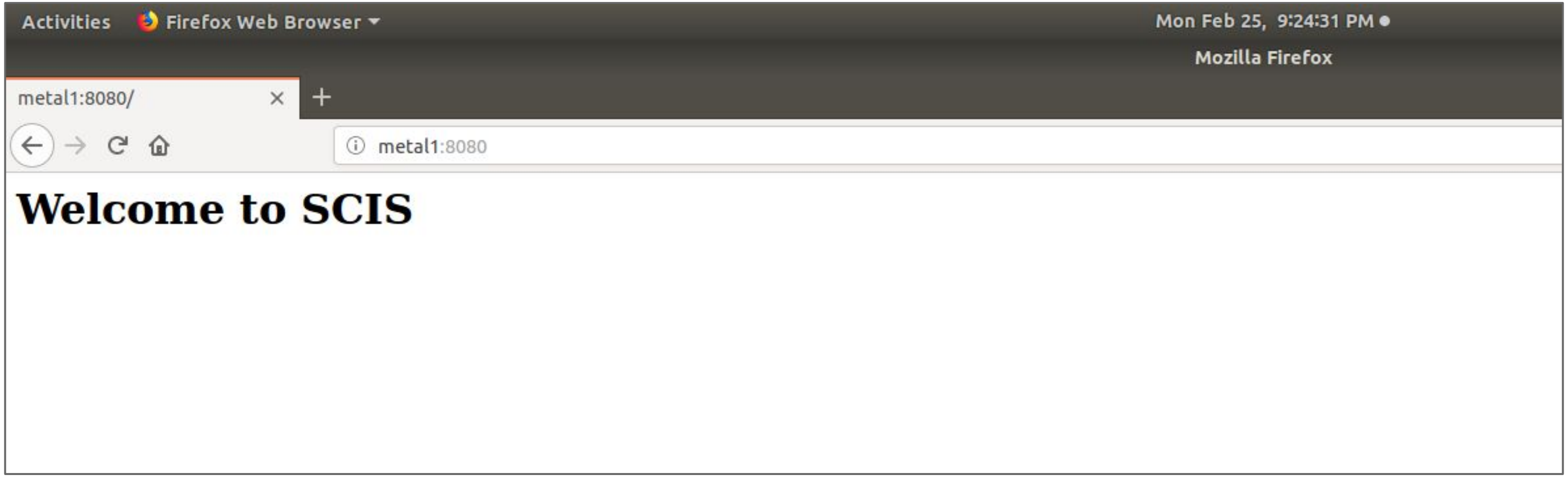
```
rover@metal1:~/build$ docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
nitishmod/ubuntuvim   v3                  5430a6e937b1        2 minutes ago       200MB
<none>                <none>              12e7d41f679b        2 weeks ago         411MB
rover_wp              latest              378620ce8cc4        2 weeks ago         411MB
wordpress             php7.1-apache       378620ce8cc4        2 weeks ago         411MB
mysql                 5.7                 e47e309f72c8        2 weeks ago         372MB
ubuntu                latest              47b19964fb50        2 weeks ago         88.1MB
debian                latest              d508d16c64cd        2 weeks ago         101MB
ubuntu                xenial              7e87e2b3bf7a        4 weeks ago         117MB
hello-world           latest              fce289e99eb9        7 weeks ago         1.84kB
rover@metal1:~/build$
```

Apache web server

```
> mkdir /home/rover/public_html

> sudo docker run -dit --name webserver-test -p 8080:80 -v
/home/user/website/:/usr/local/apache2/htdocs/ httpd:2.4

> cd /home/rover/public_html

> echo """
  <html> <head>
  <h1> Welcome to SCIS </h1>
  </head> </html> """ > index.html
```



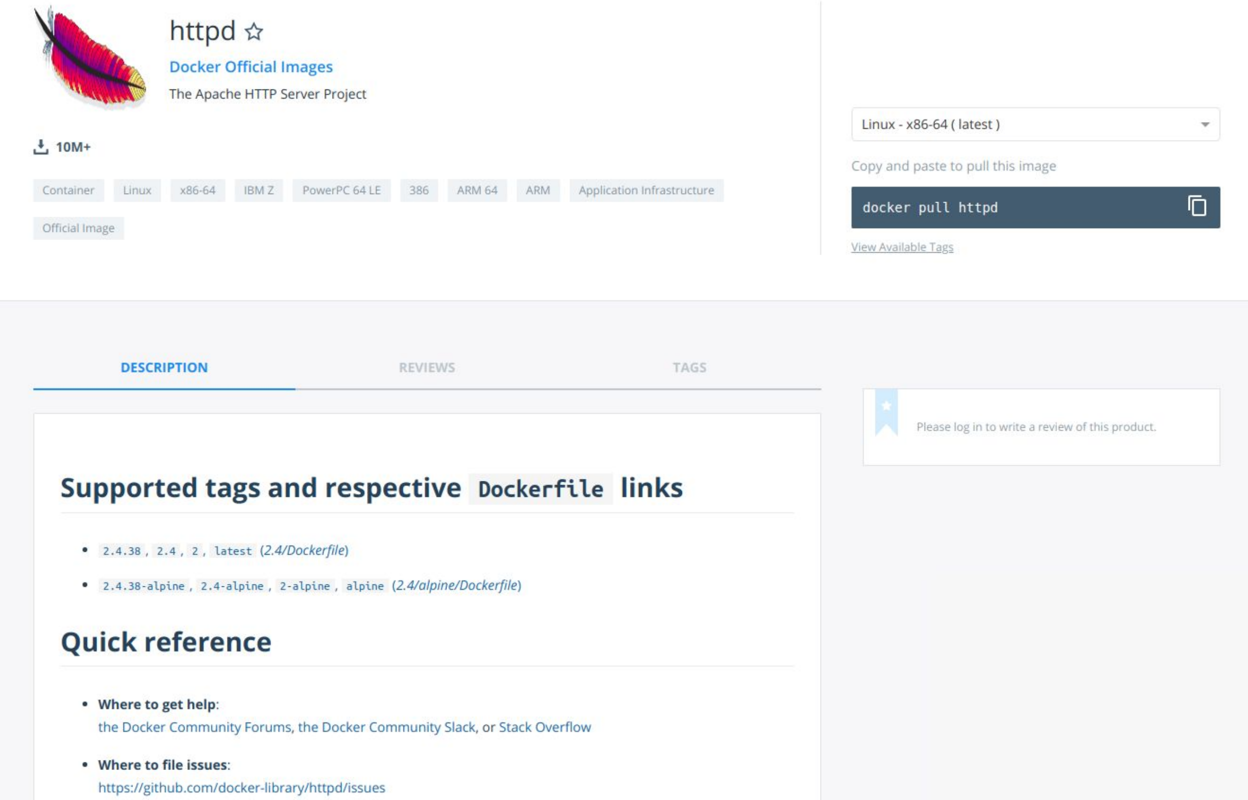
```
rover@metal1:~$ mkdir public_html
rover@metal1:~$ sudo docker run -dit --name webserver-test -p 8080:80 -v /home/rover/public_html/:/usr/local/apache2/htdocs/ httpd:2.4
Unable to find image 'httpd:2.4' locally
2.4: Pulling from library/httpd
6ae821421a7d: Already exists
0ceda4df88c8: Pull complete
24f08eb4db68: Pull complete
ddf4fc318081: Pull complete
fc5812428ac0: Pull complete
Digest: sha256:5e7992fcdaa214d5e88c4dfde274befe60d5d5b232717862856012bf5ce31086
Status: Downloaded newer image for httpd:2.4
c8df1dc8e72f59c1db09b2e0379ed94e3bba59c7692eeebca9380d8fa58d4813
rover@metal1:~$ cd /home/rover/public_html/
rover@metal1:~/public_html$ echo """
> <html>
> <head>
> <h1> Welcome to SCIS </h1>
> </head> </html> """ > index.html
rover@metal1:~/public_html$ 
```



Welcome to SCIS

Docker Hub

- Docker Hub is a cloud-based repository in which Docker users and partners create, test, store and distribute container images.
- https://hub.docker.com/
- Users can host and share their own custom images
- Apache HTTP server image: https://hub.docker.com/_/httpd



httpd ☆

Docker Official Images

The Apache HTTP Server Project

10M+

Container | Linux | x86-64 | IBM Z | PowerPC 64 LE | 386 | ARM 64 | ARM | Application Infrastructure | Official Image

Linux - x86-64 (latest)

Copy and paste to pull this image

```
docker pull httpd
```

View Available Tags

Please log in to write a review of this product.

Supported tags and respective `Dockerfile` links

- `2.4.38`, `2.4`, `2`, `latest` (*2.4/Dockerfile*)
- `2.4.38-alpine`, `2.4-alpine`, `2-alpine`, `alpine` (*2.4/alpine/Dockerfile*)

Quick reference

- **Where to get help**:
  the Docker Community Forums, the Docker Community Slack, or Stack Overflow
- **Where to file issues**:
  https://github.com/docker-library/httpd/issues

Apache HTTP server image on Docker Hub

Outline

- Container recipe
- Why should I care?
- A quick docker demo
- **Building blocks**
- Security

Building blocks of containers

- Again.. what is a container?
- Cgroups
- Namespaces

Building blocks of containers

- **Again.. what is a container?**
- Cgroups
- Namespaces

What is a container?

- How it "feels" like:
  - own process space
  - own network space
  - run stuff as root
  - can install packages
  - can run services
  - can mess up routing, iptables ...

What is a container?

- It's not quite like a VM:
  - uses the host kernel
  - can't boot a different OS
  - can't have its own modules
  - doesn't need init as PID 1
  - doesn't need syslogd, cron...
- It's just normal processes on the host machine
  - contrast with VMs which are opaque

Building blocks of containers

- Again.. what is a container?
- **Cgroups**
- Namespaces

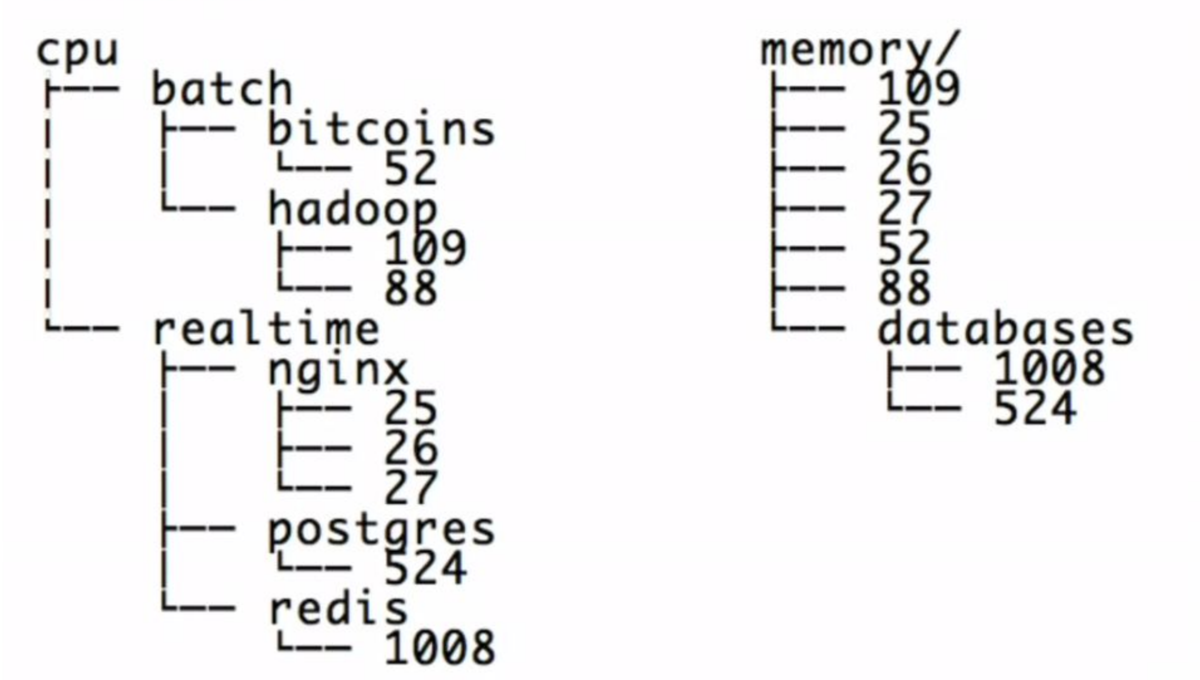
Control Groups (Cgroups)

- Resource metering and limiting
  - memory
  - CPU
  - block I/O
  - network*
- Device node (/dev/*) access control
- Crowd control

Control Groups (Cgroups)

- Each subsystem has a hierarchy (tree)
    - separate hierarchies for for CPU, memory, block I/O...
- Hierarchies are independent
    - the trees for e.g. memory and CPU can be different
- Each process is in a node in each hierarchy
    - think of each hierarchy as a different dimension
- Each hierarchy starts with 1 node (the root)
- Each node = group of processes
    - sharing the same resources

Control Groups (Cgroups)



```
cpu                                memory/
├── batch                          ├── 109
│   ├── bitcoins                   ├── 25
│   │   └── 52                     ├── 26
│   └── hadoop                     ├── 27
│       ├── 109                    ├── 52
│       └── 88                     ├── 88
└── realtime                       └── databases
    ├── nginx                          ├── 1008
    │   ├── 25                         └── 524
    │   ├── 26
    │   └── 27
    ├── postgres
    │   └── 524
    └── redis
        └── 1008
```

source: docker-con EU 2015

Outline

- Container recipe
- Why should I care?
- A quick docker demo
- **Building blocks**
- Security

Building blocks of containers

- Again.. what is a container?
- Cgroups
- Namespaces

Building blocks of containers

- **Again.. what is a container?**
- Cgroups
- Namespaces

What is a container?

- How it "feels" like:
  - own process space
  - own network space
  - run stuff as root
  - can install packages
  - can run services
  - can mess up routing, iptables ...

What is a container?

- It's not quite like a VM:
  - uses the host kernel
  - can't boot a different OS
  - can't have its own modules
  - doesn't need init as PID 1
  - doesn't need syslogd, cron...
- It's just normal processes on the host machine
  - contrast with VMs which are opaque

Building blocks of containers

- Again.. what is a container?
- **Cgroups**
- Namespaces

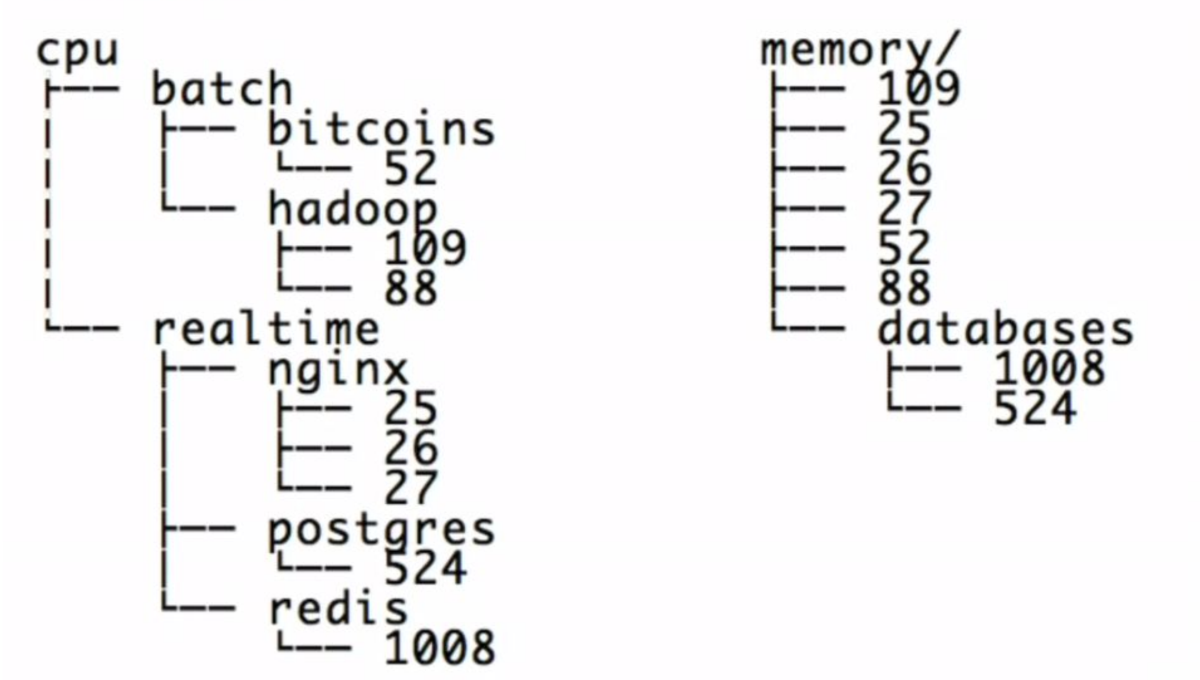
Control Groups (Cgroups)

- Resource metering and limiting
  - memory
  - CPU
  - block I/O
  - network*
- Device node (/dev/*) access control
- Crowd control

Control Groups (Cgroups)

- Each subsystem has a hierarchy (tree)
  - separate hierarchies for for CPU, memory, block I/O...
- Hierarchies are independent
  - the trees for e.g. memory and CPU can be different
- Each process is in a node in each hierarchy
  - think of each hierarchy as a different dimension
- Each hierarchy starts with 1 node (the root)
- Each node = group of processes
  - sharing the same resources

Control Groups (Cgroups)



```
cpu
├── batch
│   ├── bitcoins
│   │   └── 52
│   └── hadoop
│       ├── 109
│       └── 88
└── realtime
    ├── nginx
    │   ├── 25
    │   ├── 26
    │   └── 27
    ├── postgres
    │   └── 524
    └── redis
        └── 1008
```

```
memory/
├── 109
├── 25
├── 26
├── 27
├── 52
├── 88
└── databases
    ├── 1008
    └── 524
```

source: docker-con EU 2015

Memory cgroup: accounting

- Keeps track of pages used by each group:
    - file (read/ write from block devices)
    - active (recently accessed)
    - inactive (candidate for eviction)
- Each page is "charged" to a group
- Page can be shared across multiple groups
    - e.g. multiple processes reading from the same files when pages are shared, only one group "pays" for a page.

Memory cgroup: accounting

- Keeps track of pages used by each group:
  - file (read/ write from block devices)
  - active (recently accessed)
  - inactive (candidate for eviction)
- Each page is "charged" to a group
- Page can be shared across multiple groups
  - e.g. multiple processes reading from the same files when pages are shared, only one group "pays" for a page.

Memory cgroup: limits

- Each group can have its own limits
    - limits are optional
    - two kinds of limits:
        - Soft limits
        - Hard limits
- Soft limits are not enforced
    - they influence reclaim memory pressure

Memory cgroup: limits

- Hard limits will trigger a per-group OOM killer
- Limits can be set for different kinds of memory
  - physical memory
  - kernel memory
  - total memory
- Multiple groups use the same page, only first one is "charged"
  - but if it stops using it, the charge is moved to another group.

CPU cgroup

- Keep track of user/system CPU time
- Keeps track of usage per CPU
- Allows to set weights

CPUset cgroup

- Pin groups to specific CPU(s)
- Reserve CPUs for specific apps
- Avoid processes bouncing between CPUs

Blkio cgroup

- Keeps track of I/O for each group
    - per block device
    - read vs write
- Set throttle (limits) for each group
    - per block device
    - read vs write

net_cls and net_prio cgroup

- Automatically set traffic class or priority, for traffic generated by processes in the group
- Only works for egress traffic
- Net_cls will assign traffic to a class
  - class then has to be matched with tc/iptables, otherwise traffic just flows normally
- Net_prio will assign traffic to a priority
  - priorities are used by queuing disciplines

Devices cgroup

- Controls what the group can do on devices nodes
- Permissions include read/write
- Typical use:
  - allow /dev/{tty,zero,random,null}
  - deny everything else
- A few interesting nodes that we can expose:
  - /dev/net/tun (network interface manipulation)
  - /dev/fuse (filesystems in use space)
  - /dev/dri (GPU)

Freezer cgroup

- Allows to freeze a group of processes
- Similar in functionality to SIGSTOP
- Cannot be detected by the processes
    - that's how it differs from SIGSTOP
- Can be used in process migration

Building blocks of containers

- Again.. what is a container?
- Cgroups
- **Namespaces**

Namespaces

- Provide processes with their own system view
  - Cgroups = limits how much you can use;
  - Namespaces = limits what you can see
- Multiple namespaces:
  - pid
  - net
  - mnt
  - uts
  - ipc
  - user
- Each process is in one namespace of each type

pid namespace

- Processes within a PID namespace only see processes in the same PID namespace
- Each PID namespace has its own numbering
  - starting at 1
- If PID 1 goes away, whole namespace is killed
- Those namespaces can be nested
- A process ends up having multiple PIDs
  - one per namespace in which its nested

Network namespaces: in theory

- Processes within a given network namespace get their own private network stack, including:
    - network interfaces
    - routing tables
    - iptable rules
    - sockets

Network namespaces: in practice

- Use virtual interfaces acting as a cross-over cable
- eth0 in container network namespace, paired with vethXX in host network namespace.
- All the vethXX are bridged together via virtual switch inside the container host.
    - Docker calls the bridge docker0

Mnt namespace

- Process can have their own root fs
  - conceptually close to chroot
- Processes can also have "private" mounts
  - /tmp
  - masking of /proc, /sys
- Mounts can be totally private, or shared

Uts namespace

- Let containers set their own hostname.
  - gethostname/ sethostname

IPC namespace

- Allows a process (or group of processes) to have own:
  - IPC semaphores
  - IPC message queues
  - IPC shared memory
- without risk of conflict with other instances.

User namespace

- Allows to map UID.
  - UID 0 in container --> UID 9999 in host
  - UID 1000 in container --> UID 1564 in host
- UID in containers becomes irrelevant
- Security improvement

Outline

- Container recipe
- Why should I care?
- A quick docker demo
- Building blocks
- **Security**

Use RunC Flaw to gain Root access on Host

- The vulnerability, identified as **CVE-2019-5736**, was discovered by two open source security researchers on 11th Feb 2019.
- “High level” container runtimes like Docker does image creation and management
- Use "Low level" runC to handle tasks related to running containers
  - creating a container
  - attaching a process to an existing container (docker exec)

The Vulnerability

Overview given by the runC team:

The vulnerability allows a malicious container to (with minimal user interaction) overwrite the host runc binary and thus gain root-level code execution on the host. The level of user interaction is being able to run any command ... as root within a container in either of these contexts:

- Creating a new container using an attacker-controlled image.
- Attaching (docker exec) into an existing container which the attacker had previous write access to.

Bibliography

1. What is container? | https://www.docker.com/resources/what-container
2. Demystifying containers 101 | https://tinyurl.com/yxd7fnpe
3. Getting started with Docker | https://tinyurl.com/y35e879j , http://bit.do/eJwmy
4. Docker for development and deployment | http://bit.do/eJwmH
5. RunC Flaw Lets Attackers Escape Linux Containers to Gain Root on Hosts | https://is.gd/3qPVsQ
6. CVE-2019-5736 | https://nvd.nist.gov/vuln/detail/CVE-2019-5736
7. Explaining CVE-2019-5736 | http://bit.do/eJLcG
8. LXC | https://linuxcontainers.org/lxc/introduction/ , https://is.gd/i0F9i4
9. Container basics | https://is.gd/Eq4gaP https://is.gd/pGgMqW
10. Namespaces in operation | https://lwn.net/Articles/531114/
11. Future of Linux Containers | https://is.gd/U0QR9K
12. cgroups | https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt , https://is.gd/1x1ihc
13. Attacks on Linux Containers | https://ieeexplore.ieee.org/document/7854163

Bibliography

14. Comparing different containers | https://ieeexplore.ieee.org/document/8075934
15. Beginners Guide to Containers Technology and How it Actually Works[video] | https://is.gd/CzTb1d
16. My Docker Guide | https://github.com/w4rb0y/devOpsNotes/blob/master/dockerGuide.md
17. Google Cloud | https://cloud.google.com/containers/
18. Containers from scratch | https://ericchiang.github.io/post/containers-from-scratch/

Thank You

Query?

Next slides are still under development.

Container without Docker

- Setup a file system
- chroot
- unshare
- nsenter
- bind mounts
- cgroups

chroot

A **chroot** is an operation that changes the apparent root directory for the current running process and their children. A program that is run in such a modified environment cannot access files and commands outside that environmental directory tree. This modified environment is called a **chroot jail.**

chroot

Download the rootfs

```
wget https://is.gd/nEYUgv

sudo tar -zxf rootfs.tar.gz

sudo chroot rootfs /bin/bash

mount -t proc proc /proc

pkill top
```

chroot

```
ldd /bin/bash | awk '{print $3}' | sed '/^$/d'
```

cgroups

```
  ls /sys/fs/cgroup/
sudo su

mkdir /sys/fs/cgroup/memory/demo

ls /sys/fs/cgroup/memory/demo/
```

unshare

```
sudo chroot rootfs /bin/bash

sudo unshare -p -f --mount-proc=$PWD/rootfs/proc \

      chroot rootfs /bin/bash
```

-p -> pid

-f -> Fork the specified program as a child process of unshare rather than running it directly. This is useful when creating a new PID namespace.

Thank You