Docker: Introdução e experiências iniciais com uma tecnologia de virtualização leve e ágil

Javier Richard Quinto Ancieta (MSc Candidate)
Raphael Vicente Rosa (Phd Candidate)
Prof. Christian Esteve Rothenberg

10/12/2014
Agenda

• Introduction to Linux Containers
• Docker: Basics (theory and examples)
• Experiences at the INTRIG Lab @ FEEC/UNICAMP
• Hands-on examples
  – Docker and Networking
    Docker & OVS, tunneling, etc.
Credits

* http://docker.io/
* http://docker.com/
* https://github.com/docker/docker
* @docker
* @jpetazzo – OSCON 2014
Intro

CONTAINERS
What’s the problem?

<table>
<thead>
<tr>
<th></th>
<th>my laptop</th>
<th>your laptop</th>
<th>QA</th>
<th>staging</th>
<th>prod on cloud VM</th>
<th>prod on bare metal</th>
</tr>
</thead>
<tbody>
<tr>
<td>message queue</td>
<td></td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
A Similar Matrix...

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution: the *intermodal shipping container*
Solution:
the *intermodal shipping container*
Solution to the deployment problem: the *Linux* container

- Units of software delivery (**ship it!**)
- run **anything**
  - if it can run on the host, it can run in the container
  - i.e., if it can run on a Linux kernel, it can run
Deploy *(almost)* everywhere

- Linux servers
- VMs or bare metal
- Any distro
- Kernel 3.8 (or RHEL 2.6.32)
What’s a Container?

• High level approach: it's a lightweight VM
  – own process space
  – own network interface
  – can run stuff as root
  – can have its own /sbin/init
    • (different from the host)
What’s a Container?

• Low level approach: it's chroot on steroids
  – can also *not* have its own /sbin/init
  – container = isolated process(es)
  – share kernel with host
  – no device emulation (neither HVM nor PV)
Separation of Concerns

Developer
• inside my container:
  – my code
  – my libraries
  – my package manager
  – my app
  – my data

Operations
• outside the container:
  – logging
  – remote access
  – network configuration
  – monitoring
How does it work?

• Isolation with *namespaces*
  – pid
  – mnt
  – net
  – uts
  – ipc
  – user

• Isolation with *cgroups*
  – memory
  – cpu
  – blkio
  – devices
Containers: Why is this a hot topic?

• LXC (Linux Containers) have been around for *years*
• Lightweight, fast, disposable... virtual environments
  – boot in milliseconds
  – just a few MB of intrinsic disk/memory usage
  – bare metal performance is possible
• The new way to build, ship, deploy, run your apps!
• Everybody* wants to deploy containers now
• Tools like Docker made containers very easy to use
And now...

DOCKER
Docker

- Open Source engine to commoditize LXC
  - the whole point is to commoditize,
  - i.e. make it ridiculously easy to use
Containers

Before Docker

After Docker
Docker-what?
The Big Picture

- Open Source engine to commoditize LXC
- using copy-on-write for quick provisioning
- allowing to create and share images
- standard format for containers
- (stack of layers; 1 layer = tarball+metadata)
- standard, reproducible way to easily build
- trusted images (Dockerfile, Stackbrew...
Docker-what?
Under the hood

• rewrite of dotCloud internal container engine
  – original version: Python, tied to dotCloud's internal stuff
  – released version: Go, legacy-free
• the Docker daemon runs in the background
  – manages containers, images, and builds
  – HTTP API (over UNIX or TCP socket)
  – embedded CLI talking to the API
• Open Source (GitHub public repository + issue tracking)
• user and dev mailing lists
• FreeNode IRC channels #docker, #docker-dev
Docker-what?
The Ecosystem

• Docker Inc. (formerly dotCloud Inc.)
  – ~30 employees, VC-backed
  – SaaS and support offering around Docker

• Docker, the community
  – more than 300 contributors, 1500 forks on GitHub
  – dozens of projects around/on top of Docker
  – x100k trained developers
Working with Docker

• On your servers (Linux)
  – Packages (Ubuntu, Debian, Fedora, Gentoo...)
  – Single binary install (Golang)
  – Easy provisioning on Rackspace, Digital Ocean, EC2...

• On your dev env (Linux, OS X, Windows)
  – Vagrantfile (describe machine config reqs)
  – boot2docker (25 MB VM image)
  – Natively (if you run Linux)
Why not?! Let’s try it...

HANDS-ON
Get in Touch

• First Contact
  – https://www.docker.com/tryit/

• All useful information
  – https://docs.docker.com/
Install

• On Ubuntu 14.04
  – $ sudo apt-get install docker.io

• Other distros
  – https://docs.docker.com/installation/#installation

• *windows / OSX: boot2docker
Dockerizing Applications: A "Hello world"

$ sudo docker run ubuntu:14.04 /bin/echo 'Hello world'
   – First App

$ sudo docker run -t -i ubuntu:14.04 /bin/bash
   – Inside a container (Comparison with outside world)

$ sudo docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
   – Something about id/name, logs, stop
Working with Containers

$ sudo docker .... [version]
  – What docker client can do
  – More: https://docs.docker.com/reference/commandline/cli/

$ sudo docker run -d -P training/webapp python app.py
  – Let’s inspect, stop, rm, top, ps, ...
Working with Images

$ sudo docker images
  – Repo, tag, id, ...

$ sudo docker search ...
  – Or https://hub.docker.com/

Creating your images:

$ sudo docker pull training/sinatra
$ sudo docker run -t -i training/sinatra /bin/bash
# gem install json
$ sudo docker commit -m="Added json gem" -a="Kate Smith" ID ouruser/sinatra:v2
Working with Images

Dockerfiles:

“# This is a comment
FROM ubuntu:14.04
MAINTAINER Kate Smith <ksmith@example.com>
RUN apt-get update && apt-get install -y ruby ruby-dev
RUN gem install sinatra”
$ sudo docker build -t="ouruser/sinatra:v2" .
— More: https://docs.docker.com/reference/builder

Tagging:

$ sudo docker tag ID ouruser/sinatra:devel
Managing Data in Containers

- **A simple mount:**
  
  
  $ sudo docker run -d -P --name web -v /webapp training/webapp python app.py

- **Or**
  
  $ sudo docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py

- **And then:**
  
  $ sudo docker run -d --volumes-from web --name db training/postgres

- **Removing volumes**
  
  $ sudo docker rm -v /src/webapp
Linking Containers

• Links:
  – Env Variables
  – /etc/hosts updates

$ sudo docker run -d --name db training/postgres

$ sudo docker run -d -P --name web --link db:db training/webapp /bin/bash
Docker in Production...

DEVOPS
The Docker Workflow 1/2

• Work in dev environment
  (local machine or container)
• Other services (databases etc.) in containers
  (and behave just like the real thing!)
• Whenever you want to test « for real »:
  – Build in seconds
  – Run instantly
The Docker Workflow 2/2

• Satisfied with your local build?
  – Push it to a *registry* (public or private)
  – Run it (automatically!) in CI/CD
  – Run it in production
  – Happiness!

• Something goes wrong? *Rollback painlessly!*
DevOps

WORKED FINE IN DEV

OPS PROBLEM NOW
Authoring images with run/commit

1) docker run ubuntu bash
2) apt-get install this and that
3) docker commit <containerid> <imagename>
4) docker run <imagename> bash
5) git clone git://.../mycode
6) pip install -r requirements.txt
7) docker commit <containerid> <imagename>
8) repeat steps 4-7 as necessary
9) docker tag <imagename> <user/image>
10) docker push <user/image>
Authoring images with run/commit

• Pros
  – Convenient, nothing to learn
  – Can roll back/forward if needed

• Cons
  – Manual process
  – Iterative changes stack up
  – Full rebuilds are boring, error-prone
Authoring images with a Dockerfile

FROM ubuntu
RUN apt-get -y update
RUN apt-get install -y g++
RUN apt-get install -y make wget
EXPOSE 80
CMD ["/bin/ping"]

docker build -t AUTHOR/DOCKER-NAME.
Authoring images with a Dockerfile

- Minimal learning curve
- Rebuilds are easy
- Caching system makes rebuilds faster
- Single file to define the whole environment!
Sysadmin chores

- Backups
- Logging
- Remote access
File-level Backups

• **Use volumes**
  
  $ docker run --name mysql.data -v /var/lib/mysql
  busybox true
  $ docker run --name mysql --volumes-from mysql.data
  mysql
  $ docker run --rm --volumes-from mysql.data
  mysqlbackup \ tar -cJf- /var/lib/mysql | stream-it-to-the-cloud.py

• Of course, you can use anything fancier than tar (e.g. rsync, tarsnap...)


Data-level backups

• **Use links**
  
  ```bash
  $ docker run --name mysql mysql
  $ docker run --rm --link mysql:db mysqlbackup
  $ mysqldump --all-databases | stream-it-to-the-cloud.py
  ```

• **Can be combined with volumes**
  – put the SQL dump on a volume
  – then backup that volume with file-level tools (previous slide)
Logging for legacy apps

• Legacy = let me write to eleventy jillion arbitrary files in /var/lib/tomcat/logs!

• Solution: volumes
  $ docker run --name logs -v /var/lib/tomcat/logs busybox true
  $ docker run --name tomcat --volumes-from logs my_tomcat_image

  – Inspect logs:
    $ docker run --rm --volumes-from logs ubuntu bash
  – Ship logs to something else:
    $ docker run --name logshipper --volumes-from logs sawmill
Remote Access

• If you own the host: SSH to host + nsenter
  https://github.com/jpetazzo/nsenter

• If you don't own the host: SSH in the container
  https://github.com/phusion/baseimage-docker

• More on that topic (“do I need SSHD in containers?”):

• In the future:
  – run separate SSH container
  – log into that
  – “hop” onto the target container
Orchestration

• There's more than one way to do it *(again!)*
  – describe your stack in files
    (Fig, Maestro-NG, Ansible and other CMs)
  – submit requests through an API
    (Mesos)
  – implement something that looks like a PAAS
    (Flynn, Deis, OpenShift)
  – the “new wave”
    (Kubernetes, Centurion, Helios...)
  – OpenStack
Do you (want to) use OpenStack?

• Yes
  – if you are building a PaaS, keep an eye on Solum
    (and consider contributing)
  – if you are moving VM workloads to containers, use Nova
    (that's probably what you already have; just enable the Docker driver)
  – otherwise, use Heat
    (and use Docker resources in your Heat templates)

• No
  – go to next slide
Are you looking for a PaaS?

• Yes
  – CloudFoundry (Ruby, but increasing % Go)
  – Deis (Python, Docker-ish, runs on top of CoreOS)
  – Dokku (A few 100s of line of Bash!)
  – Flynn (Go, bleeding edge)
  – OpenShift geard (Go)

• Choose wisely (or go to the next slide)
  – http://blog.lusis.org/blog/2014/06/14/paas-for-realists/
    “I don’t think ANY of the current private PaaS solutions are a fit right now.”
How many Docker hosts do you have?

• Only one per app or environment
  – Fig

• A few (up to ~10)
  – Maestro-NG
  – your favorite CM (e.g. Ansible has a nice Docker module)

• A lot
  • Mesos
    – have a look at (and contribute to) the “new wave” (Centurion, Helios, Kubernetes...)
Performance: measure things

• cgroups give us per-container...
  – CPU usage
  – memory usage (fine-grained: cache and resident set size)
  – I/O usage (per device, reads vs writes, in bytes and in ops)

• cgroups don't give us...
  – network metrics (have to do tricks with network namespaces)

https://github.com/google/cadvisor
http://jpetazzo.github.io/2013/10/08/docker-containers-metrics/
Our Experience...

INTRIG LAB
Most images already available

• We can built our own dockerfiles
  – Images without being verified
  – Some of them old
  – Custom configs

• Old x86 machines (Core 2Duo, 4GB, 250 GB)
  – Monitoring containers performance
  – Following Apps recommendations
Inside only!

• Evaluating...
Outside world: 2015

• Start deploying our private repository
• Containers configuration management  
  – Documentation
• No need for CM platform right now!
• Security configs required for production  
  – Under analysis!
Training...

EXAMPLES
Hands-on #1

# Access to the VM recently started:
$ ssh ubuntu@192.168.122.179

# To start each VM remotely:
$ sudo virsh net-start default
$ sudo virsh start ubuntu1
$ sudo virsh start ubuntu2

# Use different terminals to access each VM
From terminal 1
$ ssh ubuntu@192.168.123.2
From terminal 2
$ ssh ubuntu@192.168.123.3
# First, we check if docker is up
$ sudo ps aux |grep docker
root  624  0.0  1.5 328816 11536 ?   Ssl   17:30   0:00 /usr/bin/docker -d

# If docker is not up then give the below command:
service docker start

# In each VM, search and pull the pre-configured container from docker hub
KVM-1: $ docker search intrigue/tutorial
KVM-2: $ docker search intrigue/tutorial
KVM-1: $ docker pull intrigue/tutorial
KVM-2: $ docker pull intrigue/tutorial

# Check if docker was correctly downloaded from repositories Docker Hub
KVM-1: $ docker images
KVM-2: $ docker images
# Virtual Machine 1 (KVM-1)

# Resetting the docker interface
sudo ip link set docker0 down
sudo brctl delbr docker0
sudo brctl addbr docker0

# Assigning IP to the Docker interface
sudo ip addr add 172.16.1.1/24 dev docker0
sudo ip link set docker0 up
sudo ovs-vsctl add-br br0

# Creating a tunnel GRE
sudo ovs-vsctl add-port br0 gre0 -- set interface
gre0 type=gre options:remote_ip=192.168.123.3

# Adding the bridge ‘br0’ as interface to the bridge ‘docker0’
sudo brctl addif docker0 br0
# Virtual Machine 2 (KVM-2)

# Resetting the docker interface
```
sudo ip link set docker0 down
sudo brctl delbr docker0
sudo brctl addbr docker0
```

# Assigning IP to the Docker interface
```
sudo ip addr add 172.16.1.2/24 dev docker0
sudo ip link set docker0 up
sudo ovs-vsctl add-br br0
```

# Creating a tunnel GRE
```
sudo ovs-vsctl add-port br0 gre0 -- set interface
gre0 type=gre options:remote_ip=192.168.123.2
```

# Adding the bridge ‘br0’ as interface to the bridge ‘docker0’
```
sudo brctl addif docker0 br0
```
# Virtual Machine 1 (KVM-1)

# Activate docker for the container 1

```bash
$ docker run -i -t --privileged --name=container1 --hostname=container1 --publish 127.0.0.1:2222:22 intrigue/tutorial:v3 /bin/bash
```

# If you get the below mentioned prompt, then configure an IP address for it. This prompt signifies that you have successfully started the container.

```bash
root@container1:/#
root@container1:/# ifconfig eth0 172.16.1.11
netmask 255.255.255.0
root@container1:/# route add default gw 172.16.1.11
```
# GRE Tunnel in Docker 4/4

# Virtual Machine 2 (KVM-2)

# Activate docker for the container 2

$docker run -i -t --privileged --name=container2 --hostname=container2 --publish 127.0.0.1:2222:22 intrigue/tutorial:v3 /bin/bash

# If you get the below mentioned prompt, then configure an IP address for it. This prompt signifies that you have successfully started the container.

root@container2:/#
root@container2:/# ifconfig eth0 172.16.1.12 netmask 255.255.255.0
root@container2:/# route add default gw 172.16.1.2
Testing GRE Tunnel

# Testing the connectivity between dockers

- From the Container 1 to Container 2
  Container1:/# ping 172.16.1.12
- From the Container 2 to Container 1
  Container2:/# ping 172.16.1.11

# Copy the binary 'iperf' from KVM1 to the container docker of KVM-1

KVM1: sudo cp /usr/bin/iperf /var/lib/docker/aufs/diff/<ID-docker1>/usr/bin/
KVM2: sudo cp /usr/bin/iperf /var/lib/docker/aufs/diff/<ID-docker2>/usr/bin/
# Testing GRE Tunnel

# Verify the RTT using IPERF

# From the Container Docker
172.16.1.12, we launch Iperf Server
listening on TCP port 5001
$ sudo iperf -s

# From the another Container Docker
172.16.1.11, we launch Iperf Client
connecting to 172.16.1.12, TCP port
5001
$ sudo iperf -c 172.16.1.12

What can you say of the “Bandwith”?
Hands-on #2

**Fig:** Work Environment with Docker and Open vSwitch
Docker with Open vSwitch and GRE Tunnel 1/7

Creating GRE Tunnel using OVS

# Virtual Machine 1 (KVM-1)

$ sudo ovs-vsctl del-br br0
$ sudo ovs-vsctl add-br br0
$ sudo ovs-vsctl add-br br2
$ sudo ovs-vsctl add-port br0 tep0 -- set interface tep0 type=internal
$ sudo ifconfig tep0 192.168.200.21 netmask 255.255.255.0
$ sudo ovs-vsctl add-port br2 gre0 -- set interface gre0 type=gre\options:remote_ip=192.168.123.3

# Virtual Machine 2 (KVM-2)

$ sudo ovs-vsctl del-br br0
$ sudo ovs-vsctl add-br br0
$ sudo ovs-vsctl add-br br2
$ sudo ovs-vsctl br0 tep0 -- set interface tep0 type=internal
$ sudo ifconfig tep0 192.168.200.22 netmask 255.255.255.0
$ sudo ovs-vsctl add-port br2 gre0 -- set interface gre0 type=gre\options:remote_ip=192.168.123.2
# Virtual Machine 1 (KVM-1)

# Delete the container docker created in the last exercise
$ docker stop container1
$ docker rm container1

# Create two containers docker and set the network mode to none. Each containers is on a local variable.
$ C1=$(docker run -d --net=none -t -i --privileged --name=container1 --hostname=container1 intrig/tutorial:v3 /bin/bash)
$ C2=$(docker run -d --net=none -t -i --privileged --name=container2 --hostname=container2 intrig/tutorial:v3 /bin/bash)
Starting Containers

# Virtual Machine 2 (KVM-2)

# Delete the container docker created in the last exercise
$ docker stop container2
$ docker rm container2

# Create two containers docker and set the network mode to none. Each containers is on a local variable.
$ C3=$(docker run -d --net=none -t -i --privileged --name=container3 --hostname=container3 intrig/tutorial:v3 /bin/bash)
$ C4=$(docker run -d --net=none -t -i --privileged --name=container4 --hostname=container4 intrig/tutorial:v3 /bin/bash)
# Virtual Machine 1 (KVM-1)

# To know the PID value of the container created, we use the script findPID.sh

$.findPID.sh $C1

The PID value of the container created is: 6485 (for example). Same for $C2

# Bind dockers with a Open vSwitch interface

$ sudo ./ovswork-1.sh br2 $C1 1.0.0.1/24 1.0.0.255 1.0.0.254 10
$ sudo ./ovswork-1.sh br2 $C2 1.0.0.2/24 1.0.0.255 1.0.0.254 20

# Virtual Machine 2 (KVM-2)

# Bind dockers with a OpenVswitch interface

$ sudo ./ovswork-1.sh br2 $C3 1.0.0.3/24 1.0.0.255 1.0.0.254 10
$ sudo ./ovswork-1.sh br2 $C4 1.0.0.4/24 1.0.0.255 1.0.0.254 20
Docker with Open vSwitch and GRE Tunnel 5/7

Initiating Docker

# Using different terminals, start the container1, container2, container3, container4

From terminal 1:
$ docker start -a -i container1

From terminal 2:
$ docker start -a -i container2

From terminal 3:
$ docker start -a -i container3

From terminal 4:
$ docker start -a -i container4
Testing of connectivity

# From the Container1 (Terminal 1)
Container1$ ping 1.0.0.3 -c 2
Container1$ ping 1.0.0.4 -c 2

# From the Container3 (Terminal 3)
Container3$ ping 1.0.0.1 -c 2
Container3$ ping 1.0.0.2 -c 2

What ping is successful? And why?

root@container1:/# ping 1.0.0.3
PING 1.0.0.3 (1.0.0.3) 56(84) bytes of data.
64 bytes from 1.0.0.3: icmp_seq=1 ttl=64 time=2.52 ms
64 bytes from 1.0.0.3: icmp_seq=2 ttl=64 time=0.651 ms
Testing of connectivity with Iperf

# Verify the RTT using IPERF
# From the Container #1 1.0.0.3 launch Iperf Server listening on TCP port 5001
$ sudo iperf -s
# From the another Container #3, launch Iperf Client connecting to 1.0.0.3, TCP port 5001
$ sudo iperf -c 1.0.0.3

What can you say of the “Bandwith”?

# Virtual Machine 1 (KVM-1)
$ sudo ovs-vsctl show br2
$ sudo ovs-ofctl show br2
$ sudo ovs-ofctl dump-flows br2

# Virtual Machine 2 (KVM-2)
$ sudo ovs-vsctl show br2
$ sudo ovs-ofctl show br2
$ sudo ovs-ofctl dump-flows br2
THANKS