Docker Toolbox

Hands-On Workshop

rackspace®

the #1 managed cloud company

Agenda

- LESSON 1 - PART 1 Intro to Docker (15 min)
  - Docker, Cgroups, Kernel Namespaces, and Docker Images
- LESSON 1 - PART 2 – Toolbox Overview (15 min)
  - Docker Hub, Trusted build system, Weave, Flannel, etcd, CoreOS, Magnum, Kubernetes, and how Docker works with OpenStack
- Hands-on LAB 1 - Using docker-machine (15 min)
  - Spin up Docker containers on Rackspace Cloud, Switch between docker hosts.
  - How to run ad-hoc containers on the fly for experimentation, How to get a shell on a Docker host, and a Docker container without running sshd in the container.
  - How to move a container from one host to another.
- LESSON 2 – Writing Dockerfiles (15 min)
  - Dockerfile Directives
  - Demonstration of copying files into containers using bind mounts, and copying files out of containers using “docker cp”.
  - Why Dockerfiles are better than customized images made manually and stored with commit.
- Hands-on LAB 2 - Writing Dockerfiles (15 min)
  - Saving a Dockerfile in a source repo, and using a build script for configuration injection.
- LESSON 3 - Linking and Networking containers (15 min)
  - Demonstration of how to use links to communicate between containers on the same host. Using weave to communicate between containers on separate hosts.
- Hands on LAB 3 - Linking containers (15 min)
  - Container linking on the same host, and mesh networking between containers on different hosts.

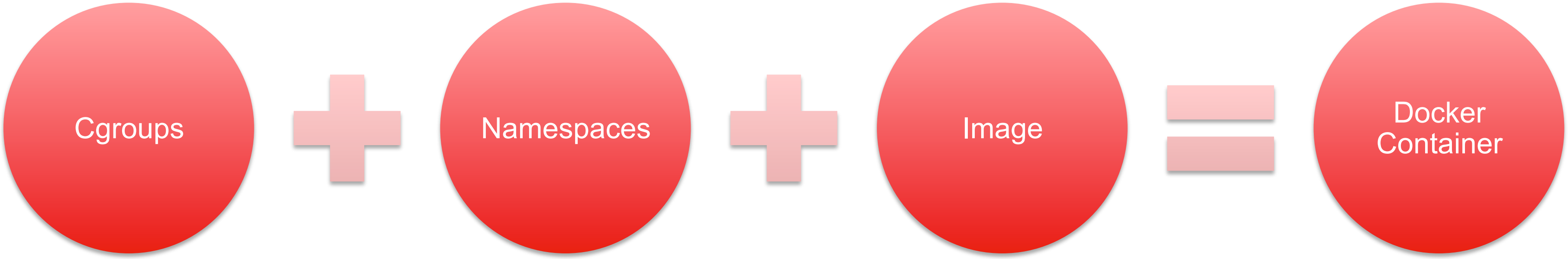
1.1

Lesson One – Part One

Introduction to Docker

Container

- Combines several things
  - Linux Cgroups
  - Kernel Namespaces
  - Docker Image
  - Has a lifecycle



Linux Cgroups

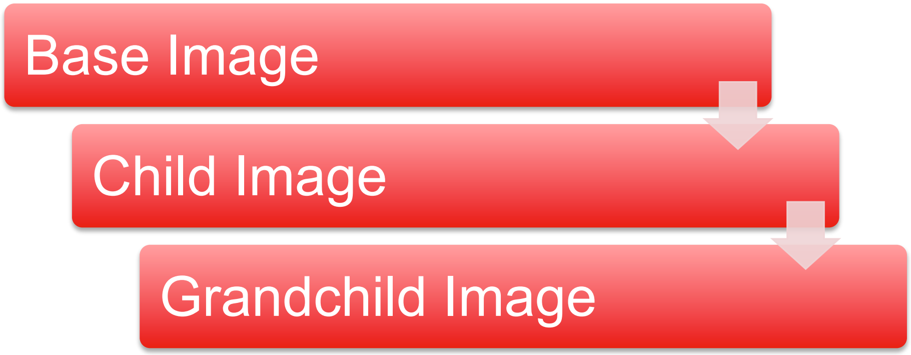
- Kernel Feature
- Groups of processes
- Control resource allocations
  - CPU
  - Memory
  - Disk
  - I/O
- May be nested

Linux Kernel Namespaces

- Kernel Feature
- Restrict your view of the system
  - Mounts (CLONE_NEWNS)
  - UTS (CLONE_NEWUTS)
    - `uname()` output
  - IPC (CLONE_NEWIPC)
  - PID (CLONE_NEWPID)
  - Networks (CLONE_NEWNET)
  - User (CLONE_NEWUSER)
    - Not supported in Docker yet
    - Has privileged/unprivileged modes today
- May be nested

Docker Image

- NOT A FILESYSTEM
- NOT A VHD
- Basically a tar file
- Has a hierarchy
  - Arbitrary depth
- Fits into the Docker Registry



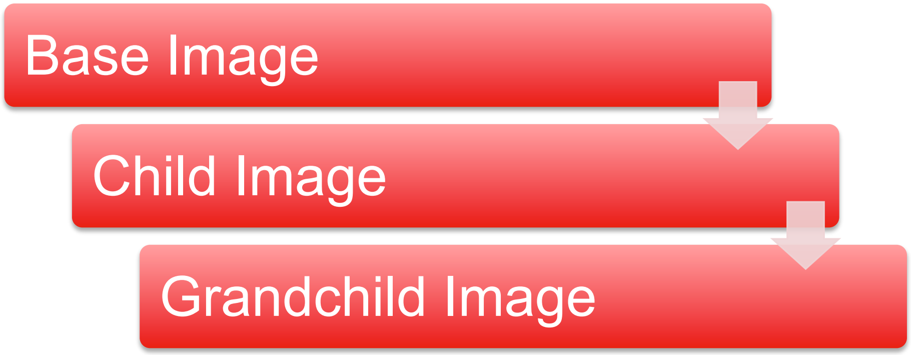
Base Image

Child Image

Grandchild Image

Docker Registry

- Git Repo Semantics
  - Pull
  - Push
  - Commit
  - Hierarchy



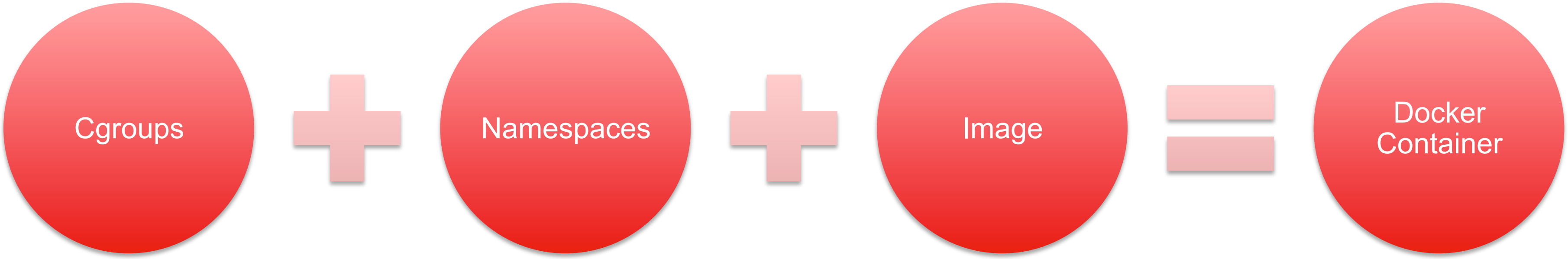
Base Image

Child Image

Grandchild Image

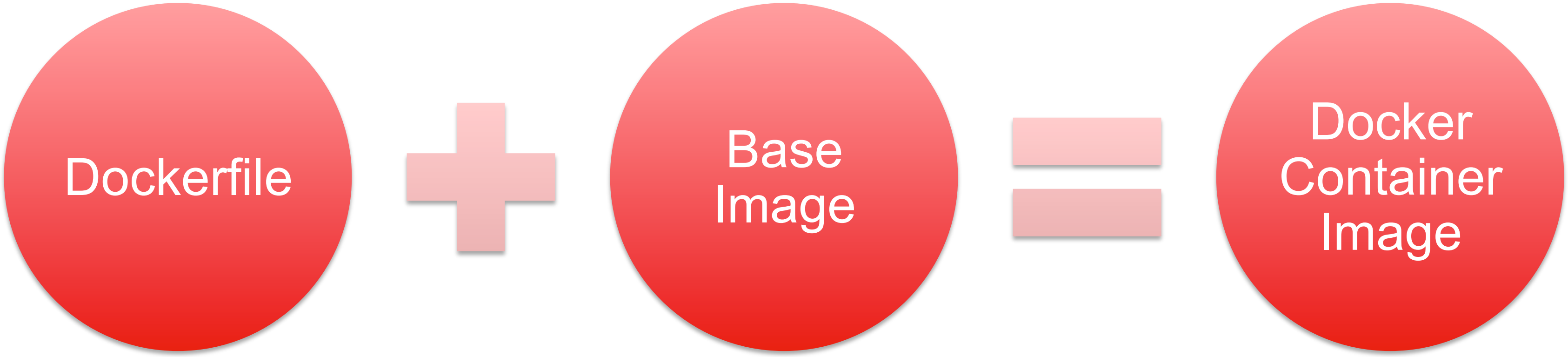
Container

- Combines several things
  - Linux Cgroups
  - Kernel Namespaces
  - Docker Image
  - Has a lifecycle



Dockerfile

- Like a Makefile (shell script with keywords)
- Extends from a Base Image
- Results in a new Docker Image
- Imperative, not Declarative



Dockerfile

Base Image

Docker Container Image

Dockerfile Example

```
FROM centos:centos6
MAINTAINER aotto@aotto.com
RUN yum -y install openssh-server
EXPOSE 22
ADD start.sh /start.sh
CMD /start.sh
```

Dockerfile Example

```
FROM adrian_server_with_ssh
MAINTAINER aotto@aotto.com
RUN yum -y install httpd
EXPOSE 22 80
ADD start.sh /start.sh
CMD /start.sh
```

Docker Container Lifecycle

- The Life of a Container
  - Conception
    - BUILD an Image from a Dockerfile
  - Birth
    - RUN (create+start) a container
  - Reproduction
    - COMMIT (persist) a container to a new image
    - RUN a new container from an image
  - Sleep
    - KILL a running container
  - Wake
    - START a stopped container
  - Death
    - RM (delete) a stopped container
- Extinction
  - RMI a container image (delete image)

Docker CLI Commands (v1.4.1)

attach	Attach to a running container
build	**Build an image from a Dockerfile**
commit	**Create new image from container's changes**
cp	Copy files from containers fs to host
create	Create a new container
diff	Inspect changes on a container's fs
events	Get real time events from the server
exec	Run a command in a running container
export	Stream contents of container as tar
history	Show the history of an image
images	**List images**
import	Create new fs image from a tarball
info	Display system-wide information
inspect	Return low-level info on a container
kill	**Kill a running container**
load	Load an image from a tar archive
login	Login to the docker registry server
logout	Log out from a Docker registry server
logs	Fetch the logs of a container
port	Lookup public-facing port
pause	Pause all processes within a container
ps	**List containers**
pull	Pull image or repo from docker registry
push	Push image or repo to docker registry
restart	Restart a running container
rm	**Remove one or more containers**
rmi	**Remove one or more images**
run	**Run a command in a new container**
save	Save an image to a tar archive
search	Search for an image in the docker index
start	Start a stopped container
stop	Stop a running container
tag	Tag an image into a repository
top	Lookup running processes of a container
unpause	Unpause a paused container
version	Show the docker version information
wait	Block and print exit code upon cont exit

rackspace®
the #1 managed cloud company

Example Docker Commands

- Get an interactive bash shell on a variety of linux distros
  - docker run -i -t ubuntu:latest bash
  - docker run -i -t centos:latest bash
  - docker run -i -t debian:latest bash
- Start a Debian container that runs an Apache server
  - docker run -d -p 80:80 –p 443:443 httpd:latest
- Start a Debian container running mysql that uses the /var/lib/mysql directory from the host
  - docker run --name db -p 3306:3306 -e MYSQL_ROOT_PASSWORD=1234.Rack4U2 -v /var/lib/mysql:/var/lib/mysql -d mysql

1.2

Lesson One – Part Two

Toolbox Overview

Docker Hub

- Public Repository for base images and community images
- Private Repositories for sale (get one for free per account)
  - Revenue model for Docker, Inc.
  - Requires authentication to view or update

Trusted Build System

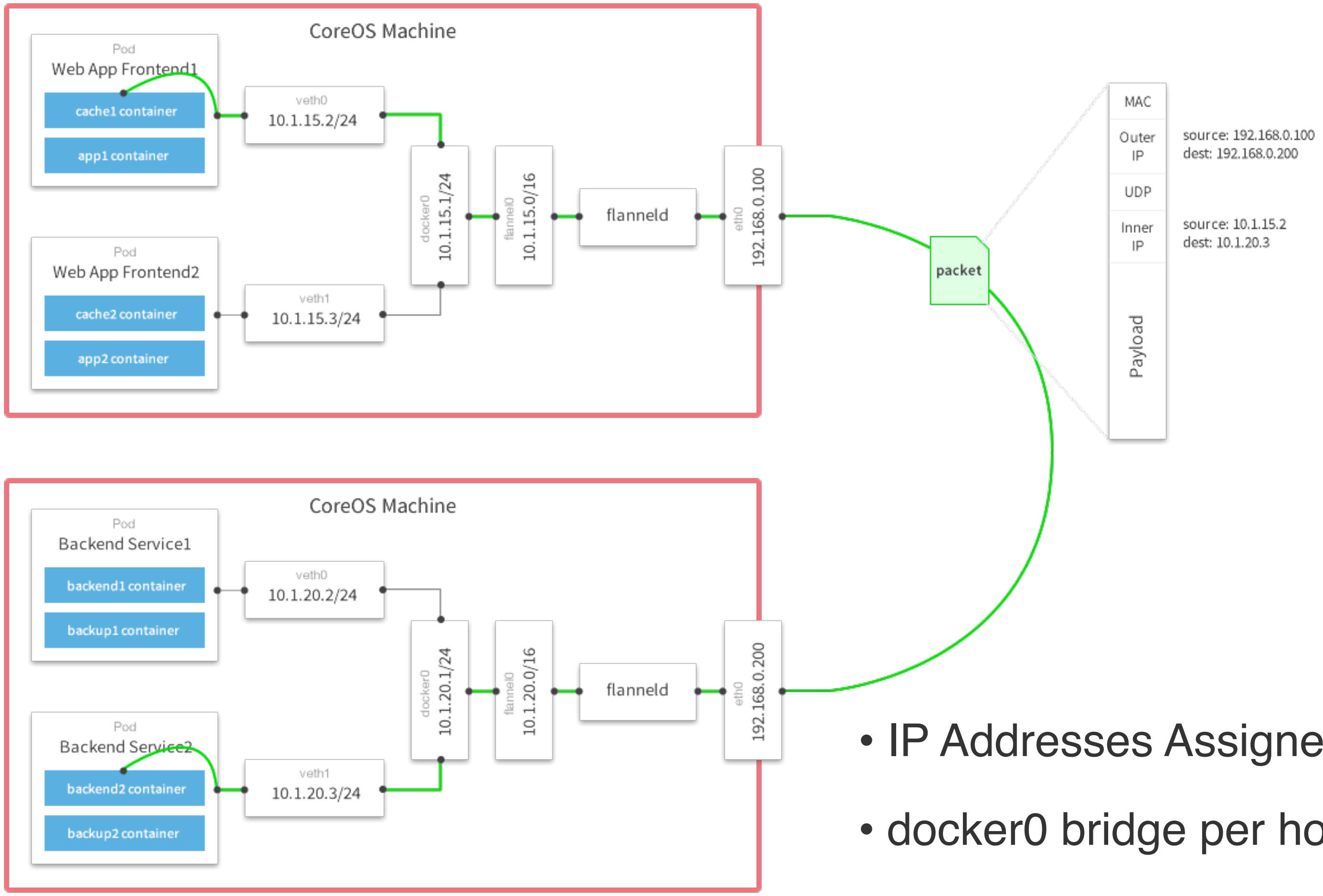
- Docker hosted service
- GitHub source code to docker image repository
- Uses Dockerfile from the GitHub repo

CoreOS

- Micro-OS
- Kernel + SSH + Docker + etcd + systemd
- Uses “fleet” to create clusters of CoreOS hosts
- Suitable for running microservices
- Automatic updates that require system level reboots

Weave and Flannel

- Both
  - Overlay network systems
    - Conceptually a way to build a VPN between your containers
  - Build a full mesh container-to-container network using VxLAN or UDP tunnels
- Weave
  - More mature than most of the container networking solutions
  - Expected to work well for deployments of < 200 containers
  - Simple user interface that behaves like “docker run”
- Flannel
  - Part of the CoreOS project, intended for use with Kubernetes
  - Each machine in the cluster is assigned a non-routable subnet.



- IP Addresses Assigned per pod
- docker0 bridge per host

Kubernetes

- Started by Google
- Opinionated Orchestration for Docker Containers
- Declarative expression of the desired end state
- Arranges services in pods
- Appropriate for most container use cases
- Hosts run a “minion” service that reports to the “master” which tracks the cluster state

How Docker works with OpenStack

- Nova-docker
  - Virt driver for Nova
  - Basic start and stop (basic VM lifecycle)
- Heat Resource
  - Create containers on Nova instances
  - Represent them in HOT files
  - No resource scheduler
  - Native management interface only
- Magnum
  - OpenStack Service for Containers as a first class resource
    - Integrates with Keystone
    - Integrates with Heat
    - Multi-Tenant
    - Asynchronous API
    - Concepts for Node, Bay, Pod, Container, Service, Replication Controller

LAB 1

Hands-On Lab 1

Introduction to Docker

LAB 1.0 – Get Ready

- Set your shell environment variables – Region value is case sensitive!

```
export OS_REGION_NAME=IAD
export OS_USERNAME=jdoe
export OS_API_KEY=278db780c7d943a1bc0b41ac359d01a8
```

LAB 1.1

Do This First

```
export OS_REGION_NAME=IAD
export OS_USERNAME=your_cloud_username
export OS_API_KEY=your_api_key_here
```

- Spin up two new Docker machines on Rackspace Cloud, switch between them.

```
$ docker-machine create -d rackspace machine1
$ docker-machine create -d rackspace machine2
$ docker-machine ls
$ docker-machine active machine1
```

- Get a shell on a Docker host

```
$ docker-machine ssh machine1
```

- Run ad-hoc containers on the fly for experimentation

```
$ docker run --rm -i -t ubuntu:latest bash
$ docker run --rm -i –t centos:latest bash
$ docker run --name one -d ubuntu:latest sleep 1d
$ docker run --name two –d centos:latest sleep 1d
$ docker ps
```

LAB 1.2 – Get a shell in a container without running sshd

- Get a shell in container “two” without running sshd:

```
$ docker exec -i —t two bash
$ uname —a
$ cat /etc/redhat-release
```

Notice the kernel belongs to the host?

Alternate Approach - nsenter

```
root@machine1:~# docker run --rm -v /usr/bin:/target
jpetazzo/nsenter

root@machine1:~# cat << "EOF" > enter.sh

PID=`docker inspect --format {{.State.Pid}} $1`
shift
nsenter --mount --uts --ipc --net --pid --target $PID $*
EOF
root@machine1:~# sh -x enter.sh one
```

LAB 1.3 – Move a container to another host

- Get a hub.docker.com account
  - Browse to https://hub.docker.com
  - Cick “Sign up with Github” black button (Provide your Github username+password)
  - Create a repo named “private” (Click “Add Repository” -> Repository)
    - Mark it as private, not public
- Move a container from one host to another.

```
$ $(docker-machine env machine1)
$ docker login
$ docker commit one your_username/private:2015-02-26
$ docker images
$ docker push your_username/private:2015-02-26
$ docker kill one; docker rm one; docker rmi your_username/private:2015-02-26
$ $(docker-machine env machine2)
$ docker login
$ docker run --name one -d your_username/private:2015-02-26
$ docker ps
```

Enter your Github username and password

Break

2

Lesson Two

Writing Dockerfiles

Dockerfile Example (review)

```
FROM centos:centos6
MAINTAINER aotto@aotto.com
RUN yum -y install openssh-server
EXPOSE 22
ADD start.sh /start.sh
CMD /start.sh
```

Dockerfile Directives

- **`FROM`**
  - The `name:tag` of the parent image that this image will be based on
- **`MAINTAINER`**
  - Optional line for documenting who does the care and feeding for this image. Typically an email address or name.
- **`RUN <command>`**
  - A command to run in the container to adjust something on this image. This runs at the time of the `docker build` command.
- **`EXPOSE`**
  - What TCP ports should be exposed on the host when `docker run -P` is used
- **`ADD <src>... <dest>`**
  - Add files from the `src` directory in the context of the build to the container filesystem at `dest`
- **`CMD command param1 param2`**
  - Unless otherwise specified by the `docker run` command, run this command when the container is started with this image

Build an image from a Dockerfile

- Create a new directory, and enter it

```
$ mkdir build

$ cd build
```

- Create a file named Dockerfile in that directory
- Note: Each line of the Dockerfile is executed in a separate intermediate container, followed by an automatic commit
- Build the image

```
$ docker build -t <name>:<tag> .
```

Demonstration

- Copying files into containers using bind mounts
- Copying files out of containers using `docker cp`

demo

Using Dockerfile and build rather than commit

- Commit stores the current state of the container filesystem
- Dockerfiles give us a repeatable way to create a container image

Best Practice: docker build

LAB 2

Hands-On Lab 2

Writing Dockerfiles

LAB 2.1 – Writing your own Dockerfile

- Fork and/or clone the demo repo

```
git clone https://github.com/adrianotto/dockerfile_demo.git
```

- Edit the Dockerfile to change something
  - Maybe add a new file of your own or change what packages are installed
- Run the build.sh script

```
$ sh –x build.sh
```

- Notice the docker build command that runs
- Now add an additional line to the Dockerfile after the existing yum install line:

```
RUN yum -y install vim
```

- Run the build script again
  - Notice that the previous commands are skipped, and cached output is used!

Break

3

Lesson Three

Linking and Networking Containers

Understanding Links

- Networks namespaces between containers can be bridged together by Docker
- A “Link” is a network path between two containers on the same host
- Syntax:

```
docker run -d --name my_db training/postgres
docker run -d --name web --link my_db:db -p 80:80 training/webapp python app.py
```

- This creates a pair of containers that can connect to each other
- The /etc/hosts file on the ‘web’ container will have an entry for a host named ‘db’
- Processes running inside the ‘web’ container can access the database at the hostname ‘db’

Understanding Overlay Networks

- Overlay networks are like a VPN between containers
- Typical transports are UDP tunneling and VxLAN.
  - Weave uses UDP tunneling
  - Flannel supports both UDP tunneling and VxLAN
- Gives appearance of containers on separate hosts are on the same physical network segment.
- Performance will be reasonable in most cases, but slower than without an overlay network.

Weave Example https://github.com/zettio/weave/releases

```
$ docker-machine ssh machine1
root@machine1:~# cd /usr/bin && wget https://github.com/zettio/weave/releases/download/v0.9.0/weave && chmod +x weave
root@machine1:/usr/bin# weave launch
root@machine1:/usr/bin# weave run 10.0.1.1/24 --name weave1 -d ubuntu sleep 1d
root@machine1:/usr/bin# exit
$ docker-machine ls
NAME       ACTIVE   DRIVER      STATE     URL                          SWARM
machine1            rackspace   Running   tcp://162.242.239.109:2376
machine2   *        rackspace   Running   tcp://162.242.243.251:2376
$ docker-machine ssh machine2
root@machine2:~# cd /usr/bin; wget https://github.com/zettio/weave/releases/download/v0.9.0/weave && chmod +x weave
root@machine2:/usr/bin# weave launch 162.242.239.109
root@machine2:/usr/bin# weave run 10.0.1.2/24 --name weave2 -d ubuntu sleep 1d
root@machine2:/usr/bin# exit
$ docker exec -i -t weave2 bash
root@587cd615d122:/# ping -c 4 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.048 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.030/0.040/0.049/0.009 ms
```

LAB 3

Hands-On Lab 3

Container Linking

LAB 3.1 – Linking Containers

- First, create a container with a database in it:

```
docker run -d --name my_db training/postgres
```

- Now create a second container that connects to the first using a link:

```
docker run -d --name web --link my_db:db -p 80:5000 training/webapp python app.py
```

- Connect to the web container

```
docker exec -i -t web bash
root@7f4a56aed86d:/opt/webapp# grep db /etc/hosts
172.17.0.4      db
root@7f4a56aed86d:/opt/webapp# apt-get install inetutils-ping
root@7f4a56aed86d:/opt/webapp# ping —c 4 db
PING db (172.17.0.4): 48 data bytes
56 bytes from 172.17.0.4: icmp_seq=0 ttl=64 time=0.084 ms
56 bytes from 172.17.0.4: icmp_seq=1 ttl=64 time=0.087 ms
56 bytes from 172.17.0.4: icmp_seq=2 ttl=64 time=0.089 ms
56 bytes from 172.17.0.4: icmp_seq=3 ttl=64 time=0.087 ms
--- db ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.084/0.087/0.089/0.000 ms
root@7f4a56aed86d:/opt/webapp#
```

LAB 3.2 – Networking Containers

```
$ docker-machine ssh machine1
root@machine1:~# cd /usr/bin && wget https://github.com/zettio/weave/releases/download/v0.9.0/weave && chmod +x weave
root@machine1:/usr/bin# weave launch
root@machine1:/usr/bin# weave run 10.0.1.1/24 --name weave1 -d ubuntu sleep 1d
root@machine1:/usr/bin# exit
$ docker-machine ip machine1
162.242.239.109
$ docker-machine ssh machine2
root@machine2:~# cd /usr/bin; wget https://github.com/zettio/weave/releases/download/v0.9.0/weave && chmod +x weave
root@machine2:/usr/bin# weave launch 162.242.239.109
root@machine2:/usr/bin# weave run 10.0.1.2/24 --name weave2 -d ubuntu sleep 1d
root@machine2:/usr/bin# exit
$(docker-machine env machine2)
$ docker exec -i -t weave2 bash
root@587cd615d122:/# ping -c 4 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.048 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.030/0.040/0.049/0.009 ms
```

Summary

- LESSON 1 - PART 1 Intro to Docker
  - Docker, Cgroups, Kernel Namespaces, and Docker Images
- LESSON 1 - PART 2 – Toolbox Overview
  - Docker Hub, Trusted build system, Weave, Flannel, etcd, CoreOS, Magnum, Kubernetes, and how Docker works with OpenStack
- Hands-on LAB 1 - Using docker-machine
  - Spin up Docker containers on Rackspace Cloud, Switch between docker hosts.
  - How to run ad-hoc containers on the fly for experimentation, How to get a shell on a Docker host, and a Docker container without running sshd in the container.
  - How to move a container from one host to another.
- LESSON 2 – Writing Dockerfiles
  - Dockerfile Directives
  - Demonstration of copying files into containers using bind mounts, and copying files out of containers using “docker cp”.
  - Why Dockerfiles are better than customized images made manually and stored with commit.
- Hands-on LAB 2 - Writing Dockerfiles.
  - Saving a Dockerfile in a source repo, and using a build script for configuration injection.
- LESSON 3 - Linking and Networking containers
  - Demonstration of how to use links to communicate between containers on the same host. Using weave to communicate between containers on separate hosts.
- Hands on LAB 3 - Linking containers
  - Container linking on the same host, and mesh networking between containers on different hosts.

Credits

- Adrian Otto
- Simon Jakesch
- Thomas Maddox
- Ash Wilson
- Don Schenck

THANK YOU

rackspace®

the #1 **managed cloud** company

RACKSPACE® | 1 FANATICAL PLACE, CITY OF WINDCREST | SAN ANTONIO, TX 78218
US SALES: 1-800-961-2888 | **US SUPPORT:** 1-800-961-4454 | WWW.**RACKSPACE**.COM