Chameleon Cloud Tutorial

National Science Foundation

Program Solicitation # NSF 13-602

CISE Research Infrastructure: Mid-Scale Infrastructure - NSFCloud (CRI: NSFCloud)

Docker - Machine, Compose, and Swarm

Because of incompatibilities, part of this tutorial uses Rackspace instead of Chameleon. See the Machine section for details.

This tutorial will cover using Docker Machine, Compose and Swarm. Ultimately these tools are intended to be used together but because they're not yet mature that synthesis is limited. We'll discuss the limitations in more detail throughout the tutorial. We'll instead focus on using each tool individually and demonstrate them together in ways that currently work.

Compose

Compose simplifies the process of arranging and linking containers together. Compose lets us specify the links and runtime configurations of containers in a single config file, rather than having several lengthy commands to execute in the right sequence. In the first tutorial we setup containers on 2 different hosts and linked them together to run a simple webpage. In this tutorial we will set up a similar page that lets you post messages and lists those previously posted. It uses 3 containers and we'll arrange them with Compose.

Machine

Machine allows us to create Docker hosts and control them without interacting with the host machines directly. This way you don't have to SSH to machines running the Docker daemon to run containers. Chameleon won't work for this part of the tutorial because of problems with Chameleon's lease system. Support for Chameleon will likely happen in the future. See this issue (https://github.com/docker/machine/issues/1461) on their GitHub. You could also use virtual machines running on a Chameleon instance but we ran into issues installing VirtualBox on the default Chameleon CentOS image. So for now we're going to demo Machine with Rackspace to give you an idea of its potential. **We will be controlling everything from a Chameleon machine however.**

Swarm

Swarm is used to group multiple Docker hosts together so that containers or groups of containers can scale across machines. We'll also be demoing this on Rackspace because we use Machine to setup our Swarm.

Prerequisites

It’s expected that you have gone throughDocker Tutorial 1 (http://cloudandbigdatalab.github.io/docs/Chameleon%20Cloud%20Tutorial%20-%20Docker%20Fundamentals.pdf) or are already familiar with its content. No more prior knowledge is required past the first tutorial.

Steps Outline

The whole tutorial (barring problems) will probably take 45 mins to an hour. It can take a long time to update your Chameleon instance and creating hosts with Machine can take a few minutes per host. How long your hosts take to create depends on the type and provider.

#	Task	Approximate Time (mins)
1	Setup	20
2	Compose	5
3	Machine	10
4	Swarm	20

Setup

We’ll be using the default Chameleon CentOS image for this tutorial.

```
sudo yum update -y

sudo yum install -y docker

sudo groupadd docker

sudo usermod -a -G docker cc

sudo systemctl start docker.service
```

We also created a user group `docker` and added the default `cc` user to it before starting the Docker daemon. **After logging out and back in you will no longer have to use sudo with the Docker client or tools.**

Then follow these instructions to install Machine (https://docs.docker.com/machine/#installation) and Compose (https://docs.docker.com/compose/install/). **If you’re getting “Permission Denied” using curl, run `sudo -i` to become root, run the commands, then `exit` .**

If you’re going to try to use Machine with Rackspace, VM’s, or another provider follow they’re docs to get setup. It’s fairly easy to complete the demo with VM’s on your own physical machine.

Compose

With Compose you outline your container configuration and arrangement with a YAML file name docker-compose.yml. Our docker-compose.yml (https://github.com/cloudandbigdatalab/tutorial-cham-docker-2/blob/master/docker-compose.yml) is on our GitHub. This lays out the 3 container composition. In our docker-compose.yml we specify to pull out images from Docker Hub. All the resources, including the Dockerfile, to build these images is available on our GitHub (https://github.com/cloudandbigdatalab/tutorial-cham-docker-2). If you wanted to build the images yourself or make modifications, download the repo then change

```
image: cloudandbigdatalab/server:tutorial-2
```

to

```
build: ./server
```

to build and use a local image. We're assuming the Dockerfile for server is in the server folder within the current directory. You would do the same for the page container. Note for the db container we're using the unmodified Postgres image off Docker Hub so their isn't a folder for it. Here's a quick explanation of what's going on with our composition.

Container Name	Apps	Description
server	Nginx	handles http requests
page	uWSGI and Django	uWSGI connects Nginx to Django, Django generates the html
db	Postgres	database for page, Django connects to Postgres

Run the Composition

```
docker-compose -p tutorial up -d
```

-p tutorial specifies our project name. Otherwise it uses the name of the current directory. If the images had been changed and we wanted to run the updated versions we would run

```
docker-compose pull
docker-compose -p tutorial up -d
```

and the images would be pulled and our containers restarted.

Check your running containers.

```
docker-compose -p tutorial ps
```

The output should look similar to this.

```
Name                 Command              State            Ports
-------------------------------------------------------------------------------------
tutorial_db_1       /docker-entrypoint.sh postgres   Up      5432/tcp
tutorial_page_1     ./startup.sh                     Up      3031/tcp
tutorial_server_1   nginx -g daemon off;             Up      443/tcp, 0.0.0.0:80->80/tcp
```

Now if you visit the ip of your Chameleon machine in the browser you should see the page running.

Machine

So now we're going to do the same thing but we're going to run our composition on a Docker host we setup with Machine. As we outlined in the introduciton we can't use Machine to create hosts on Chameleon (or VM's) so we're using Rackspace.

Create a host

We have our account information in environment variables in this example. -d rackspace specifies the *driver* as Rackspace. This will take several minutes.

```
docker-machine create -d rackspace docker-main
```

Point Docker at Remote Machine

```
eval "$(docker-machine env docker-main)"
```

Now if we run `docker ps` the 3 containers our gone because we're looking at the remote host.

Run Composition on Remote Host

The commands are exactly the same as before.

Run composition.

```
docker-compose -p tutorial up -d
```

Check our running containers.

```
docker-compose -p tutorial ps
```

To see the ip of our remote machine.

```
docker-machine ip docker-main
```

Then if you visit the ip in the browser you should see the same page as before. Note that the top left string on the page is the id of the page container. It will be different from before.

Swarm

As noted in the introduction we'll be using Rackspace for this part of the tutorial as well. It is possible to manually setup a Swarm cluster of Chameleon Docker hosts but we won't be doing that here. We'll be using Machine which simplifies the process.

Our Composition

For this demo we can't really use the multi-container setup we used earlier. This is for two reasons:

1. Currently linked containers must be run on the same host. This defeats the point of Swarm. Docker's networking is being overhauled to allow cross-host links and the feature is available in experimental builds. We were unable to get it working at the time of this writing however.
2. Even with cross-host linking, there's no automatic proxying or load balancing. So if for example we scaled the page container to 10, that's easy enough. But we'd also have to configure Nginx to load balance between those containers. Or we could have a proxy container in between the two. This is all possible but again we didn't get it working at the time of this writing. This is something you must build into your app design, there's no automatic mechanisms for this as of yet.

We're still using an (extremely sparse) docker-compose.yml (https://github.com/cloudandbigdatalab/tutorial-cham-docker-2/blob/master/swarm/docker-compose.yml) for this. It consists of one service / container that runs folding@home (https://folding.stanford.edu). We're going to run it and scale it across a few nodes.

Generate Swarm Token

We're generating the token and saving to an environment variable.

export SWARM_TOKEN=$(docker run swarm create)

Swarm Master

Again the account information needed for Rackspace is stored in environment variables. Creating the machine will a few minutes.

```
docker-machine create -d rackspace --swarm --swarm-master \
  --swarm-discovery=token://$SWARM_TOKEN docker-swarm-master
```

Swarm Nodes

Here we're using a bash loop to create 2 nodes.

```
for ((i = 0; i < 2; i++)); do \
  docker-machine create -d rackspace --swarm \
    --swarm-discovery=token://$SWARM_TOKEN docker-swarm-node-$i; \
done
```

Point Docker at Swarm

Now we're going to point the Docker client at our Swarm cluster.

```
eval "$(docker-machine env --swarm docker-swarm-master)"
```

We can see info about the swarm with

```
docker info
```

which should output something like this.

```
Containers: 4
Images: 3
Storage Driver:
Role: primary
Strategy: spread
Filters: affinity, health, constraint, port, dependency
Nodes: 3
 swarm-master: 104.130.134.163:2376
  └ Containers: 2
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 1.014 GiB
  └ Labels: executiondriver=native-0.2, kernelversion=3.13.0-37-generic, operatingsystem=Ubuntu
14.04.1 LTS, provider=rackspace, storagedriver=aufs
 swarm-node-0: 104.130.134.175:2376
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 1.014 GiB
  └ Labels: executiondriver=native-0.2, kernelversion=3.13.0-37-generic, operatingsystem=Ubuntu
14.04.1 LTS, provider=rackspace, storagedriver=aufs
 swarm-node-1: 104.130.134.76:2376
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 1.014 GiB
  └ Labels: executiondriver=native-0.2, kernelversion=3.13.0-37-generic, operatingsystem=Ubuntu
14.04.1 LTS, provider=rackspace, storagedriver=aufs
Execution Driver:
Kernel Version:
Operating System:
CPUs: 3
Total Memory: 3.041 GiB
Name:
ID:
Http Proxy:
Https Proxy:
No Proxy:
```

Run Composition

Note that you need to download the docker-compose.yml into a different directory from earlier and run Compose from there.

```
docker-compose -p tutorial up -d
```

If we run

```
docker-compose -p tutorial ps
```

and look at the output we see a single worker container running.

```
Name                  Command              State    Ports
-------------------------------------------------------------------
tutorial_worker_1   /bin/sh -c /etc/init.d/FAH ...   Up
```

We can scale our *worker* service to 6.

```
docker-compose -p tutorial scale worker=6
```

Now if we run `docker-compose -p tutorial ps` again and look at the output we should see multiple worker containers running.

```
Name                 Command             State   Ports
-------------------------------------------------------------
tutorial_worker_1   /bin/sh -c /etc/init.d/FAH ...   Up
tutorial_worker_2   /bin/sh -c /etc/init.d/FAH ...   Up
tutorial_worker_3   /bin/sh -c /etc/init.d/FAH ...   Up
tutorial_worker_4   /bin/sh -c /etc/init.d/FAH ...   Up
tutorial_worker_5   /bin/sh -c /etc/init.d/FAH ...   Up
tutorial_worker_6   /bin/sh -c /etc/init.d/FAH ...   Up
```

If we run `docker ps` we can look at the `NAMES` field and see that our containers our spread across the 3 hosts in our cluster.

```
CONTAINER ID        IMAGE                       COMMAND             CREATED             STATUS              POR
TS                  NAMES
faadba6dff79        jordan0day/folding-at-home   "/bin/sh -c '/etc/in   About a minute ago   Up About
a minute                  swarm-master/tutorial_worker_6
3457647206b0        jordan0day/folding-at-home   "/bin/sh -c '/etc/in   About a minute ago   Up Abou
t a minute                swarm-node-1/tutorial_worker_5
97daf03f52c2        jordan0day/folding-at-home   "/bin/sh -c '/etc/in   About a minute ago   Up About
a minute                  swarm-node-0/tutorial_worker_4
fd381b18544e        jordan0day/folding-at-home   "/bin/sh -c '/etc/in   About a minute ago   Up About
a minute                  swarm-master/tutorial_worker_3
c2edd0380540        jordan0day/folding-at-home   "/bin/sh -c '/etc/in   About a minute ago   Up Abou
t a minute                swarm-node-1/tutorial_worker_2
8ddadc49ec72        jordan0day/folding-at-home   "/bin/sh -c '/etc/in   2 minutes ago       Up 2 minut
es                        swarm-node-0/tutorial_worker_1
```

Cross-Provider Swarm

You can also setup a Swarm cluster across different providers. For example we could have launched one of our containers on Digital Ocean with:

```
docker-machine create -d digitalocean --swarm \
  --swarm-discovery=token://$SWARM_TOKEN docker-swarm-node-<i>;
```

and have a mixed cluster. In testing this worked just as well as if when they were on the same provider.

Conclusion

Docker intends for Compose, Machine, and Swarm to work together to enable simple yet powerful workflows. The experience of putting this tutorial together shows that’s not reality today. However, Compose and Machine work pretty well on their own barring Machine’s Chameleon incompatibility. The synthesis between Compose and Machine is also solid right now. Swarm is problematic and not as useful as one might initially think. But Docker does disclaim that these tools are not production ready yet. In the future they should work better for multi-container apps and services.