Chameleon Cloud Tutorial

National Science Foundation

Program Solicitation # NSF 13-602

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

Docker - Fundamentals

In this tutorial we're going to guide you through the fundamentals of using Docker on Chameleon Cloud. You should already be familiar with managing resources on Chameleon Cloud, if not follow the "Getting Started" tutorial. At the end of this tutorial you will have setup a demo website utilizing 5 Docker containers and 2 physical hosts.

Prerequisites

It's expected that you have a general knowledge of Linux command-line environments, though most of the steps can be copied exactly without modification. No previous knowledge of Docker is required and we will provide some explanation of key Docker terms and concepts below. See the official Docker docs (https://docs.docker.com/) for more detail and reference.

What is Docker?

Docker is conceptually similar to virtual machines but has much less resource overheard because it doesn't run a full guest OS. Docker containers start in seconds vs minutes, take up less space, and are less hardware demanding because they share resources with the host OS. Read in-depth here (https://www.docker.com/whatisdocker).

Terms & Concepts

Most of the docker descriptions are taken directly from theirglossary (https://docs.docker.com/reference/glossary/).

Docker Engine or "Docker": The docker daemon process running on the host which manages images and containers

Image: Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.

Docker Hub: The Docker Hub is a centralized resource for working with Docker and its components. Like GitHub but for Docker images.

Container: A container is a runtime instance of a docker image. A Docker container consists of:

- A Docker image
- Execution environment
- A standard set of instructions
- The concept is borrowed from Shipping Containers, which define a standard to ship goods globally. Docker defines a standard to ship software.

Dockerfile: A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile. Dockerfile Reference (https://docs.docker.com/reference/builder/)

Postgres: An SQL database. Official Site (http://www.postgresql.org)

Nginx “engine x”: A web server. Official Site (http://nginx.com)

uWGSI: An application server that connects to Nginx. In our tutorial we’re using it to run a simple Python app that generates the demo page. Official Site (https://uwsgi-docs.readthedocs.org/en/latest/)

Steps Outline

#	Description	Time (mins)
1	Spin up Chameleon resources	10
2	Software installation	10
3	Setup app and ambassador containers, Postgres on host 1, Nginx and uWSGI on host 2	10
4	Test demo site to see if configuration was successful	1

1. Chameleon Resources

Create 2 Chameleon baremetal servers. We used a CentOS 7 image for this tutorial but feel free to use any other Distro as long it runs Docker.

2. Software Installation

Install Docker on each server with `sudo yum install docker` . This installs the Docker daemon and client tools. You may also wish to install an editor such as vim and git (if not already installed, included in our CentOS image).

Important
The Docker daemon needs to be running before you can use Docker. Start it with `sudo service docker start` . **If you’re getting errors with every Docker command this may be the cause.**

2. Container Setup

Before you move on let’s explain some things. You will be setting up one host with a Postgres (SQL database) container. The other host will be setup with Nginx (web server) and uWGSI (interface to Python script that generates actual page) containers. To connect the uWGSI

container across hosts to the Postgres container we will use *ambassador* containers, one on each host.

Note
You have two options to deploy the containers. You can pull already built containers from our Docker Hub (https://hub.docker.com/u/cloudandbigdatalab/) repos and run them. Or you can pull this GitHub repo and build the Docker images yourself using the Dockerfile in each directory. If you want to edit the site content you will need to build the images yourself after making your edits, although you can edit the database by simply connecting to it. The ambassador containers we're using are maintained by a Docker employee and thus we'll only be pulling those. You can pull an image before running it with `sudo docker pull image_name` or you can just `sudo docker run --name container_name -d image_name` and Docker will automatically pull the image for you.

Useful Commands

```
# show running containers
sudo docker ps

# show all, even stopped, containers
sudo docker ps -a

# check container's logs (stdout of container)
# useful if there's a problem
sudo docker logs container_name

# to stop container
sudo docker stop container_name_or_id

# to remove container
sudo docker rm container_name_or_id

# to remove and stop together
sudo docker rm -f container_name_or_id

# to remove image
sudo docker rmi image_name_or_id
```

Host 1

Pulling from Docker Hub

```
# start postgres container
# port 5432 is set to be exposed in Dockerfile
# -d run as daemon (run in background)
# user: cloudandbigdatalab, repo: postgres
sudo docker run --name postgres -d cloudandbigdatalab/postgres

# start ambassador container, linking to postgres
# -p map port 5432 from within container to outside
sudo docker run --name host2_ambassador -d --link postgres:postgres -p 5432:5432 svendowideit/
ambassador
```

Building from Dockerfile

```
# clone repo
git clone https://github.com/cloudandbigdatalab/tutorial-cham-docker-1.git

# move into postgres directory
cd postgres

# build postgres image
# -t to name
# . is path to Dockerfile
sudo docker build -t postgres .

# from here you run the same commands as if you pulled the images
# EXCEPT change cloudandbigdatalab/image_name to image_name
```

Host 2

Pulling from Docker Hub

Note
You will need to replace the ip in one of the following commands. If you need to restart the uwsgi container you will need to rm both the uwsgi and nginx containers then run again. The uwsgi container must be run first.

```
# start ambassador container
# --expose port 5432 (postgres default) to linking containers
# -e sets environment variable for postgres, shared to linked containers
# replace host1_local_ip with local ip of your host 1 instance
sudo docker run --name host1_ambassador -d --expose 5432 -e POSTGRES_PORT_5432_TCP=tcp://
host1_local_ip:5432 svendowideit/ambassador

# start uwsgi container, linking to host1_ambassador
sudo docker run --name uwsgi -d --link host1_ambassador:postgres cloudandbigdatalab/uwsgi

# start nginx container, linking to uwsgi container
# map port 80 to outside, http default port
sudo docker run --name nginx -d --link uwsgi:uwsgi -p 80:80 cloudandbigdatalab/nginx
```

Building from Dockerfile

```
# clone repo
git clone https://github.com/cloudandbigdatalab/tutorial-cham-docker-1.git

# move into uwsgi directory
cd uwsgi

# build uwsgi image
# -t to name
# . is path to Dockerfile
sudo docker build -t uwsgi .

# move into nginx directory
cd ../nginx

# build nginx image
sudo docker build -t nginx .

# from here you run the same commands as if you pulled the images
# EXCEPT change cloudandbigdatalab/image_name to image_name
```

Test Website

Visit the public ip of your host 2 instance in your browser. If it worked congratulations!