DataStax Monday Learning

DataStax Monday Learning

Upgrade yourself, unlock new skills

- Every Week
- Best Instructors
- Most Important Topics
- From Engineers to Engineers
- Absolutely Free

DataStax Monday Learning

Docker Containers

From Basics to Best Practices

4-weeks Learning Path: 28.09-23.10.2020

Speakers:

- Aleks Volochnev
- Developer Advocates of DataStax

Schedule:

- **Week I 28.09.2020** Docker Fundamentals I
- **Week II 05.10.2020** Docker Fundamentals II
- **Week III 12.10.2020** Application Development with Docker
- **Week IV 19.10.2020** Best Practices + Final Assignment

DataStax Monday Learning

Docker Containers

From Basics to Best Practices

Your Goals:

- Learn a New Powerful Technology
- Become a better Developer

Your Weekly Tasks:

- Attend the Live Class or Watch the Recording
- Answer the Quiz Questions
- Do the Assignment

By the End of the Course:

- Do the Final Assignment
- Get your “Docker Upgrade Complete” Achievement Unlocked!

DATASTAX®
DataStax Monday Learning

Docker Containers
From Basics to Best Practices

Week I

Docker Fundamentals I

4 Questions to tune the Class

WHAT IS DOCKER?

What is Docker?

"Write Once, Run Anywhere"
Sun Microsystems about Java, 1995

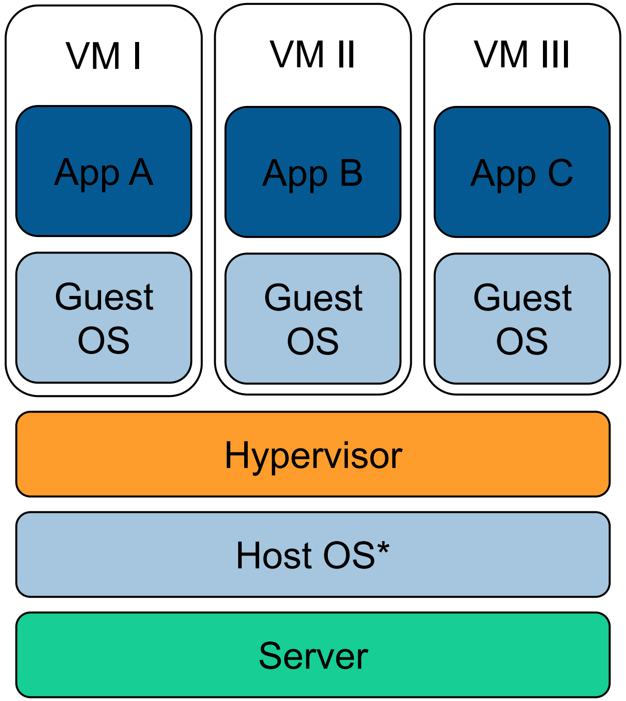
"Build Once, Run Everywhere"
Docker about any piece of software, 2013

In short, docker is a set of tools to **build**, **ship**, and **run any software with all its dependencies at ease**. It allows you to pack the code, its binaries, dependencies, whatever it may need into a reusable immutable unit called "*Image*" and then run this image as a "*Container*" on any docker-ready system.

What is Docker? **Virtualization?**

Docker Containers are often **confused** with *Virtual Machines*, but this is a mistake. Despite some similarities in ideas, workflows and use-cases, those are two very different technologies providing different features.

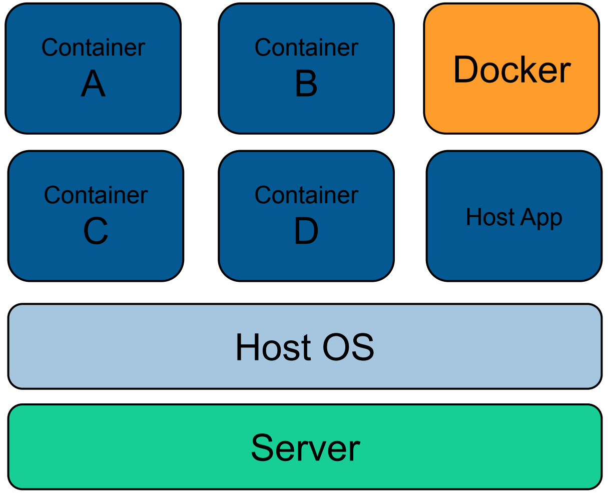
Virtual Machines are implemented using **Virtualisation**. Usually, they are much more heavy-weight, deliver better isolation from the host system and bring their own operating system.



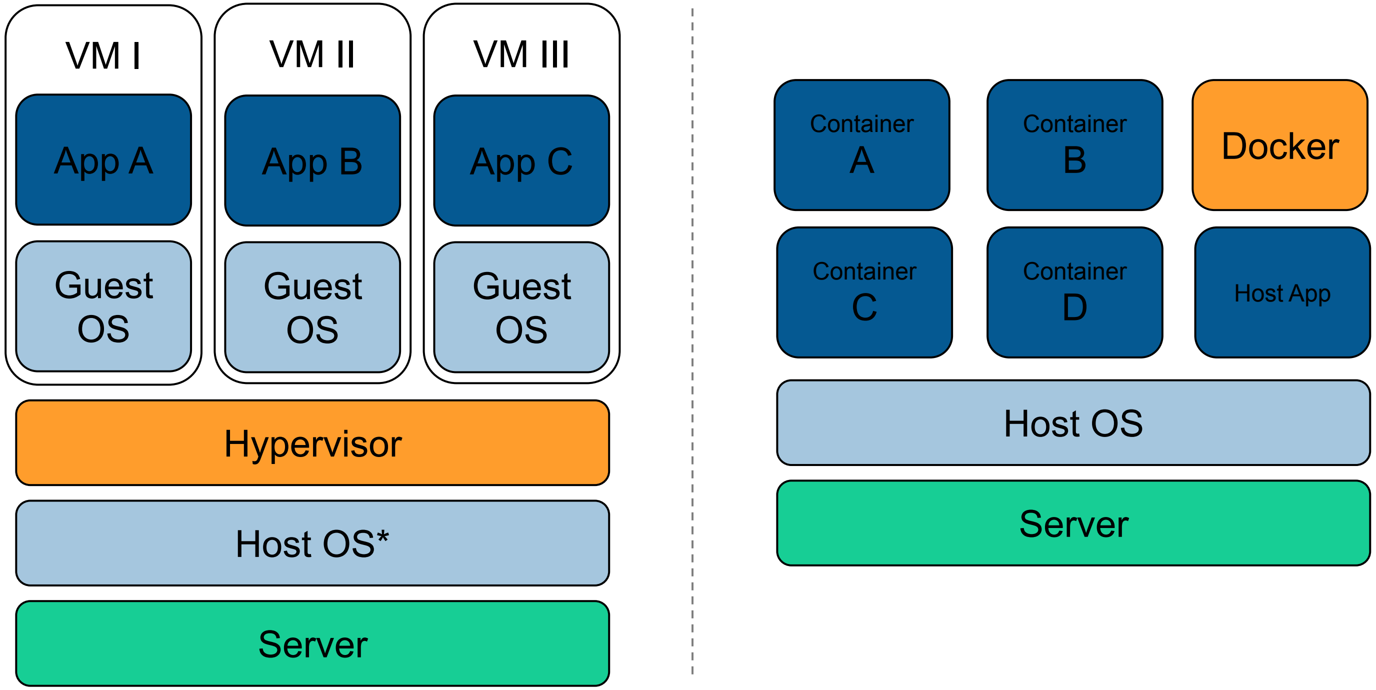
* Depends on infrastructure, may be optional

What is Docker?
Containerisation!

In comparison with virtual machines, docker containers are much more light-weight, as they don't use their own operating system and Hypervisor. Originally based on LXC (LinuX Containers), docker runs its Containers directly as the processes of the Host OS, making it much faster to initialise (no OS initialisation required) and much easier to manage at the cost of less isolation from the Host OS.



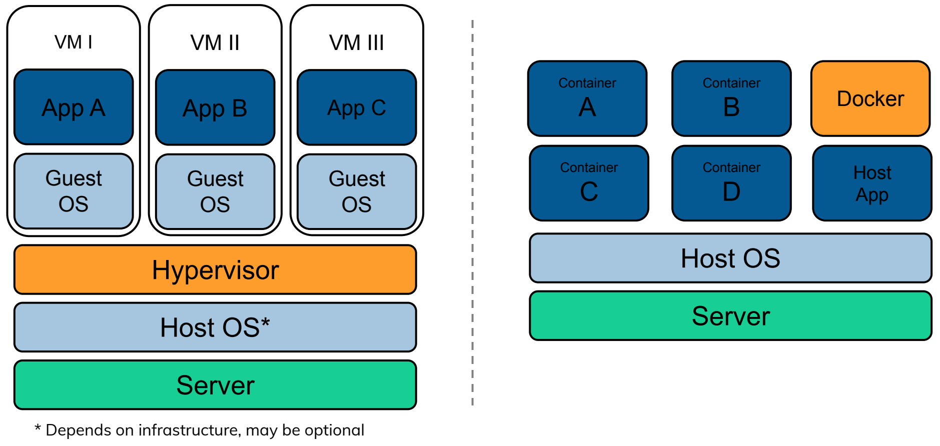
Containerisation vs Virtualisation



* Depends on infrastructure, may be optional

Containerisation vs Virtualisation

Is Virtualisation dead? No way! Is Containerisation better? For some use-cases yes. Remember, it's always about getting a proper tool for the particular job!

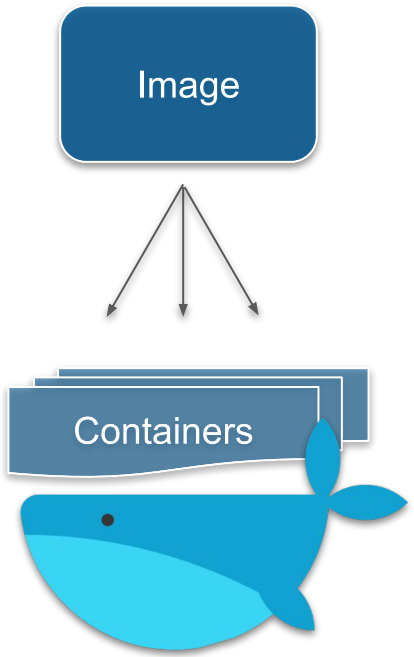


HOW DOCKER WORKS?

How Docker Works?

As you remember, docker allows to build, ship and run any software. At first, we will only **build** and **run**, so there are only two base concepts to learn: **an Image** and a **Container**.

The workflow is very simple: we build Images with docker and use them to run Containers.



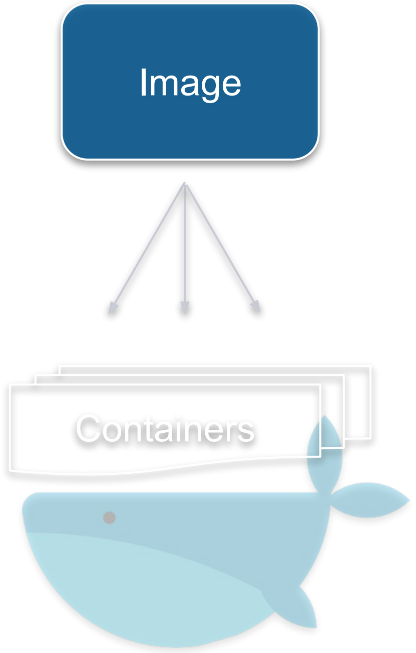
What is an Image?

The Image is the most important thing in the docker world. The Image is an immutable read-only template for Containers. First we build the Image, then we use it to launch Containers as we need them. The Docker Image is a static thing, not a process in memory.

The only three things you can do with the Image:

- Build an Image
- Launch a Container using Image
- Delete an Image

If you are into Object-Oriented Programming, you may consider Image as an OOP Class.

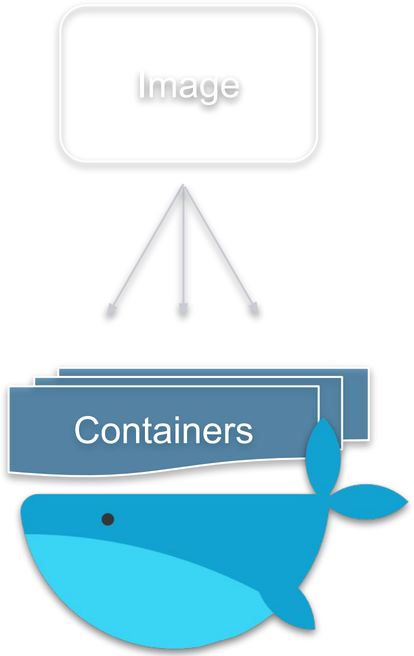


What is a Container?

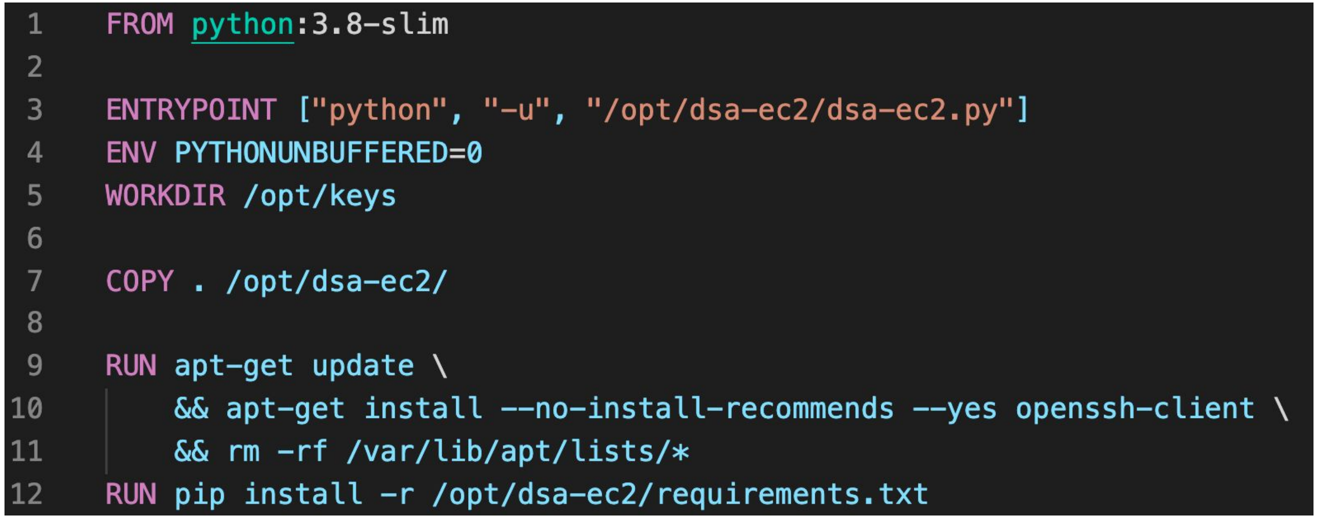
The Container is an instantiated Image, a process (or multiple processes) running in memory. As the Container is a running entity, you have more things to do:

- Stop the Container
- Start the Container
- Run a command into the Container
- Watch the logs
- etc.

If you are into Object-Oriented Programming, you may consider Container as an instance of the OOP Class, an OOP Object.



How to build an Image?



```dockerfile
FROM python:3.8-slim

ENTRYPOINT ["python", "-u", "/opt/dsa-ec2/dsa-ec2.py"]
ENV PYTHONUNBUFFERED=0
WORKDIR /opt/keys

COPY . /opt/dsa-ec2/

RUN apt-get update \
    && apt-get install --no-install-recommends --yes openssh-client \
    && rm -rf /var/lib/apt/lists/*
RUN pip install -r /opt/dsa-ec2/requirements.txt
```

Dockerfile

The easiest way to build an Image is the Dockerfile. Dockerfile is the set of instructions explaining to docker what to do to build the image.

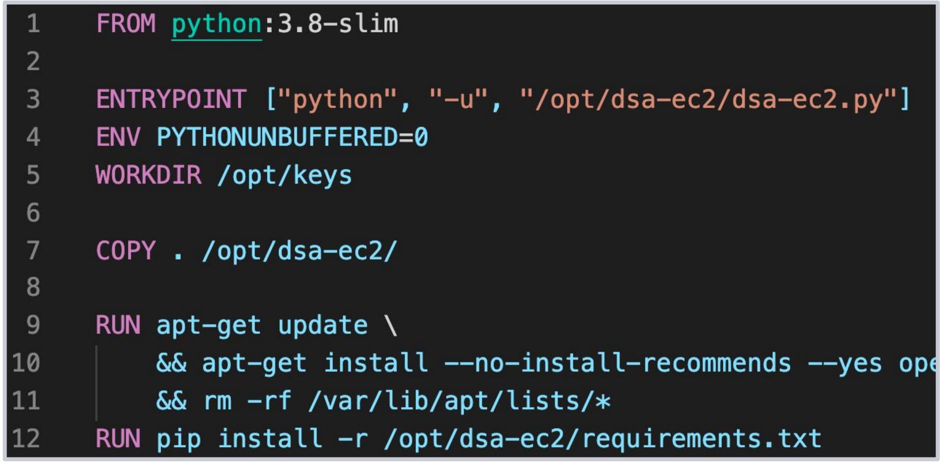
How to build an Image?

Dockerfile Instructions

There are 4 main instructions (although there are more to cover later).

- FROM
- COPY
- RUN
- CMD

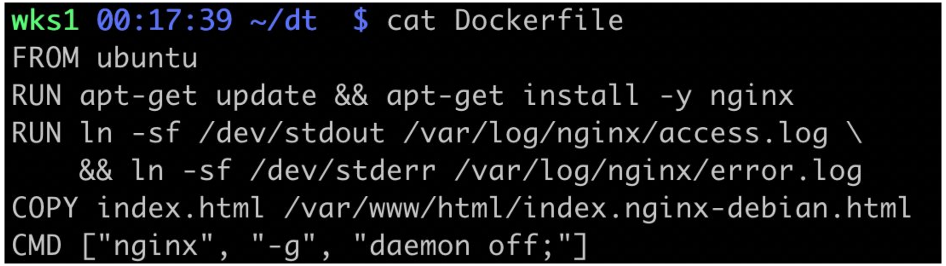
Docker executes the instructions you give - boom - and you have the image!



```
FROM python:3.8-slim

ENTRYPOINT ["python", "-u", "/opt/dsa-ec2/dsa-ec2.py"]
ENV PYTHONUNBUFFERED=0
WORKDIR /opt/keys

COPY . /opt/dsa-ec2/

RUN apt-get update \
    && apt-get install --no-install-recommends --yes ope
    && rm -rf /var/lib/apt/lists/*
RUN pip install -r /opt/dsa-ec2/requirements.txt
```

TALK IS CHEAP...
SHOW ME THE CODE.

Live Demo I

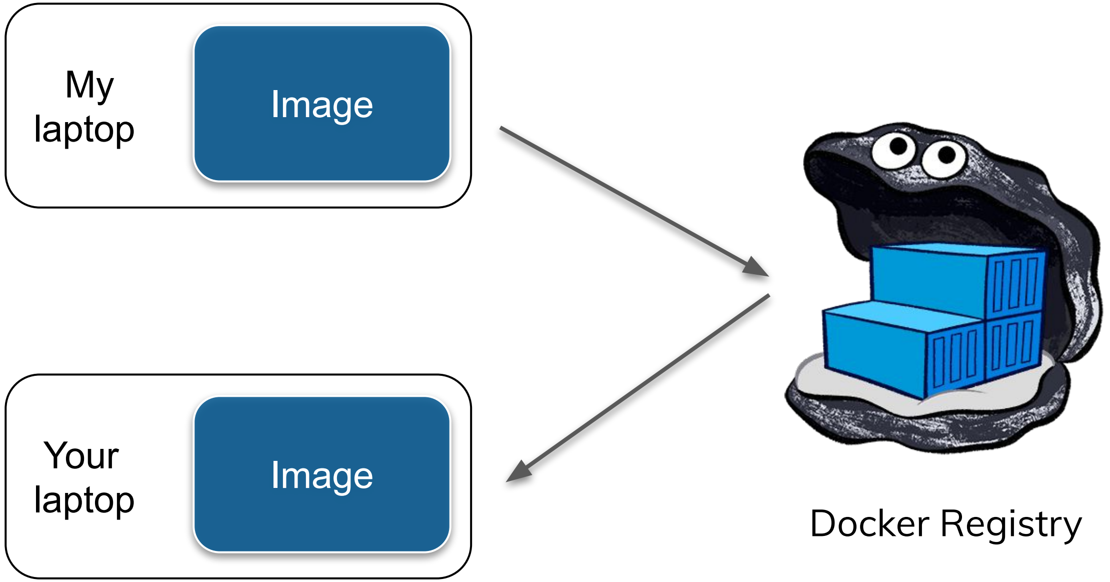

```
wks1 00:17:39 ~/dt  $ cat Dockerfile
FROM ubuntu
RUN apt-get update && apt-get install -y nginx
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log
COPY index.html /var/www/html/index.nginx-debian.html
CMD ["nginx", "-g", "daemon off;"]
```

- **Dockerfile**
  - from
  - copy
  - run
  - cmd
- **docker build**
  - default
- **docker images**
- **docker run**
  - default
  - --publish
  - --detached
- **docker ps**
- **docker stop**
- **docker rm**
- **docker image rm**

How to ship Images?

To transfer Images, we need to know how to push and pull images. Besides that, we will need a place to store images - the docker images repository.

There are many of them, and you can run your own, but the main one is the **hub.docker.com**, the default repository for docker images.

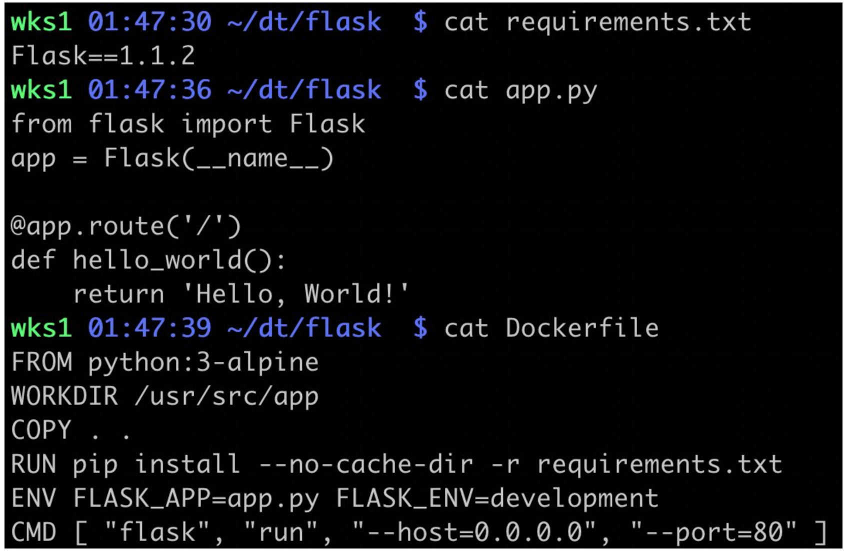


Live Demo II

Docker Registry

- docker pull
- docker tag
- docker login
- docker push

That’s cool but how about real applications?



```
wks1 01:47:30 ~/dt/flask  $ cat requirements.txt
Flask==1.1.2
wks1 01:47:36 ~/dt/flask  $ cat app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
wks1 01:47:39 ~/dt/flask  $ cat Dockerfile
FROM python:3-alpine
WORKDIR /usr/src/app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
ENV FLASK_APP=app.py FLASK_ENV=development
CMD [ "flask", "run", "--host=0.0.0.0", "--port=80" ]
```

Live Demo III

- Source Code
- Configuration
- Dependencies
- --env
- -it exec
- docker logs
- docker build --tag

Flask

LIVE QUIZ!

Week I Assignment

Week I Assignment

Very simple assignment this time!

1. Create an account at hub.docker.com
2. Install Docker **or** make yourself familiar with labs.play-with-docker.com
3. Containerize a **SIMPLE** hello-world application in the language/framework of your choice. Don't take a complex system but **prefer a simple one,** like the Flask example we did today. It's the week I assignment after all, we will cover more advanced scenarios next week!
4. Ensure you used every command from today's class - both for Dockerfile and command line.
5. Invite a friend or a colleague! If you are at this slide with us, it means you liked the class, don't you? It's week I so it's still the perfect time to jump in!

Resources:

- https://github.com/datastaxdevs/docker-learning-path
- https://discord.gg/va4vnsm

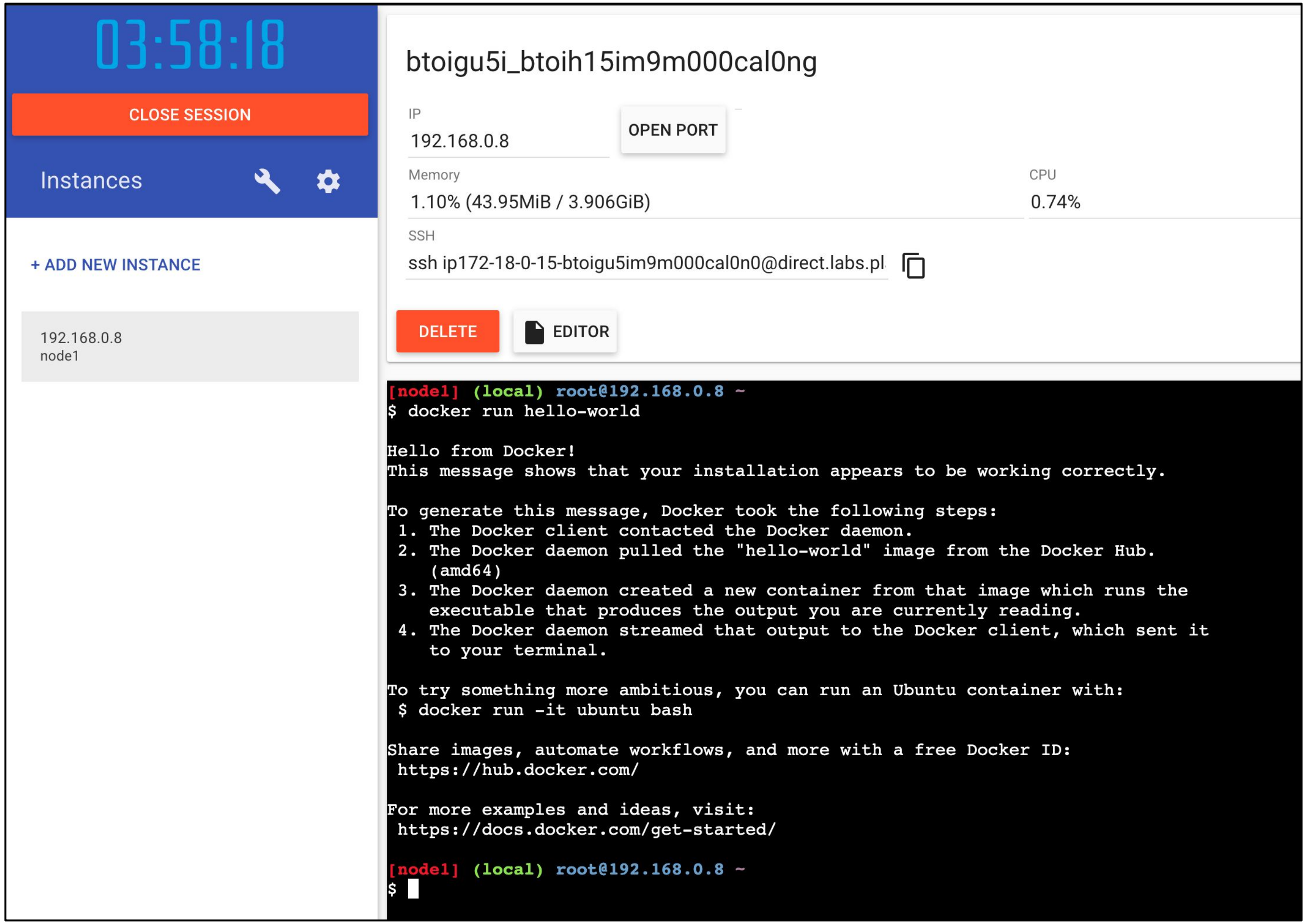
Docker

Docker Installation:

docs.docker.com/get-docker

If it doesn't work for you, go for the web-based sandbox:

labs.play-with-docker.com



btoigu5i_btoih15im9m000cal0ng

IP
192.168.0.8

Memory
1.10% (43.95MiB / 3.906GiB)

CPU
0.74%

SSH
ssh ip172-18-0-15-btoigu5im9m000cal0n0@direct.labs.pl

```
[node1] (local) root@192.168.0.8 ~
$ 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/

[node1] (local) root@192.168.0.8 ~
$
```

DATASTAX®

Thank You!

See You Next Week!