Getting Started with Docker

Contents

1. Introduction to Containerization

- What is containerization?
- Containers vs. virtual machines
- Docker editions

What is Containerization?

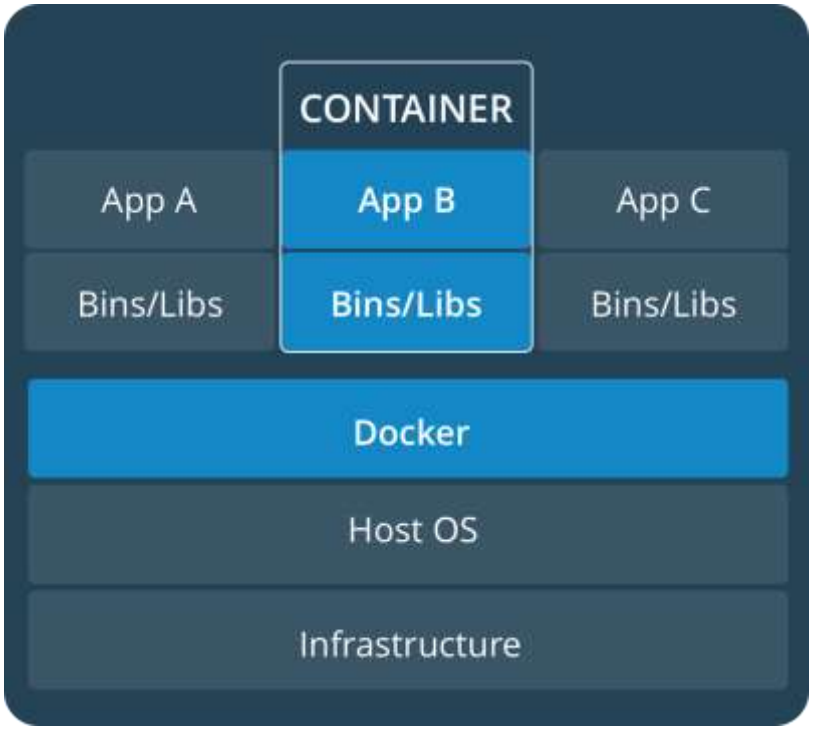
- Containerization is a way of wrapping an application, plus its environment, into a shrink-wrapped container
  - Makes it easy to deploy and run the application, because it runs in a virtualized environment

- Docker is a very popular containerization tool
  - You define an "image" that contains your app, properties, etc.
  - You then run the image - a running image is called a "container"

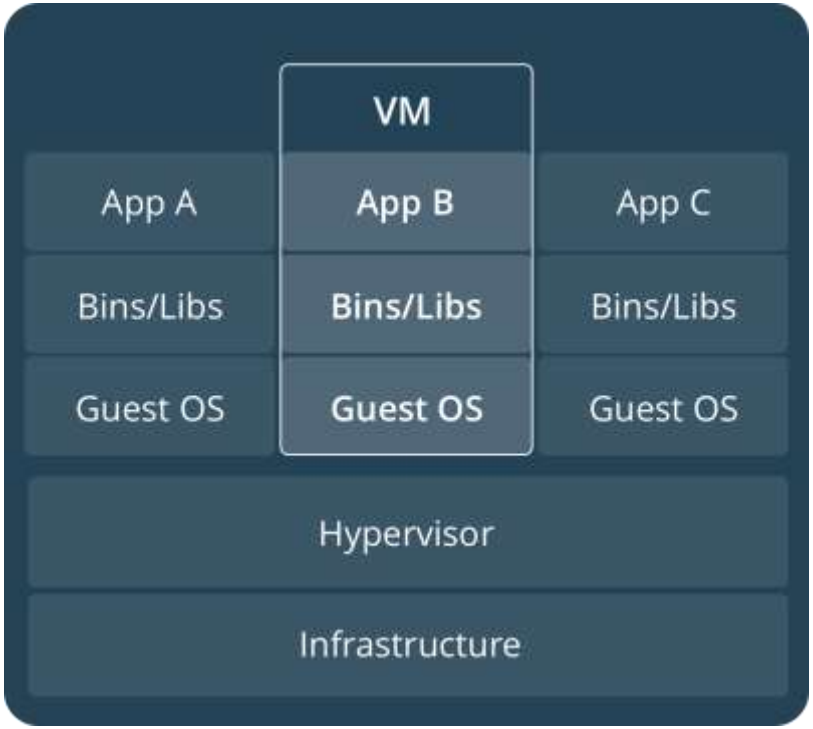
Containers vs. Virtual Machines

- Containers are much more lightweight that VMs
  - Containers run on top of the host OS, e.g. Linux
  - VMs are much bulkier because they actually contain a guest OS

Docker containers



Virtual machines



Docker Editions

- Docker comes in two editions
  - Docker Community Edition (CE)
  - Docker Enterprise Edition (EE)

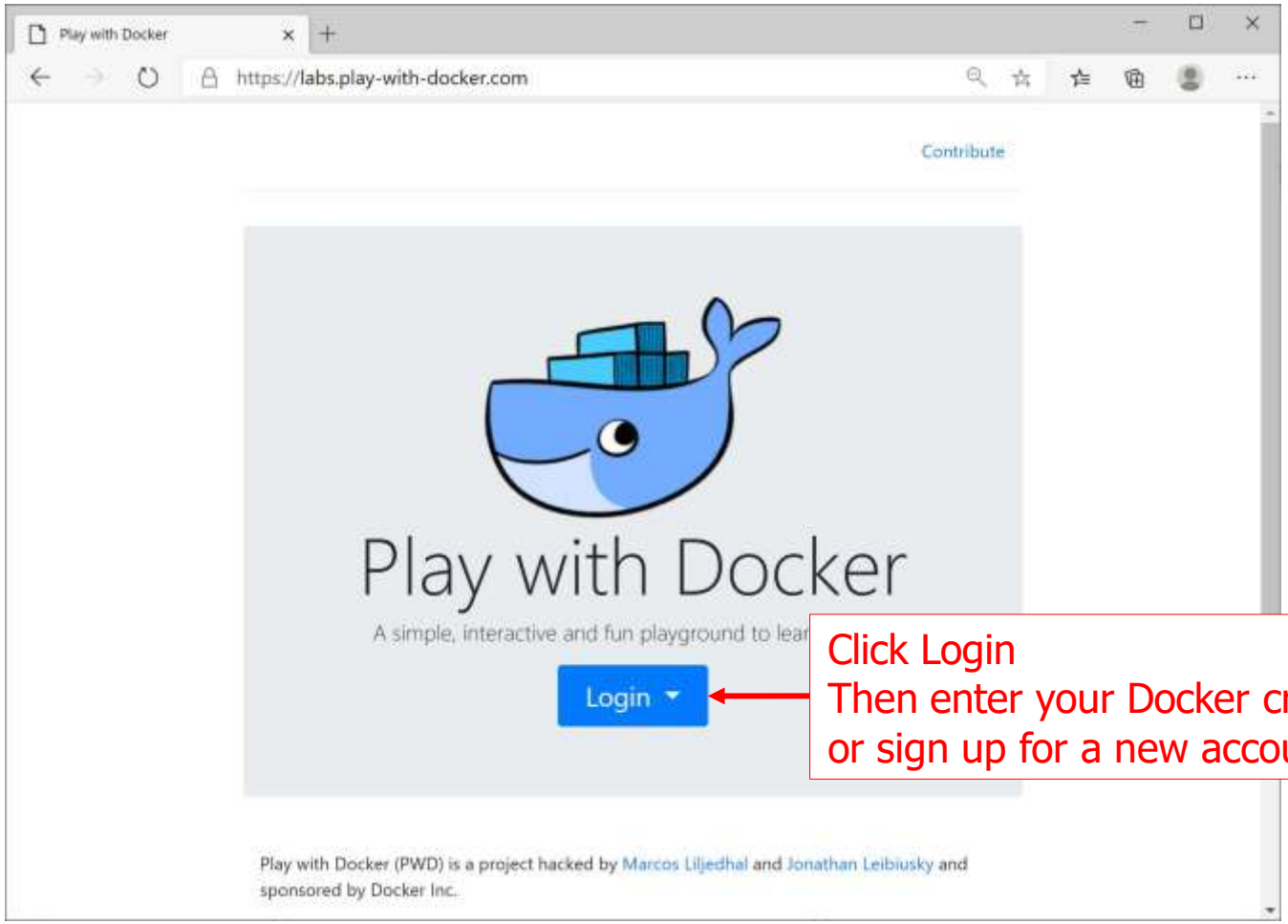
- You can install Docker on various platforms
  - Docker for Linux
  - Docker for Windows
  - Docker for Mac

2. Using Docker during this course

- Using 'Play with Docker'
- Starting your 'Play with Docker' instance
- Running Docker in your Linux machine

Using 'Play with Docker' (1 of 2)

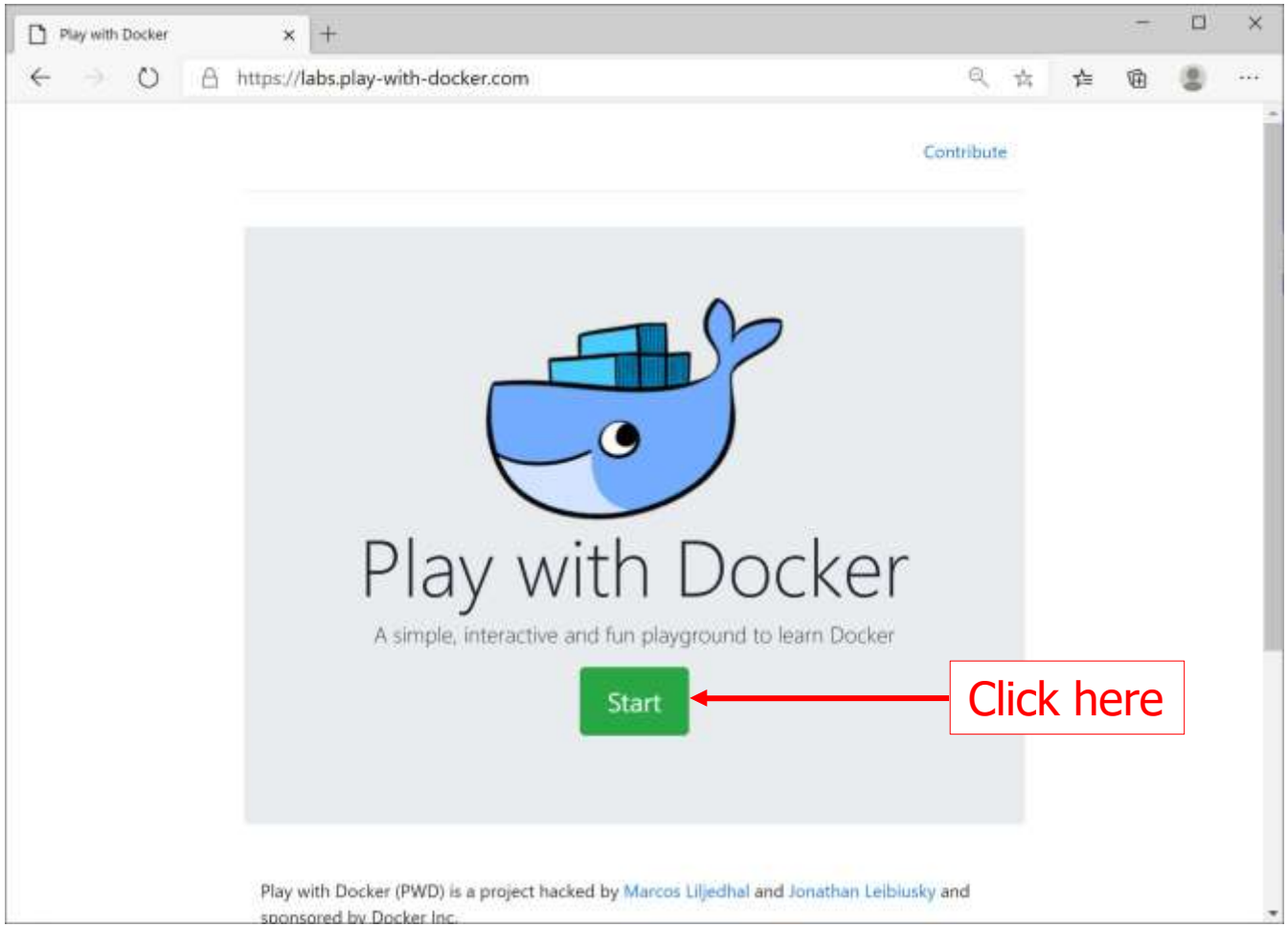
- We'll use 'Play with Docker', an online Docker environment
  - https://labs.play-with-docker.com/



Click Login
Then enter your Docker credentials
or sign up for a new account

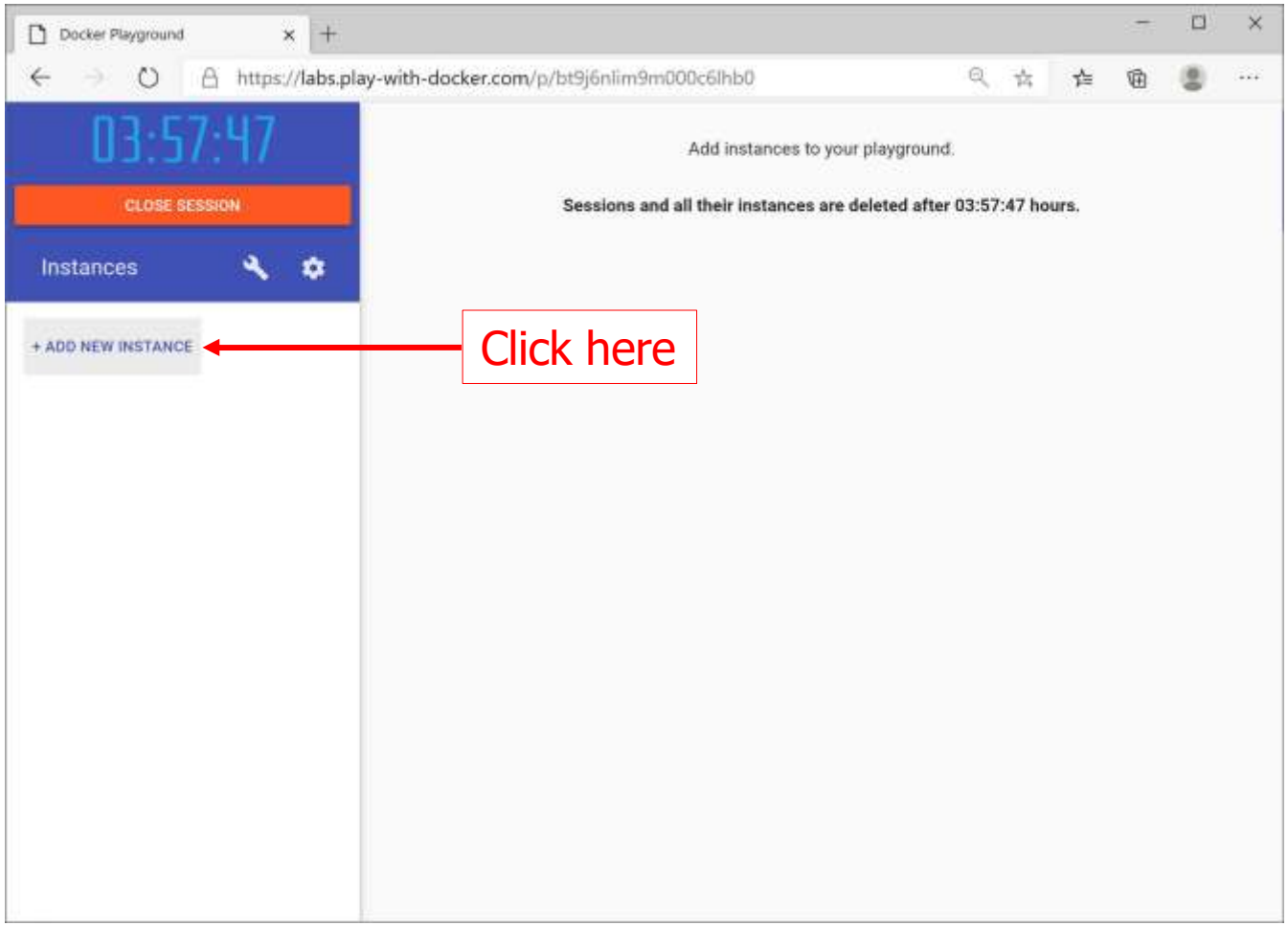
Using 'Play with Docker' (2 of 2)

- You're now ready to start your 'Play with Docker' session



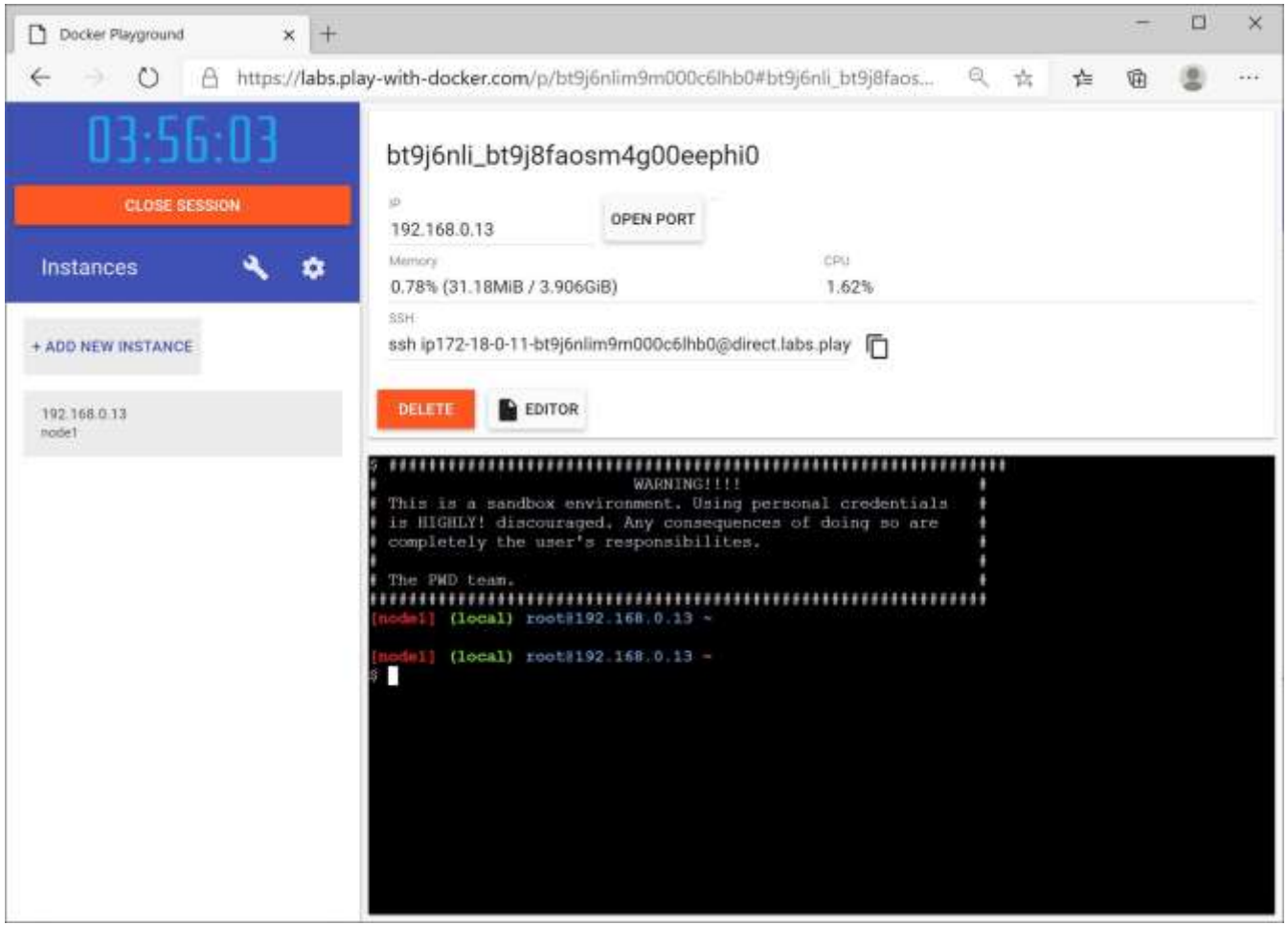
Starting your 'Play with Docker' Instance (1 of 2)

- 'Play with Docker' gives you a 4-hour session
  - To start a Docker instance, click ADD NEW INSTANCE



Starting your 'Play with Docker' Instance (2 of 2)

- The 'Play with Docker' environment is an Alpine Linux VM in the cloud, with the Docker tools installed



Running Docker in your Linux machine

- You can run Docker in your Linux machine, as follows

```
docker --version
```

```
$ ##############################################################################
#                                WARNING!!!!                                   #
# This is a sandbox environment. Using personal credentials                    #
# is HIGHLY! discouraged. Any consequences of doing so are                     #
# completely the user's responsibilites.                                       #
#                                                                              #
# The PWD team.                                                                #
##############################################################################
[node1] (local) root@192.168.0.13 ~

[node1] (local) root@192.168.0.13 ~
$ docker --version
Docker version 19.03.11, build 42e35e61f3
[node1] (local) root@192.168.0.13 ~
$
```

- For more info about the `docker` command, see:
  - https://docs.docker.com/engine/reference/commandline/docker/

3. Understanding Docker Images

- Overview
- Images vs. containers
- Running a sample image
- Listing images in the local Docker registry

Overview

- A Docker image is a "black box" executable package
  - It includes everything needed to run an application

- E.g. a Docker image for a Java micro-service might have:
  - A JVM
  - A web server (e.g. Tomcat)
  - Any additional JARs necessary, e.g. database drivers
  - A JAR containing your REST service

- In this section we're going to see how to download ("pull") and run a pre-built image from Docker Hub

Images vs. Containers

- When you run a Docker image…
  - Docker creates an in-memory instance of the image
  - This in-memory instance is called a "container"
  - You can run many container instances for an image, if you like

- Docker is a Linux technology
  - Docker containers run inside of Linux

Running a Sample Image (1 of 3)

- Docker has a sample pre-built image called "hello-world"
  - You can run it as follows:

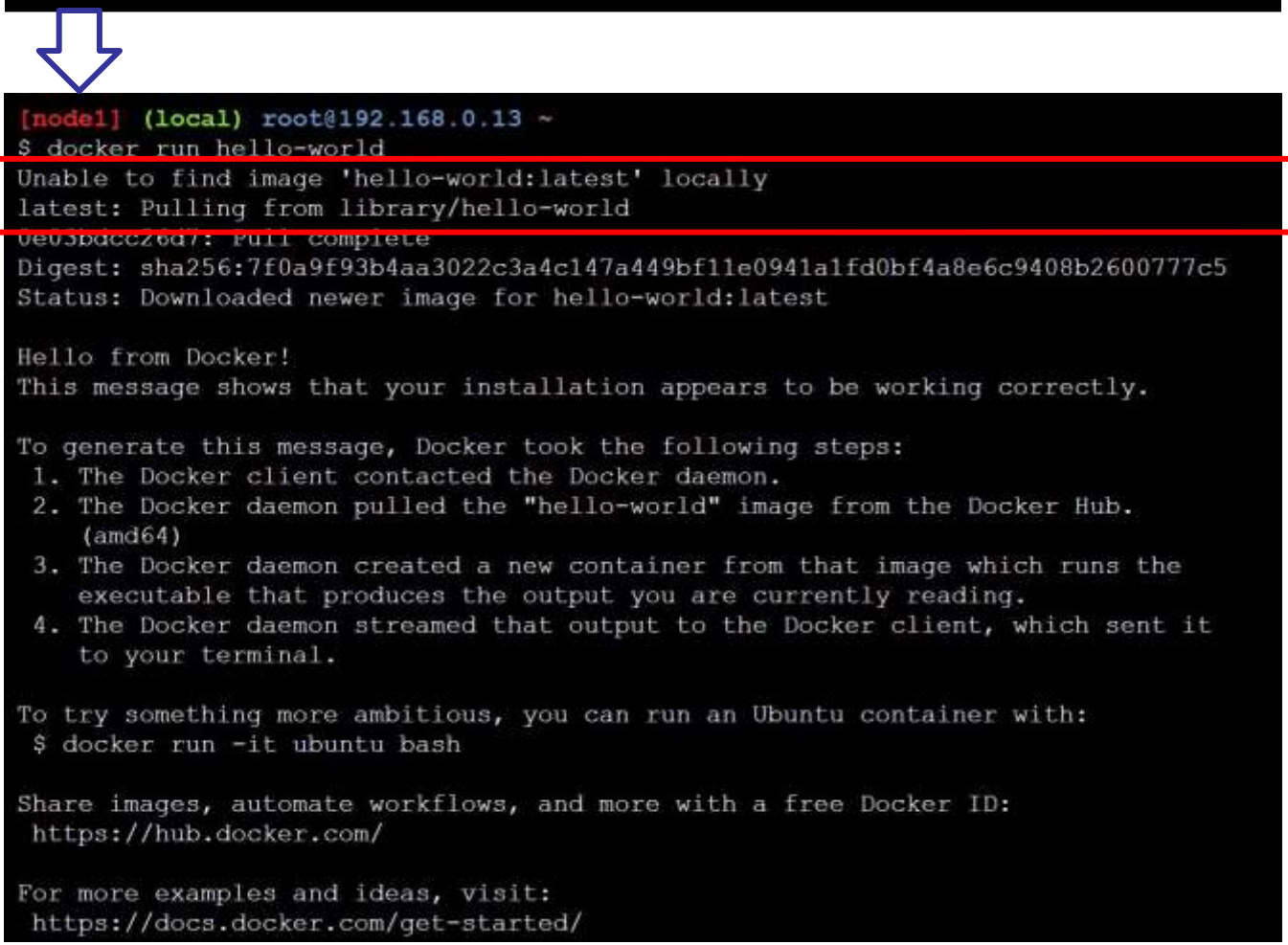
```
docker run hello-world
```

- Docker looks for the image in the local registry
  - The default location for images is /var/lib/docker
- If the image isn't in the local registry…
  - Docker issues a "pull" request, to download it from a global registry (Docker Hub)
  - Docker stores the downloaded image in the local registry
- Docker then runs the image
  - i.e. it creates a container, a running instance of the image

Running a Sample Image (2 of 3)

- Here's a screenshot - note the "pull" request near the top

```
docker run hello-world
```



Running a Sample Image (3 of 3)

- Do another Docker run, and see what happens
  - Note there's no "pull" request this time – why not…?

```
docker run hello-world
```

```
[node1] (local) root@192.168.0.13 ~
$ 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/
```

Listing Images in the Local Docker Registry

- You can get a list of all the Docker images in your local Docker registry, as follows:

```
docker image ls
```

```
[node1] (local) root@192.168.0.13 ~
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              bf756fb1ae65        8 months ago        13.3kB
[node1] (local) root@192.168.0.13 ~
$
```

4. A Closer Look at Images and Containers

- The power of containerization
- Creating multiple containers from an image
- Creating containers in detached mode
- Accessing port 8080 on the Linux box
- Listing containers
- Stopping a container

The Power of Containerization (1 of 2)

- Docker Hub contains thousands of really useful images, providing containerized shrink-wrapped functionality
  - E.g. Tomcat, MySQL, MongoDB, etc.

- E.g. run this command to download and run Tomcat

```
docker run -d -p 8123:8080 tomcat
```

- This downloads the Tomcat image into your local registry, and creates an instance of the image (i.e. a container)
  - Tomcat runs inside the container (the -d option runs the container in "detached" mode)
  - Tomcat listens on port 8080 inside the container (the -p option maps Tomcat's port 8080 to your machine's port 8123)

The Power of Containerization (2 of 2)

- You can ping Tomcat from your host computer,
  - E.g. via curl

```
curl localhost:8123
```

- This sends an HTTP request to `localhost:8123`
  - This is mapped to port 8080 in your Docker container
  - Tomcat is listening on port 8080 in your Docker container
  - Tomcat returns an HTML "welcome" page

```
[node1] (local) root@192.168.0.13 ~
$ curl localhost:8123
<!doctype html><html lang="en"><head><title>HTTP Status 404 – Not Found</title><style type
="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;backgro
und-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font
-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:none;}</st
yle></head><body><h1>HTTP Status 404 – Not Found</h1><hr class="line" /><p><b>Type</b> Sta
tus Report</p><p><b>Description</b> The origin server did not find a current representatio
n for the target resource or is not willing to disclose that one exists.</p><hr class="lin
e" /><h3>Apache Tomcat/9.0.37</h3></body></html>[node1] (local) root@192.168.0.13 ~
$
```

Creating Multiple Containers from an Image

- You can easily spin up another Tomcat container
  - Tomcat will run on port 8080 within that container
  - You must map it to a different port in your Linux machine

```
docker run -d -p 8246:8080 tomcat
```

- You can then ping it as follows:

```
curl localhost:8246
```

Accessing Port 8080 on the Linux Box (1 of 2)

- 'Play with Docker' lets you open ports on your Linux box

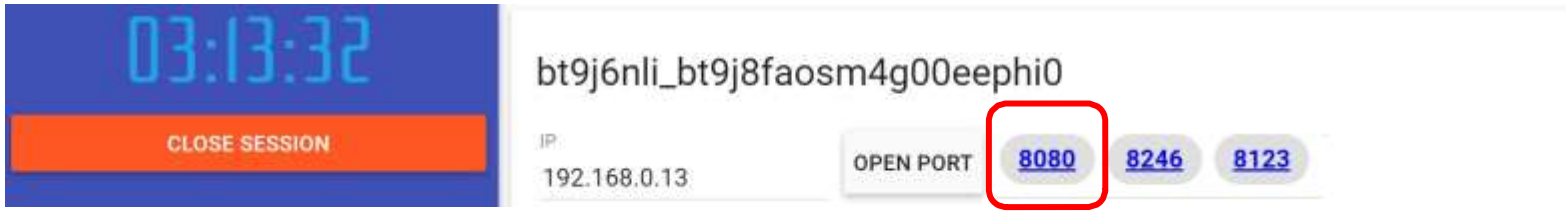
- This lets you access the Linux box from other machines

- To prove the point, start another Tomcat container and map its port 8080 to the Linux box's port 8080

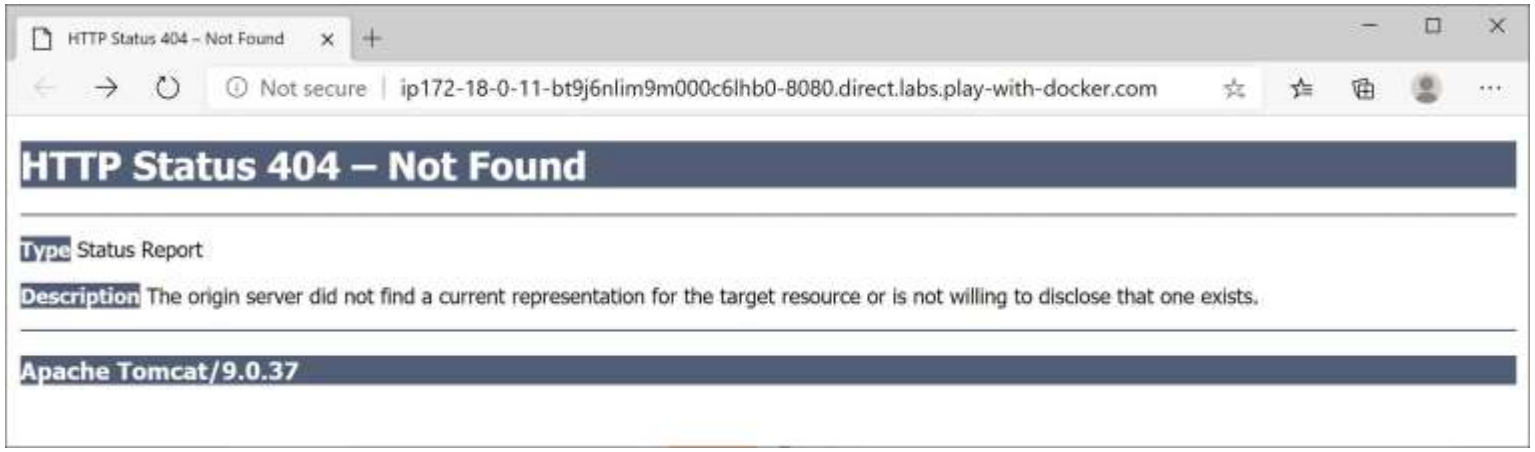
```
docker run -d -p 8080:8080 tomcat
```

Accessing Port 8080 on the Linux Box (2 of 2)

- Then click 8080, to open this port on your PWD Linux box



- A browser window opens and sends an HTTP GET request to port 8080 on your PWD Linux box



Listing Containers

- You can get a list of all the containers currently running

```
docker container ls
```

```
[node1] (local) root@192.168.0.13 ~
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAM
ES
681c9bb761b9        tomcat              "catalina.sh run"   48 minutes ago      Up 48 minutes       0.0.0.0:8080->8080/tcp   inf
allible_mcclintock
e666b1eb1cb9        tomcat              "catalina.sh run"   50 minutes ago      Up 50 minutes       0.0.0.0:8246->8080/tcp   sil
ly_blackwell
2f32330e62c9        tomcat              "catalina.sh run"   52 minutes ago      Up 52 minutes       0.0.0.0:8123->8080/tcp   ang
ry_ardinghelli
[node1] (local) root@192.168.0.13 ~
$
```

- Each container has:
  - A unique container id (abbreviated)
  - The name of the image (of which this container is an instance)
  - The command that is executed within the container
  - Created timestamp and status
  - Port mappings
  - A name for the container (random name by default)

Stopping a Container

- To stop a container, run the following command with the container ID you want to stop
  - Either specify the container ID or its name

```
docker stop relaxed_bassi
```

- If you ever need to remove a container from memory, you can do so as follows

```
docker container rm -f agitated_lewin
```

5. Container Techniques

- Off-the-shelf Linux distributions
- Passing commands to a container
- Attaching to a container
- Executing commands in a container
- Executing interactive commands in a container

Off-the-Shelf Linux Distributions

- Docker Hub provides various off-the-shelf images for common Linux distributions
  - E.g. Centos, Ubuntu, Debian, etc.
  - See https://hub.docker.com for details

- `alpine` is a minimal Docker image based on Linux
  - Only 5MB in size
  - Let's pull the `alpine` image into our local Docker repo

```
docker pull alpine
```

  - Let's verify the `alpine` image downloaded OK

```
docker image ls
```

Passing Commands to a Container

- You can pass commands to a container…

- Example
  - In this example, we run the `alpine` container
  - We pass in a command, to open a shell and run some script

```
docker container run alpine \
/bin/sh -c "while :; do printf '.'; sleep 1; done"
```

```
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine \
> /bin/sh -c "while :; do printf '.'; sleep 1; done"
..................
```

Attaching to a Container

- Consider the following example
  - Runs the `alpine` container as a detached process
  - You don't see the console messages in your window now

```
docker container run -d --name ticker alpine \
/bin/sh -c "while :; do printf '.'; sleep 1; done"
```

- You can attach to a container, to see its console

```
docker container attach ticker
```

```
[node1] (local) root@192.168.0.13 ~
$ docker container run -d --name ticker alpine \
> /bin/sh -c "while :; do printf '.'; sleep 1; done"
69751404abdca6470603e31f44cba4171c034e8910f1b1a570e88e5ebefafa46
[node1] (local) root@192.168.0.13 ~
$ docker container attach ticker
...............
```

Executing Commands in a Container

- You can execute commands within a container
  - Use `docker container exec`

- Example
  - In this example, we run the `ps` command inside the container named `ticker` (see prev slide for reminder about this container)

```
docker container exec ticker ps
```

```
[node1] (local) root@192.168.0.13 ~
$ docker container exec ticker ps
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh -c while :; do printf '.'; sleep 1; done
   19 root      0:00 sleep 1
   20 root      0:00 ps
[node1] (local) root@192.168.0.13 ~
$
```

Executing Interactive Commands in a Container

- You can execute commands within a container that allow you to interact with the container via a TTY window
  - Use `docker container exec -i -t`

- Example - execute a shell within the container itself

```
docker container exec -i -t ticker /bin/sh
```

```
[node1] (local) root@192.168.0.13 ~
$ docker container exec -i -t ticker /bin/sh
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    s
/ # echo "Cymru am byth"
Cymru am byth
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh -c while :; do printf '.'; sleep 1; done
  122 root      0:00 /bin/sh
  159 root      0:00 sleep 1
  160 root      0:00 ps
/ #
```

We can now run commands within the container environment

Any Questions?