Beginners Track

Introduction to Docker

- What is Docker?
- Difference between Docker & Container
- What are Containers? What are they used for?
- Difference between VM and Docker
- Similarity between VM and Docker
- How is Container different from Virtual Machine?
- Top Reasons why to & why not to run Docker containers directly on Bare Metal System
- How is Docker Networking different from VM Networking
- Understanding Docker Underlying Technologies
- What happen when Containers are Launched?
- Can container communication cross over to noncontainerized apps?
- Architecture of Docker
    - Docker Enterprise Edition
- Docker Engine Release Features
    - 18.09
    - 19.03 Community Edition
        - How to install latest Docker 19.03.0 Test Build
        - Support for `docker context`
        - Support for rootless Docker
        - Context Switching Made Simple for Swarm & Kubernetes in Docker 19.03.0
        - Test Drive –gpu option during docker CLI runtime on Docker 19.03.0 Beta 3
    - 19.03 Enterprise Edition

Installing, Upgrading & Maintaining Docker

- Installing Docker on
    - Linux
    - Windows
    - IoT Platform
        - How to install Docker 18.09.0 on Raspberry Pi 3?
        - How to setup Docker Swarm Cluster on Raspberry Pi
        - How to install Docker 19.03 on NVIDIA Jetson Nano
        - Building up K3s Cluster on Raspberry Pi 3 Nodes
        - How to monitor a Docker Swarm with Blinkt! LED using Raspberry Pi 3
        - Docker on Arduino Uno & Johnny Five
- Compiling Your Own Docker Binary from Source
- Upgrading Docker from CE to EE

Docker Components

- Docker Client-Server Architecture
- Docker Daemon
    - How to open Docker Daemon to External world?
- What is a Docker Image?
    - Building Your own Docker Image from Scratch
- What is Docker Container?
- Difference between Docker Image Vs Docker Container?
- What is Docker registry?
    - Building a Private Docker Registry
    - Building a Private Docker Registry using Portus

Working with Docker Image & Container

- Running Hello World Example
- Working with Docker Image
    - Saving Images and Containers as Tar Files for Sharing
    - Versioning an Image with Tags
- Building Your First Alpine Docker Image and Push it to DockerHub
- Building Docker Image from Scratch
- Creating Docker Base Image
- Using ONBUILD Images

Working with Dockerfile

- Building Docker Image from Dockerfile
    - Writing Your First DockerFile
    - Injecting files into your image using ADD
    - Rebuilding without Cache
- How is ENTRYPOINT instruction under Dockerfile different from RUN instruction?
- Difference between Docker Compose Vs Dockerfile
- How to use ARG to pass enviornmental variable at runtime

Accessing & Managing Docker Container

- Accessing the Container Shell
- Running a Command inside running Container
- Managing Docker Containers

Getting Started with Docker Volume

Creating Volume Mount from Dockerfile
Managing volumes through Docker CLI
Creating Volume Mount from **docker run** command & sharing same Volume Mounts among multiple containers

Mounting host directory into container
Creating Volume with Alpine OS

Docker Networking

- Using Docker Networks

Intermediate Track

Docker for Developers

Multi-Stage Build

Docker Desktop for Windows

Install Docker Desktop for Windows
Geting Started with Windows Containers
Multi-Container Applications

Introduction to Docker Networking

- Understanding Docker Container Networking
- Difference between Bridge Vs Overlay Network
- Verifying host-level settings that impact Docker networking
- What is MacVLAN networking?
    - Implementing MacVLAN
- Docker Networking CLI
    - Disable Networking for Container
    - Finding IP address of Container
    - Exposing a Container Port on the Host
    - Linking Containers in Docker
    - Configuring DNS
    - Advanced network configuration
    - Quick Configuration guide

Docker Compose

- Docker Compose Introduction
- Docker Compose Cheatsheet
- Docker compose with swarm secrets

Docker Swarm - Introduction

- What is Docker Swarm?
- How Docker Swarm Mode works?
- What is difference between Docker Swarm(Classic Swarm) Vs Swarm Mode Vs Swarmkit?
- Docker Swarm - Under the Hood

Getting Started with Docker Swarm - Lab Sessions

- Getting Started with Docker Swarm
- Lab01 - Create Overlay Network
- Lab02 - Deploy Service
- Lab03 - Inspecting State
- Lab04 - Scale Service
- Lab05 - Deploy the application components as Docker services
- Lab06 - Drain a node and reschedule the containers
- Demonstrating Service Discovery under Docker Swarm Mode
- How to Lock Docker Swarm
- Demonstrating Swarm Synchronous Services
- NFS Volume with Docker Swarm
- Building Docker Swarm Topology using Script
- Docker Service Inspection Filtering & Template Engine under Swarm Mode

Docker Swarm - Logging Solutions

- [Service Logs with Docker Swarm]

Docker Swarm - Networking Solution

- Implementing MacVLAN under Docker Swarm
- Toplology Aware Scheduling with Docker Swarm
- What’s new in Docker 1.12 Scheduling? – Part-I
- What’s new in Docker 1.12.0 Load-Balancing feature?

Docker Swarm - Updates & Rollback

- Service Rollback with Docker Swarm

Docker Swarm - High Availability, Placement & Constraints

- Implementing High Availability with Docker Swarm
- Secrets Management with Docker Swarm
- Scheduling Placement with Docker Swarm

Docker Swarm - HealthCheck Solution

- Health Aware Orchestration with Docker Swarm

Docker Swarm - Security

- TLS and Certificate Authority with Docker Swarm

Docker Desktop for Mac

- Install Docker Desktop
- Win container Deploy ,Login,exit container
- List, Start, Stop, restart containers
- Where containers are stored
  Working with container hostnames
  Working on multiple containers
  Container inspect
  Deleting containers

Introduction to Docker Application Packages

A First Look at Docker Application Packages - “docker-app”

Advanced Track

Docker Security

[What is Container Security?]
[Is Docker secure?]
[what is Docker privilege Mode?]
[Explain Docker Security in terms of Kernel Namespace]
[Docker daemon requires root privileges. Is it still secure?]
[What are Kernel Security Features?]
[How to protect the Docker daemon socket]
[How to build security into the container pipeline]
Docker Security - An Easy Way
How Docker bypasses Linux Auditing Mechnism?

Docker Monitoring

Manage and Monitor the Docker Containers with Dry Tool in 5 Min

Docker Content Trust

What is Docker Content Trust?
[How to enable Docker Content Trust?]
[How to manage keys for content trust?]

[How to implement automation with content trust?]
[Implementing delegations for content trust?]
[Play in a content trust sandbox]
[Deploy Notary Server with Compose]

Continous Integration & Deployment(CI-CD)

5 min CI/CD pipelining using Docker & circle-ci
CI - CD using Docker & Azure DevOps integrated with MS Teams

Automation Tool - Ansible, Puppet, Terraform & Chef

[Spin Up AWS Infrastructure using Terraform]
[Spin Up GCP Infrastructure using Terraform]
[Spin Up Azure Infrastruture with Terraform]
Using Terraform to build Nginx Docker Container running on Docker for Mac 18.05
[Deploy Kubernetes Cluster on Linux Vagrant Instances using Ansible]
[Deploy Docker Swarm Cluster on Linux Vagrant Instances using Ansible]

Docker Enterprise

Introduction

What is Docker Enterprise and why do we need it?
[What are Docker Enterprise Supported Platforms?]
What are components of Docker Enterprise?
Overview of Universal Control Plane
[Deep Dive into Docker Enterprise Architecture]
[What Kubernetes Features are included under Docker Enterprise 2.1?]
[What security features are in-built into Docker Enterprise 2.1?]
[Compatibility Matrix under Docker Enteprise 2.1]
[Deploy Application using Swarm on Docker Enterprise]
[Deploy Application using Kubernetes on Docker Enterprise]

Docker Enterprise 2.0

[What's New in Docker Enterprise 2.0?]
[How to Install Docker Enterprise 2.0 on RHEL?]
[How to Install Docker Enterprise 2.0 on Ubuntu?]
[How to Install Docker Enterprise 2.0 on CentOS?]
How to Install Docker Enterprise 2.0 on Windows?
[How to Upgrade from Docker CE to EE?]
[What Kubernetes Features are included under Docker Enterprise 2.0?]
[What security features are in-built into Docker Enterprise 2.0?]
[Compatibility Matrix under Docker Enteprise 2.0]
How to Install helm under Docker Enterprise 2.0?

[How to Install OpenFaas under Docker Enterprise 2.0?]
[How to Install Istio under Docker Enterprise 2.0?]

Docker Enterprise & Logging Solution

[TBD]

Docker Enterprise & Backup Solution

[TBD]

Docker Enterprise & Secrets Configuration

[TBD]

Docker Advance Labs

[Installing TICK stack using docker]
[Dockerize MAven Project]
[Docker bench]
[Contaienr Scanning Using Trivy]
[Raspberry pi Temperature sensing using docker]
[Using Docker for Go development]

Docker in Production

Best Practices for Deploying Production-Level Web Services using Docker

https://dockerlabs.collabnix.com/intermediate/workshop/

Docker compose

Docker compose is a tool built by docker to ease the task to creating and configring multiple containers in a development environment counter-part of docker-compose for prodcution environment is `docker swarm`. Docker compose takes as input a `YAML` configuration file and creates the resources (*containers*, *networks*, *volumes* etc.) by communicating with the docker daemon through docker api.

Introduction to Compose

`Compose` project is the official open source project for Docker and is responsible for the rapid orchestration of Docker container clusters. Functionally, it is very similar to Heat OpenStack .

The code is currently open sourced at https://github.com/docker/compose .

`Compose` positioned as “Defining and running multi-container Docker applications”, and its predecessor is the open source project Fig.

Through the introduction in the first part, we know that using a Dockerfile template file allows users to easily define a separate application container. However, in daily work, it is often the case that multiple containers need to cooperate to complete a certain task. For example, to implement a Web project, in addition to the Web service container itself, it is often necessary to add a back-end database service container, and even a load balancing container.

`Compose` just meets this need. It allows the user to define a set of associated application containers as a project through a separate docker-compose.yml template file (YAML format).

There are two important concepts in Compose :

`Service:` A container for an application that can actually include several container instances running the same image.

Project: A complete business unit consisting of a set of associated application containers, defined in the `docker-compose.yml` file.

`Compose` ‘s default management object is a project that provides convenient lifecycle management of a set of containers in a project through subcommands.

`Compose` project is written in Python, and the implementation calls the API provided by the Docker service to manage the container. Therefore, as long as the platform being operated supports the Docker API, you can use Compose to manage it.

Compose file used in examples

```
version: '3'
```

```
services:
    web:
        build: .
        image: web-client
        depends_on:
        - server
        ports:
        - "8080:8080"
    server:
        image: akshitgrover/helloworld
        volumes:
        - "/app" # Anonymous volume
        - "data:/data" # Named volume
        - "mydata:/data" # External volume

volumes:
    data:
    mydata:
        external: true
```

Refer this for configuring your compose file.

CLI Cheatsheet

- Docker compose
    - Compose file used in examples
    - CLI Cheatsheet
        - Build
        - Bundle
        - Config
        - Up
        - Down
        - Scale
        - Start
        - Stop

Build

Used to build services specified in docker-compose.yml file with `build` specification.

Refer this for more details.

Note: Images build will be tagged as {DIR}_{SERVICE} unless image name is specified in the service specification.

```
docker-compose build [OPTIONS] [SERVICE...]

OPTIONS:

--compress | Command line flag to compress the build context, Build context is nothing but a directory where docker-compose.yml file is located. As this directory can container a lot of files, sending build context to the container can take a lot of time thus compression is needed.

--force-rm | Remove any intermediate container while building.
```

```
--no-cache | Build images without using any cached layers from previoud builds.

--pull | Allways pull newer version of the base image.

-m, --memory | Set memory limit for the container used for building the image.

--parallel | Exploit go routines to parallely build images, As docker daemon is written in go.

--build-arg key=val | Pass any varaible to the dockerfile from the command line.


SERVICE:

If you want to build any particular services instead of every service specified in the compose file pass the name (same as in the compose file) as arguments to the command.

Example:

docker-compose build --compress     # Will compress the build context of service web.
```

Bundle

Used to generate distributed application bundle (DAB) from the compose file.

Refer this for more details about DBA.

```
docker-compose bundle [OPTIONS]

OPTIONS:

--push-image | Push images to the register if any service has build specifcation.

-o, --output PATH | Output path for .dab file.
```

Config

Used to validate the compose file

NOTE: Run this command in direcotry where docker-compose.yml file is located.

```
docker-compose config
```

Up

Creates and starts the resources as per the specification the docker-compose.yml file.

```
docker-compose up [OPTIONS] [SERVICE...]

OPTIONS:

-d, --detach | Run containers in background.

--build | Always build images even if it exists.

--no-deps | Avoid creating any linked services.
```

```
--force-recreate | Force recreating containers even if specification is not
changed.

--no-recreate | Do not recreate containers.

--no-build | Do not build any image even if it is missing.

--no-start | Just create the containers without starting them.

--scale SERVICE=NUM | Create multiple containers for a service.

-V, --renew-anon-volumes | Recreate anonymous volumes instead of getting data
from previous ones.

Example:

docker-compose up -d        # Will run service containers in background
docker-compose up web       # Will start service web and server because of
'depends_on' field
docker-compose up server    # will start server service only.
```

Down

Stop and clear any resources created while lifting docker-compose.

By default only containers and networks defined in the compose file are removed. Networks and Volumes with external = true and never removed.

```
docker-compose down [OPTIONS]

--rmi type | Remove images Type = all (Remove every image in the compose file),
local (Remove images with no custom tag)

-v, --volumes | Remove named volumes except the external ones and also remove
anonymous volumes

-t, --timeout TIMEOUT | Speficy shutdown time in seconds. (default = 10)

Example:

docker-compose down         # Will delete all containers of both web and server
and no volume will be removed

docker-compose down -v      # Will also delete anonymous and data volumes.
```

Scale

Scale particular services

```
docker-compose scale [SERVICE=NUM...]

Example:

docker-compose scale server=3 web=2
```

Start

Start created containers.

```
docker-compose start [SERVICE...]
```

```
Example:

docker-compose start        # Start containers for every service.
docker-compose start web    # Start containers only for service web.
```

Stop

Stop running containers.

```
docker-compose stop [SERVICE...]

Example:

docker-compose stop         # Stop containers for every service.
docker-compose stop web     # Stop containers only for service web.
```

Difference between Dockerfile and docker-compose

A **Dockerfile** is a text document that contains all the commands/Instruction a user could call on the command line to assemble an image.
For example

```
FROM centos:latest
LABEL maintainer="collabnix"
RUN yum update -y && \
        yum install -y httpd net-tools && \
        mkdir -p /run/httpd
EXPOSE 80
ENTRYPOINT apachectl "-DFOREGROUND"
```

Using **docker build** commmand we can build an image from a Dockerfile.

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. By default, docker-compose expects the name of the Compose file as **docker-compose.yml** or **docker-compose.yaml**. If the compose file have different name we can specify it with **-f** flag.

A docker-compose.yml looks like this:

```
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Validate the docker-compose file

```
docker-compose config
```

Build Command

In this lab we are going to look into **docker-compose build** command.Docker build used to create a new image using the instructions in the Dockerfile. The **build** can be specified either as a string containing a path to the build context. The newly built image will be used to create the container for the service.

Command instructions

build

The format is docker-compose build [options] [SERVICE…] .

Build (rebuild) the service container in the project.

Once the service container is built, it will be tagged with a tagname, such as a db container in a web project, possibly web_db.

You can rebuild the service at any time by running the docker-compose build in the project directory.

Options include:

`--force-rm` removes the temporary container during the build process.

`--no-cache` does not use cache during the build image process (this will lengthen the build process).

`--pull always` tries to get a mirror of the updated version by `--pull` .

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

We are going to build an nginx image with custome page

Setup environment:

```
$ mkdir docker-compose/build
$ cd docker-compose/build
```

Now lets create the Dockerfile

```
FROM nginx:alpine
RUN echo "Welcome to Docker Workshop!" >/usr/share/nginx/html/index.html
CMD ["nginx", "-g", "daemon off;"]
```

Create a docker-compose.yml file

```
version: "3.7"
services:
  webapp:
    build:
      context: .
      dockerfile: Dockerfile
    image: webapp:v1
```

context: To specify the build context directory that is sent to the Docker daemon.
dockerfile: use to specify Alternate Dockerfile or path to Dockerfile.

Build the image using docker-compose

```
$ docker-compose build
```

Since we specified **image:** as well as **build:**, then the Compose built the image with name **webapp** and tag **v1**.
If we didnt specify the **image:** option the image name will be **buid_**

Check the image have created

```
$ docker image ls webapp:v1
```

Quick Notes:

build

Specifies the path to the folder where the Dockerfile is located (either an absolute path or a path relative to the docker-compose.yml file). Compose will use it to automatically build this image and then use this image.

```
 version: '3' services:  webapp:  build: ./dir
```

You can also use the context directive to specify the path to the folder where the Dockerfile is located.

Use the dockerfile directive to specify the Dockerfile filename.

Use the arg directive to specify the variables when the image is built.

```
 version: '3' services:  webapp:  build:  context: ./dir  dockerfile:
Dockerfile-alternate  args:  buildno: 1
```

Use cache_from specify the cache to build the image

```
 build:  context: .  cache_from:  - alpine: latest  - corp/web_app: 3.14
```

pull Command

The `docker-compose pull` command pull down the images which is specified under each service of docker-compose file from the docker hub.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Create a docker-compose.yml file
- Pull down the service images
- Pull down image of a single service

Create a docker-compose.yml file

```
version: '3.1'
services:
  #Nginx Service
   webserver:
     image: nginx:alpine
     container_name: webserver
     restart: unless-stopped
     ports:
       - "80:80"
       - "443:443"
   dbserver:
     image: mysql:5.7
     container_name: Mysqldb
     restart: unless-stopped
     ports:
       - "3306:3306"
```

Pull down the service images

```
$ docker-compose pull
Pulling webserver ... done
Pulling dbserver  ... done
```

Checking the images in local

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
mysql               5.7                 383867b75fd2        2 hours ago
373MB
nginx               alpine              d87c83ec7a66        2 weeks ago
21.2MB
```

Pull down image of a single service

Before doing pull make sure you have removed the images(docker image rm <Image_name/Image_ID>) which is already pulled.

```
$ docker-compose pull webserver
Pulling webserver ... done
```

This pulldown only the webserver service image(nginx:alpine).

push Command

The `docker-compose push` command help you tou push the service images to Docker Hub or your own private Docker registry.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Dockerfile for custom docker image
- Create a docker-compose.yml file
- Build the image using docker-compose
- Upload the image to Docker registry

Dockerfile for custom docker image

$ mkdir -p dockerlabs/{nginx,httpd} ; cd dockerlabs

Dockerfile_nginx

```
$ echo 'FROM nginx:alpine
RUN echo "nginx - Welcome to Docker Workshop!" >/usr/share/nginx/html/index.html
CMD ["nginx", "-g", "daemon off;"] ' > nginx/Dockerfile_nginx
```

Dockerfile_httpd

```
$ echo 'FROM httpd:alpine
RUN echo "httpd - Welcome to Docker Workshop!" >
/usr/local/apache2/htdocs/index.html
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"] ' > httpd/Dockerfile_httpd
```

Create a docker-compose.yml file

docker-compose.yml

```
version: "3.7"
services:
  customNginx:
    build:
```

```
      context: .
      dockerfile: nginx/Dockerfile_nginx
    image: saviovettoor/custom_nginx_dockerlabs:v1
  customHttpd:
    build:
      context: .
      dockerfile: httpd/Dockerfile_httpd
    image: saviovettoor/custom_httpd_dockerlabs:v1
```

NOTE: Make sure that image name should be `<USER_NAME> / <REPOSITORY>`.

Build the image using docker-compose

```
$ docker-compose build
```

Check the image have created

```
$ docker image ls
REPOSITORY                                TAG                 IMAGE ID
CREATED             SIZE
saviovettoor/custom_nginx_dockerlabs   v1                  3098d9cb3971        3
minutes ago       21.2MB
saviovettoor/custom_httpd_dockerlabs   v1                  866e070c373a        3
minutes ago       127MB
```

Upload the image to Docker registry

NOTE: before tryng to push the image log in to hub.

$ docker login -u

```
$ docker-compose push
```

Upload a single service image

```
$ docker-compose push customNginx
Pushing customNginx (saviovettoor/custom_nginx_dockerlabs:v1)...
The push refers to repository [docker.io/saviovettoor/custom_nginx_dockerlabs]
e4f534d7f270: Pushed
3e76d2df1790: Mounted from library/nginx
03901b4a2ea8: Mounted from library/nginx
v1: digest:
sha256:aa83133b840728922ad95133ff17ed95fed7d3fb89e9919925a874cf848cd282 size:
946
```

up Command

The `docker-compose up` command help you to bring up a multi-container application, which you have described in your docker-compose file.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Create a docker-compose.yml with custom image
- Build container and network
- Bringup the containers

Create a docker-compose.yml with custom image

Setup environment

```
$ mkdir -p docker-compose/build
$ cd docker-compose/build
```

Now lets create the Dockerfile

```
FROM nginx:alpine
RUN echo "Welcome to Docker Workshop!" >/usr/share/nginx/html/index.html
CMD ["nginx", "-g", "daemon off;"]
```

Create a docker-compose.yml file

```
version: "3.7"
services:
  webserver:
    build:
      context: .
      dockerfile: Dockerfile
    image: webapp:v1
    container_name: Nginx
    ports:
      - "80:80"
  dbserver:
     image: mysql:5.7
```

```
    container_name: Mysqldb
    restart: unless-stopped
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: Pa$$w0rd
      MYSQL_USER: test
      MYSQL_PASSWORD: Pa$$w0rd123
      MYSQL_DATABASE: test
    volumes:
      - db_data:/var/lib/mysql
volumes:
  db_data:
```

Build container and network

Won't start the container

```
$ docker-compose up --no-start
Creating network "build_default" with the default driver
Creating Nginx   ... done
Creating Mysqldb ... done
```

Checking the status

```
$ docker-compose ps
 Name               Command             State    Ports
-------------------------------------------------------
Mysqldb   docker-entrypoint.sh mysqld   Exit 0
Nginx     nginx -g daemon off;          Exit 0
```

Bringup the containers

```
$ docker-compose up -d
```

NOTE: This will build the images and bringup the containers. **-d** option which will run the container in background.

Checking webserver respone

```
$ curl http://localhost
Welcome to Docker Workshop!
```

Checking dbserver respone

```
$ docker exec -it Mysqldb mysql -u root -p
Enter password:
```

Rebuild the docker image and bring the stack up

```
$ docker-compose up -d --build
```

images Command

The `docker-compose images` command help to listout images used/created by the containers.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Create a docker-compose.yml with custom image
- Build container and network
- Bringup the containers
- List out the images used by the containers

Create a docker-compose.yml with custom image

Setup environment

```
$ mkdir -p docker-compose/build
$ cd docker-compose/build
```

Now lets create the Dockerfile

```
FROM nginx:alpine
RUN echo "Welcome to Docker Workshop!" >/usr/share/nginx/html/index.html
CMD ["nginx", "-g", "daemon off;"]
```

Create a docker-compose.yml file

```
version: "3.7"
services:
  webserver:
    build:
      context: .
      dockerfile: Dockerfile
    image: webapp:v1
    container_name: Nginx
    ports:
      - "80:80"
  dbserver:
     image: mysql:5.7
```

```
    container_name: Mysqldb
    restart: unless-stopped
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: Pa$$w0rd
      MYSQL_USER: test
      MYSQL_PASSWORD: Pa$$w0rd123
      MYSQL_DATABASE: test
    volumes:
      - db_data:/var/lib/mysql
volumes:
  db_data:
```

Bringup the containers

```
$ docker-compose up -d
```

NOTE: This will build the images and bringup the containers. **-d** option which will run the container in background.

List out the images used by the containers

```
$  docker-compose images
Container   Repository   Tag     Image Id      Size
------------------------------------------------------
Mysqldb     mysql        5.7   383867b75fd2   356 MB
Nginx       webapp       v1    3454fa918c0d   20.2 MB
```

mysql is the one used for dbserver and **webapp:v1** is the custome image used for webserver.

Logs Command

The `docker-compose logs` command help to see the service logs.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Create a docker-compose.yml file
- Bringing up the containers
- Checking Service logs
- Follow log output
- Checking last two line logs

Create a docker-compose.yml file

```
version: '3.7'
services:
  #Nginx Service
   webserver:
     image: nginx:alpine
     container_name: Nginx
     restart: unless-stopped
     ports:
       - "80:80"
       - "443:443"
   dbserver:
     image: mysql:5.7
     container_name: Mysqldb
     restart: unless-stopped
     ports:
       - "3306:3306"
     environment:
       MYSQL_ROOT_PASSWORD: Pa$$w0rd
       MYSQL_USER: test
       MYSQL_PASSWORD: Pa$$w0rd123
       MYSQL_DATABASE: test
     volumes:
       - db_data:/var/lib/mysql
volumes:
  db_data:
```

Bringing up the containers

```
$ docker-compose up -d
```

Checking container status

```
$ docker-compose ps
 Name               Command             State                      Ports
-------------------------------------------------------------------------------
--------
Mysqldb   docker-entrypoint.sh mysqld   Up      0.0.0.0:3306->3306/tcp,
33060/tcp
Nginx     nginx -g daemon off;          Up      0.0.0.0:443->443/tcp,
0.0.0.0:80->80/tcp
```

Checking Service logs

```
$ docker-compose logs
Attaching to Mysqldb, Nginx
Mysqldb      | Initializing database
Mysqldb      | 2019-10-04T08:45:43.926007Z 0 [Warning] TIMESTAMP with implicit
DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server
option (see documentation for more details).
Mysqldb      | 2019-10-04T08:45:44.832102Z 0 [Warning] InnoDB: New log files
created, LSN=45790
Mysqldb      | 2019-10-04T08:45:44.942095Z 0 [Warning] InnoDB: Creating foreign
key constraint system tables.
Mysqldb      | 2019-10-04T08:45:45.017777Z 0 [Warning] No existing UUID has been
found, so we assume that this is the first time that this server has been
started. Generating a new UUID: 5acfb862-e683-11e9-bee0-0242ac130003.
```

Follow log output

Access the nginx while check the logs, to see live logs.

```
$ docker-compose logs -f webserver
Attaching to Nginx
Nginx        | 172.18.0.1 - - [04/Oct/2019:08:48:42 +0000] "GET / HTTP/1.1" 200
612 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/77.0.3865.90 Safari/537.36" "-"
```

Checking last two line logs for a service

```
$ docker-compose logs  --tail="2"  webserver
Attaching to Nginx
Nginx        | 2019/10/04 08:48:43 [error] 7#7: *1 open()
"/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory),
client: 172.18.0.1, server: localhost, request: "GET /favicon.ico HTTP/1.1",
host: "ip172-18-0-46-bmbfiuol9uvg00c4s2s0-80.direct.labs.play-with-docker.com",
referrer: "http://ip172-18-0-46-bmbfiuol9uvg00c4s2s0-80.direct.labs.play-with-
docker.com/"
Nginx        | 172.18.0.1 - - [04/Oct/2019:08:48:43 +0000] "GET /favicon.ico
HTTP/1.1" 404 555 "http://ip172-18-0-46-bmbfiuol9uvg00c4s2s0-
80.direct.labs.play-with-docker.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36" "-"
```

Port Command

The `docker-compose port` command print the public/network facing port for a service.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Create a docker-compose.yml file
- Bringing up the containers
- Listout the services
- Check the external port binded to service webserver for port 80

Create a docker-compose.yml file

```
version: '3.7'
services:
  #Nginx Service
   webserver:
     image: nginx:alpine
     container_name: Nginx
     restart: unless-stopped
     ports:
       - "8080:80"
       - "443:443"
   dbserver:
     image: mysql:5.7
     container_name: Mysqldb
     restart: unless-stopped
     ports:
       - "3306:3306"
     environment:
       MYSQL_ROOT_PASSWORD: Pa$$w0rd
       MYSQL_USER: test
       MYSQL_PASSWORD: Pa$$w0rd123
       MYSQL_DATABASE: test
     volumes:
       - db_data:/var/lib/mysql
volumes:
  db_data:
```

Bringing up the containers

```
$ docker-compose up -d
```

Checking container status

```
$ docker-compose ps
 Name               Command             State                      Ports
-------------------------------------------------------------------------------
--------
Mysqldb   docker-entrypoint.sh mysqld   Up      0.0.0.0:3306->3306/tcp,
33060/tcp
Nginx     nginx -g daemon off;          Up      0.0.0.0:443->443/tcp,
0.0.0.0:80->80/tcp
```

Listout the services

```
$ docker-compose ps --services
webserver
dbserver
```

Check the external port binded to service webserver for port 80

```
$ docker-compose port webserver 80
0.0.0.0:8080
```

Run Command

Run command help to run a one-time command against a service. `docker-compose run` will start a new container to execute the command, not executed against a running container

```
Usage:
 docker-compose run [options] [-v VOLUME...] [-p PORT...] [-e KEY=VAL...] [-l
KEY=VALUE...]
        SERVICE [COMMAND] [ARGS...]

Options:
    -d, --detach          Detached mode: Run container in the background, print
                          new container name.
    --name NAME           Assign a name to the container
    --entrypoint CMD      Override the entrypoint of the image.
    -e KEY=VAL            Set an environment variable (can be used multiple
times)
    -l, --label KEY=VAL   Add or override a label (can be used multiple times)
    -u, --user=""         Run as specified username or uid
    --no-deps             Don't start linked services.
    --rm                  Remove container after run. Ignored in detached mode.
    -p, --publish=[]      Publish a container's port(s) to the host
    --service-ports       Run command with the service's ports enabled and
mapped
                          to the host.
    --use-aliases         Use the service's network aliases in the network(s)
the
                          container connects to.
    -v, --volume=[]       Bind mount a volume (default [])
    -T                    Disable pseudo-tty allocation. By default `docker-
compose run`
                          allocates a TTY.
    -w, --workdir=""      Working directory inside the container
```

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Assignment

- Create a docker-compose.yml file
- Bringup the containers
-

Create a docker-compose.yml file

```
version: '3.7'
services:
  #Nginx Service
   webserver:
     image: nginx:alpine
     container_name: Nginx
     restart: unless-stopped
     ports:
       - "80:80"
       - "443:443"
   dbserver:
     image: mysql:5.7
     container_name: Mysqldb
     restart: unless-stopped
     ports:
       - "3306:3306"
     environment:
       MYSQL_ROOT_PASSWORD: Pa$$w0rd
       MYSQL_USER: test
       MYSQL_PASSWORD: Pa$$w0rd123
       MYSQL_DATABASE: test
     volumes:
       - db_data:/var/lib/mysql
volumes:
  db_data:
```

Bringup the containers

```
$ docker-compose up -d
```

Checking container status

```
$ docker-compose ps
 Name               Command             State                     Ports
-------------------------------------------------------------------------------
--------
Mysqldb   docker-entrypoint.sh mysqld   Up      0.0.0.0:3306->3306/tcp,
33060/tcp
Nginx     nginx -g daemon off;          Up      0.0.0.0:443->443/tcp,
0.0.0.0:80->80/tcp
```

Listout the services

```
$ docker-compose ps --services
webserver
dbserver
```

Start webserver service container with shell

```
$ docker-compose run webserver /bin/sh
```

Exec Command

In this lab we are going to look into **docker-compose exec** command. Docker **exec** is used to run commands in a running container, similarly **docker-compose exec** runs commands in your services.

Command instructions

Exec

```
docker-compose exec [options] [-e key=val...] service command [args...]
```

Run commands in your services. Commands are by default allocating a TTY.

Options include:
`-T` disables pseudo-tty allocation.
`--index=index` specifies container when there are multiple instances of a service

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Setup enviroment

```
$ mkdir app
$ cd app
```

Create a docker-compose.yml file

```
version: '3.1'
services:
  #Webservers
   webserver:
     image: nginx:alpine
     restart: unless-stopped
     expose:
       - "80"
       - "443"
  #Load Balancer
   loadbalancer:
     image: nginx:alpine
     volumes:
       - ./nginx.conf:/etc/nginx/nginx.conf:ro
     depends_on:
       - webserver
     ports:
       - "80:4000"
```

Create a nginx.conf file

```
user  nginx;
events {
    worker_connections   1024;
}
http {
        server {
              listen 4000;
              location / {
                proxy_pass http://webserver:80;
              }
        }
}
```

Note: This file will configure our load balancer.

Create the compose containers

```
$ docker-compose up -d --scale webserver=3
```

View the containers

```
$ docker-compose ps
```

Execute commands in different webservers

```
$ docker-compose exec --index=1 webserver  sh -c "echo 'Welcome to webserver1' >
/usr/share/nginx/html/index.html"
$ docker-compose exec --index=2 webserver  sh -c "echo 'This is webserver2' >
/usr/share/nginx/html/index.html"
$ docker-compose exec --index=3 webserver  sh -c "echo 'Webserver3 is up' >
/usr/share/nginx/html/index.html"
```

Verify changes

```
$ curl http://localhost
```

Note: In order to verify all changes we'll have to use the curl command multiple times.

Create first docker compose file with ngnix and mysql

In this lab we are going to bringup a nginx and mysql containers using docker-compose.

Pre-requisite:

Tested Infrastructure

Platform	Number of Instance	Reading Time
Play with Docker	**1**	**5 min**

Pre-requisite

- Create an account with DockerHub
- Open PWD Platform on your browser
- Click on **Add New Instance** on the left side of the screen to bring up Alpine OS instance on the right side

Setup environment:

```
$ mkdir Myapp
$ cd Myapp
```

Now lets create passowrd file for our DB:

```
$ openssl rand -base64 32 > db_password.txt
$ openssl rand -base64 32 > db_root_password.txt
```

Create a docker-compose.yml file:

```
version: '3.1'
services:
  #Nginx Service
   webserver:
     image: nginx:alpine
     container_name: webserver
     restart: unless-stopped
     ports:
       - "80:80"
       - "443:443"
   #Mysql DB
   db:
     image: mysql:5.7
     container_name: Mysqldb
     restart: unless-stopped
     volumes:
       - db_data:/var/lib/mysql
     ports:
       - "3306:3306"
     environment:
       MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD_FILE: /run/secrets/db_password
     secrets:
       - db_root_password
```

```
      - db_password
secrets:
   db_password:
     file: db_password.txt
   db_root_password:
     file: db_root_password.txt

volumes:
    db_data:
```

Create the compose container:

```
$ sudo docker-compose up
```

SSH into the instance and check the app

List out the compose services:

```
$ docker-compose ps
  Name                 Command             State                     Ports
-------------------------------------------------------------------------------
----------
Mysqldb     docker-entrypoint.sh mysqld   Up      0.0.0.0:3306->3306/tcp, 33060/
tcp
webserver   nginx -g daemon off;          Up      0.0.0.0:443->443/tcp,
0.0.0.0:80->80/tcp
```

Verify the nginx is running:

```
$ curl http://localhost
```

Verify the Mysql db:

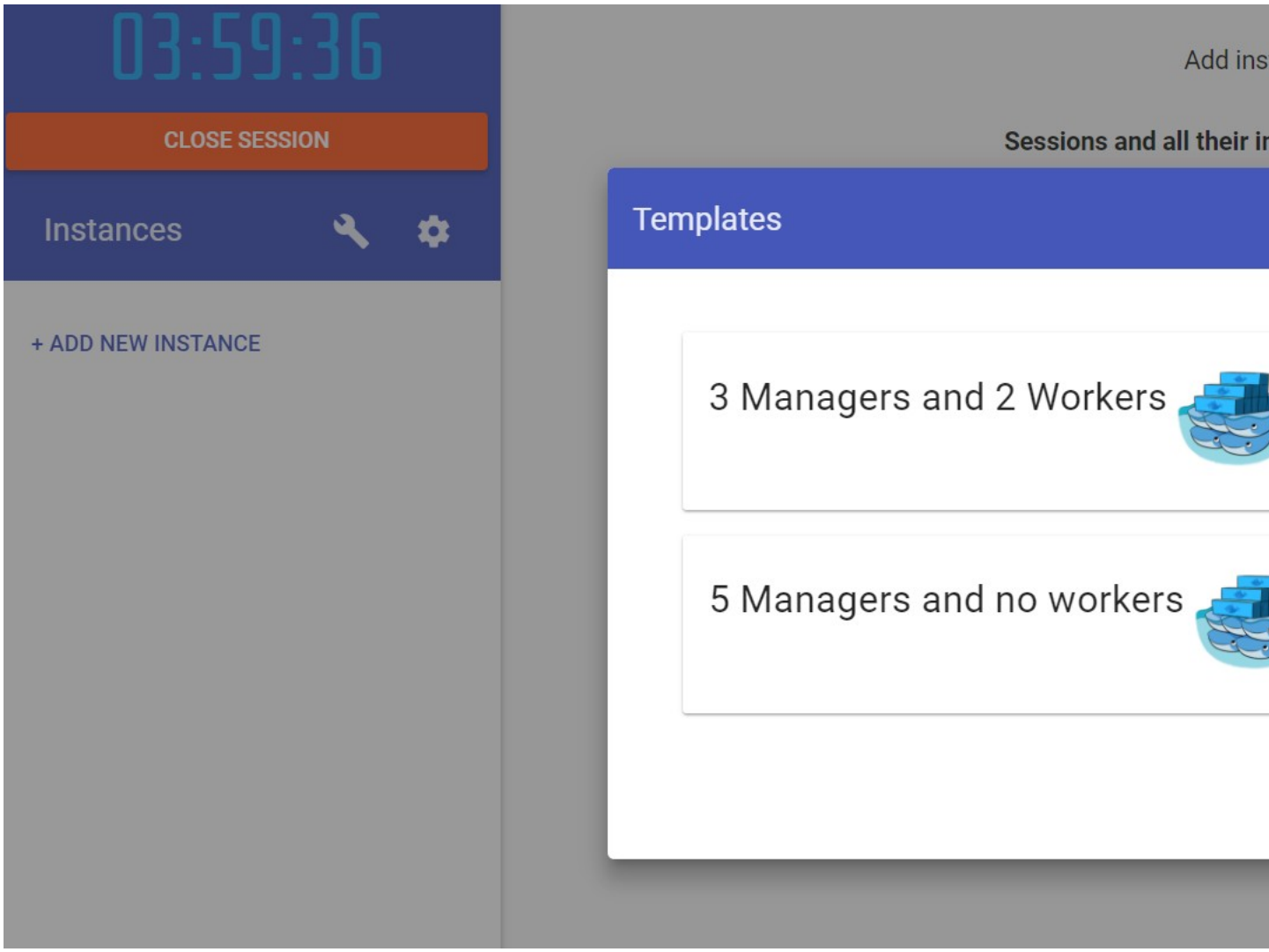
```
$ docker exec -it Mysqldb mysql -u root -p

Enter the root password which is in db_root_password.txt
```

Getting Started with Docker Swarm

To get started with Docker Swarm, you can use “Play with Docker”, aka PWD. It’s free of cost and open for all. You get maximum of 5 instances of Linux system to play around with Docker.

- Open Play with Docker labs on your browser
- Click on Icon near to Instance to choose 3 Managers & 2 Worker Nodes



- Wait for few seconds to bring up 5-Node Swarm Cluster

We recommend you start with one of our Beginners Guides, and then move to intermediate and expert level tutorials that cover most of the features of Docker. For a comprehensive approach to understanding Docker, I have categorized it as shown below:

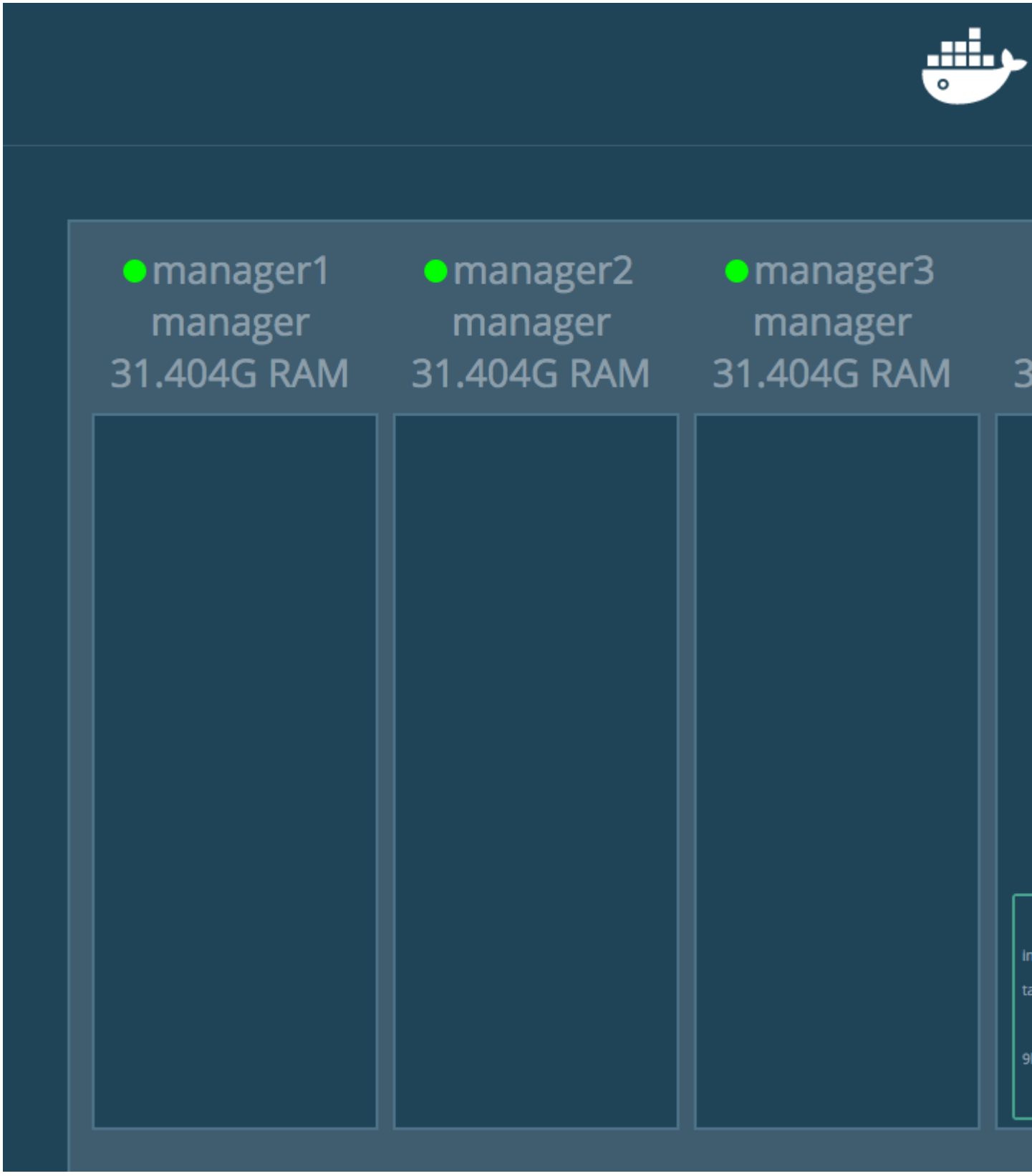
A Bonus… Docker Swarm Visualizer

Swarm Visualizer is a fancy tool which visualized the Swarm Cluster setup. It displays containers running on each node in the form of visuals. If you are conducting Docker workshop, it’s a perfect way to show your audience how the containers are placed under each node. Go..try it out..

Clone the Repository

```
git clone https://github.com/dockersamples/docker-swarm-visualizer

cd docker-swarm-visualizer
docker-compose up -d
```



To run in a docker swarm:

```
$ docker service create \
  --name=viz \
  --publish=8080:8080/tcp \
  --constraint=node.role==manager \
  --mount=type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
  dockersamples/visualizer
```

Creating an Overlay Network

Official images

All images in Docker Hub under the `library` organization (currently viewable at: https://hub.docker.com/explore/) are deemed “Official Images.” These images undergo a rigorous, open-source review process to ensure they follow best practices. These best practices include signing, being lean, and having clearly written Dockerfiles. For these reasons, it is strongly recommended that you use official images whenever possible.

Official images can be pulled with just their name and tag. You do not have to precede the image name with `library/` or any other repository name.