**2-tier application deployment using docker-compose**

**docker-compose**

**Introduction:**

**Docker-compose** is the tool which is used to deploy the infrastructure as a code from a yml file. Here I’m presenting a multi-tier application deployment using docker-compose.

It’s amazing to see that how quick an infrastructure can be created, deleted and recreated just by using a single command called docker-compose.

**Prerequisites**

In this case, we will need the following:

* A local machine installed with Docker. The machine can be running on Windows, Linux, or macOS.

**Docker Installation**

In my case, I’ve have used Redhat 8 OS and configured docker repo to pull images from the docker hub. You can use any other OS and configure in your own way.

[root@localhost ~]# cat /etc/yum.repos.d/docker.repodocker-compose -v

name=docker repo

baseurl=https://download.docker.com/linux/centos/7/x86\_64/stable/

gpgcheck=0

[root@localhost ~]# yum install docker-ce --nobest –y

[root@localhost ~]# rpm -q docker-ce

docker-ce-18.09.1-3.el7.x86\_64

[root@localhost ~]# systemctl list-unit-files|grep docker

docker.service enable

docker.socket disabled

[root@localhost ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

**docker-compose Installation**

Follow the official documentation [how to install Docker Compose](https://docs.docker.com/compose/install/) () on your system or follow below steps to install docker-compose in your machine.

[root@localhost ~]# curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-composename=docker repo

[root@localhost ~]# chmod +x /usr/local/bin/docker-compose

Verify that installed version of Docker Compose is => 1.18.0:

$ docker-compose –v

docker-compose version 1.25.5, build 8a1c60f6

**Local Container Infrastructure Description**

Docker Compose manages only Docker container infrastructure. It allows us to start containers, create networks and volumes, pass environment variables to containers, publish ports, etc.

Let's use how Docker Compose uses syntaxes to describe the infrastructure as a code.

It’s one time investment in the file of code and can be used as much as we want.

I’ve created a file called docker-compose.yml inside /myproject directory with the following content:

version: '3.3'

# define services (containers) that should be running

services:

my-mongo-db:

image: mongo:3.2

# what volumes to attach to this container

volumes:

- mongo-volume:/var/db

# what networks to attach this container

networks:

- my-network

my-app:

image: Apache:latest

# path to Dockerfile to build an image and start a container

build: .

environment:

- DATABASE\_HOST=my-mongo-db

ports:

- 6565:6565

networks:

- my-network

# start my-app only after mongod-database service was started

depends\_on:

- my-mongo-db

# define volumes to be created

volumes:

mongo-volume:

# define networks to be created

networks:

my-network:

Above code description is as below if we use docker command to create a container with a volume, network:

$ docker run --name my-mongo-db \

--volume mongo-data:/var/db \

--network my-network \

--detach mongo:3.2

It shows clearly that docker compose syntaxes are very simple to understand by it’s simple representation.

my-app services configuration is a bit different from MongoDB service in a way that we specify a build option instead of image to build the container image from a Dockerfile before starting a container:

my-app:

# path to Dockerfile to build an image and start a container

build: .

environment:

- DATABASE\_HOST=mongo-database

ports:

- 6565:6565

networks:

- my-network

# start my-app only after mongod-database service was started

depends\_on:

- mongo-database

Also, note the depends\_on option which allows us to tell Docker Compose that this my-app service depends on mongo-database service and should be started after mongo-database container was launched.

The other two top-level sections in this file are **volumes** and **networks**. They are used to define volumes and networks that should be created:

# define volumes to be created

volumes:

mongo-data:

# define networks to be created

networks:

my-network:

These basically correspond to the commands that we used in the previous lab to create a named volume and a network:

$ docker volume create mongo-data

$ docker network create my-network

**Create Local Infrastructure**

Once you described the desired state of you infrastructure in docker-compose.yml file, tell Docker Compose to create it using the following command:

$ docker-compose up

or use this command to run containers in the background:

$ docker-compose up -d

**Access Application**

The application should be accessible to your as before at [http://localhost:6565](http://localhost:9292/)

**Save and commit the work**

Save and commit the docker-compose.yml file created in this lab into your /my/project directory.

**Conclusion**

This is how how we use Docker Compose tool to implement Infrastructure as Code approach to managing a local container infrastructure. This helped us automate and document the process of creating all the necessary components for running our containerized application.

If we keep created docker-compose.yml file inside the application repository, any of our colleagues can create the same container environment on any system with just one command. This makes Docker Compose a perfect tool for creating local dev environments and simple application deployments.

To destroy the local playground, run the following command:

$ docker-compose down --volumes