Demo 2: Docker Compose

Introduction

The learning objectives of this demonstration are as follows:

- Demonstrating a Compose file with the restart policy, ports and environment variables
- Understanding the limitations of the Docker-Compose tool
- Deploying the Voting application using the Docker Compose tool

In this demonstration, we will deploy two separate services from a single compose file. The first service uses the image file **demo/nginx2,** and the second will use **demo/nginx**. Both of these images were created in the previous demo and are only a variant of each other. Initially, we will expose these services at ports 82 and 80 of the host, respectively. Then, we will enter inside the container of the first service and stop the Nginx process inside it; as we do so, the container will also stop.

Demonstrating the Compose File With the Restart Policy, Ports and Environment Variables

1. Understand the docker-compose file given below.

docker-compose.yml
version: "3" services: nginx: image: demo/nginx2 ports: - 82:80 nginx2: image: demo/nginx ports: - 80:80

2. Run the command **docker-compose up -d to** deploy the application.
3. Now, enter inside the container of the Nginx service using **docker exec -it <container_name> bash** and run the command **service nginx stop.**
4. Run **docker ps** to see only one container is running, meaning that one container has stopped.

Restart Policy

Now, we will set a restart policy for that service in the docker compose file and update the deployed services. Again, when we enter the first service and stop the Nginx process, we will see that the container has restarted.

5. Now, define a **restart policy** in the compose file like this:

docker-compose.yml
version: "3" services: nginx: image: demo/nginx2 **restart: always** ports: - 82:80 nginx2: image: demo/nginx ports: - 80:80

6. Now, if you try to stop the container running the service 'nginx', you will notice that the container has not stopped.

Environment Variables and the ‘depends_on’ Keyword

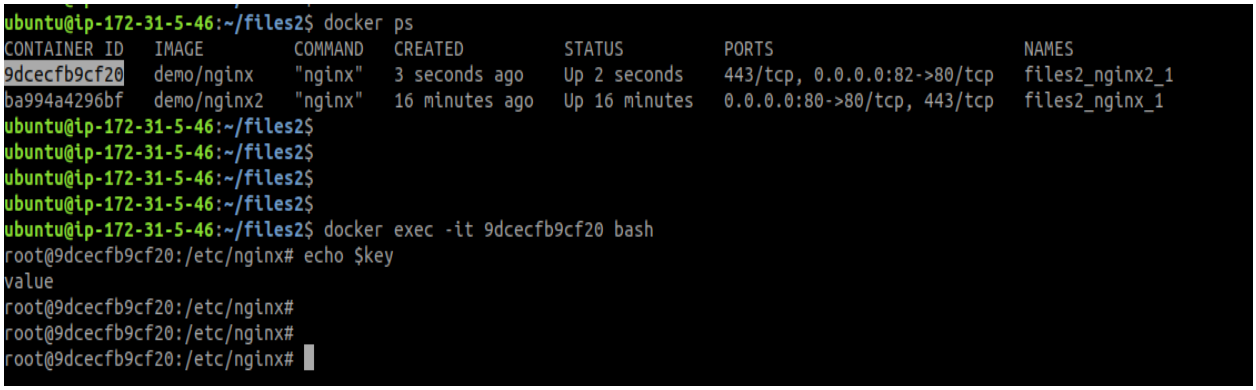
Now, edit the compose file and add an environment variable that needs to be passed inside the containers of the particular service. After we deploy using the revised docker compose file, enter inside that particular container and echo $key (environment variable name) to check whether the environment variable is passed correctly.

Below is the demonstration for environment variables and ‘depends_on’ keyword:

1. Let’s pass an environment variable in the container of the ‘nginx2’ service through the docker compose file. Also, let’s assume that our nginx2 service has some dependency on the nginx service, and the former should be deployed only after the latter. We can edit the dockerfile such that the nginx2 service starts after the nginx service only by using the depends_on keyword.

docker-compose.yml
version: "3" services: nginx: image: demo/nginx2 restart: always ports: - 82:80 nginx2: image: demo/nginx **depends_on:** **- nginx** **environment:** **- key=value** ports: - 80:80

2. Enter inside the nginx2 container **docker exec -it <container name> /bin/bash** and run **echo $key.**



Limitations of the Docker-Compose Tool

Now, we will try to deploy two replicas of one of the services by editing the compose file. We will see that it shows an error message because two containers cannot be mapped to the same host port. Now, we will remove the mapped host port from the docker compose file, leaving only the container port that needs to be exposed. Now, when you deploy using this docker compose file, you will find that the container ports are exposed to random ports of the host, and you are able to deploy replicas of a service. However, the issue is that you do not know the ports beforehand; hence, accessing containers from outside the docker network becomes impossible (although services can still communicate with each other using the service name without knowing the ip and port). This can be solved by deploying a nginx-based load balancer (at the predefined port) that redirects to these containers. This is for you to try as homework.

1. Let's scale the nginx service, replica=2.

```
ubuntu@ip-172-31-5-46:~/files2$ cat docker-compose.yml
services:
 nginx:
  image: demo/nginx2
  restart: always
  ports:
   - 80:80
 nginx2:
  image: demo/nginx
  deploy:
   replicas: 2
  depends_on:
   - nginx
  environment:
   - key=value
  ports:
   - 82:80
```

2. Run **docker-compose up --scale nginx2=2.**

```
ubuntu@ip-172-31-5-46:~/files2$ docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Creating network "files2_default" with the default driver
Creating files2_nginx_1 ... done
WARNING: The "nginx2" service specifies a port on the host. If multiple containers for this service are created on a single host, the port will
 clash.
Creating files2_nginx2_1 ...
Creating files2_nginx2_1 ... error
Creating files2_nginx2_2 ... done

ERROR: for files2_nginx2_1  Cannot start service nginx2: driver failed programming external connectivity on endpoint files2_nginx2_1 (6d25dea73
65ecfeeffdf59a633824c2414f815564ece509f78b3db7479d5c921): Bind for 0.0.0.0:82 failed: port is already allocated

ERROR: for nginx2  Cannot start service nginx2: driver failed programming external connectivity on endpoint files2_nginx2_1 (6d25dea7365ecfeeff
df59a633824c2414f815564ece509f78b3db7479d5c921): Bind for 0.0.0.0:82 failed: port is already allocated
ERROR: Encountered errors while bringing up the project.
ubuntu@ip-172-31-5-46:~/files2$
```

(The **'deploy'** keyword is ignored by the docker-compose command; it works for docker swarms.)

3. Running multiple replicas on a predefined port is not possible, as two containers cannot be mapped to the same port of the host.
4. The solution is dynamic port mapping, so simply leave the host port empty, and docker will automatically assign a random host port to the container.

docker-compose.yml

```
version: "3"
services:
 nginx:
  image: demo/nginx2
  restart: always
  ports:
   - 80
 nginx2:
  image: demo/nginx
  deploy:
   replicas: 2
  depends_on:
   - nginx
  environment:
   - key=value
  ports:
   - 80:80
```

5. However, there is an issue; let's say we have to deploy two replicas of a frontend container. Now, if we do not know the ports beforehand (dynamic port mapping), then how will we connect to container from the external network? **Remember** that the containers on the same host can still communicate simply with the servicename, and dns is resolved by docker.
6. To solve this, you can create a load balancer (deploying nginx containers on a predefined port, i.e., 82, and then redirecting the request to the service name "nginx"). Containers within the host can communicate with other containers simply by service name.

Deploying the Voting Application Using the Docker Compose Tool

Pull the repository https://github.com/dockersamples/example-voting-app on your local host and enter inside the repository folder.

The following docker-compose file will be used to deploy the voting application.

docker-compose-simple.yml

```
version: "3"

services:
  vote:
    build: ./vote
    command: python app.py
    volumes:
     - ./vote:/app
    ports:
      - "5000:80"

  redis:
    image: redis:alpine
    ports: ["6379"]

  worker:
    build: ./worker

  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"

  result:
    build: ./result
    command: nodemon server.js
    volumes:
      - ./result:/app
    ports:
```

```
- "5001:80"
- "5858:5858"
```

Now, run the **docker-compose up -d** command to deploy the application. Please ensure that you run the command in the same directory where the above-mentioned (docker-compose-simple.yml) file is present.

You can check the status of the deployed containers using the following command:

docker ps

Next, hit your instance's public IP; you will be served with the content of the result service.