Automating Deployments Using Docker Stack

A docker-compose alternative for swarm cluster

We'll cover the following

- Docker stack implementation
- Docker stack commands

In this lesson, we will automate all the steps we took to create and deploy our app using services.

> Clone the project using `git clone https://github.com/venky8283/Docker.git` Type `git checkout fc0a9af7f7fb2c8d61e4777befdbf451568cf578` to get the exact code used in this lesson.

We did the same thing while working with docker-compose. In docker-compose, we created a docker-compose file and automated all the steps in it.

We will modify the same docker-compose file to run with docker-swarm because Compose and Docker stack are similar on some level.

Let's list all the steps we need to automate as follows:

- Pull the Flask_app v3.0 image from Docker Hub
- Increase the number of replicas of the Flask app to three
- Create a database service
- Establish communication between services using the network
- Create a visualizer service to monitor the swarm

Docker stack implementation

Docker stack is a bundle of services. When you want to deploy a set of services to a machine, instead of creating each service separately, Docker stack allows users to deploy the full-stack of the services at once.

Let's see changes in our docker-compose file below.

```
version: '3.8'
services:
  web:
    # Path to dockerfile.
    # '.' represents the current directory in which
    # docker-compose.yml is present.
    image: venky8283/flask_app:3.0

    # Mapping of container port to host

    ports:
      - "5000:5000"
    # Mount volume
    volumes:
      - "./:/code"

    networks:
      - app

    deploy:
      replicas: 3

  database:

    # image to fetch from docker hub
    image: mysql/mysql-server:5.7

    # Environment variables for startup script
    # container will use these variables
    # to start the container with these defined variables.
    env_file:
      - ./.env

    # Mount init.sql file to automatically run
    # and create tables for us.
    # everything in docker-entrypoint-initdb.d folder
    # is executed as soon as container is up nd running.
    volumes:
      - "./db/init.sql:/docker-entrypoint-initdb.d/init.sql"

    deploy:
      replicas: 2

    networks:
      - app

  viz:

    # Visualizer image for swarm cluster.

    image: dockersamples/visualizer

    # Mapping container ports to host port

    ports:
      - "8080:8080"


    # Mounting docker socket to container.

    volumes:

      - "/var/run/docker.sock:/var/run/docker.sock"

    networks:
      - app




networks:
  app:
```

- `Line 1:` It is good practice to lock the exact version of Compose we are using.
- `Line 7:` We removed the build keyword and added an image location for the app. 'Build' is not supported in the swarm.
- `Line 17-21:` We added a deploy keyword to inform Docker swarm to maintain three instances of our app service. Also, we mentioned in which network our service will be located in.
- `Line 41-45:` We defined the deployment of database containers along with the network information.
- `Line 47-65:` We Defined the visualizer service. We have defined the image to be pulled from Docker Hub, ports mapping, mounted a Docker socket to sync events and the network to be used by the service.
- `Line 71-73:` We told the Docker swarm to create a network named app so that it can be used by the services.

Docker service also provides in-depth configuration for deployment at CPU level, but we will not make our docker-compose file more complex.

Since our docker-compose file is ready, let's deploy the services on the swarm manager node below.

Type `docker stack deploy <stack name> --compose-file=<location of docker-compose.yml file>`.

```
$ docker stack deploy login-app --compose-file=docker-compose.yml
$ docker stack deploy login_app --compose-file=docker-compose.yml
Creating network login_app_app
Creating service login_app_web
Creating service login_app_database
Creating service login_app_viz
```

This will deploy a stack of all services on swarm manager with just one command.

Now, verify the services using `docker service ls`.

```
$ docker service ls
ID                  NAME                MODE                REPLICAS
            IMAGE                             PORTS
iv1nkq62i6fx        login_app_database   replicated          2/2
                 mysql/mysql-server:5.7
yo7o89dr5s9b        login_app_viz       replicated          1/1
                 dockersamples/visualizer:latest   *:8080->8080/tcp
rpuagx2yj9zg        login_app_web       replicated          3/3
                 venky8283/flask_app:3.0           *:5000->5000/tcp
```

Docker stack commands

Docker stack has few commands to manage stack on the swarm node. Type `docker stack --help` to list them.

```
$ docker stack --help

Usage:  docker stack [OPTIONS] COMMAND

Manage Docker stacks

Options:
      --orchestrator string   Orchestrator to use (swarm|kubernetes|all)

Commands:
  deploy      Deploy a new stack or update an existing stack
  ls          List stacks
  ps          List the tasks in the stack
  rm          Remove one or more stacks
  services    List the services in the stack

Run 'docker stack COMMAND --help' for more information on a command.
```

- `deploy`: We used this one to deploy our stack. We can update the docker-compose file and redeploy stack

- `ls`: This provides a short overview of stack

```
 docker stack ls
NAME                SERVICES            ORCHESTRATOR
login_app           3                   Swarm
```

- `ps`: Lists down all the containers created by the stack

```
$ docker stack ps login_app
ID                  NAME                   IMAGE                             N
ODE                DESIRED STATE       CURRENT STATE              ERROR
            PORTS
zn1c6ro9f1ld        login_app_viz.1        dockersamples/visualizer:latest   d
ocker-desktop      Running             Running 9 minutes ago
sjh64hceb9jh        login_app_database.1   mysql/mysql-server:5.7            d
ocker-desktop      Running             Running 9 minutes ago
ew7zdtczb2su        login_app_web.1        venky8283/flask_app:3.0           d
ocker-desktop      Running             Running 10 minutes ago
luo5j6vjljhv        login_app_database.2   mysql/mysql-server:5.7            d
ocker-desktop      Running             Running 9 minutes ago
ug3v9p7fy7a9        login_app_web.2        venky8283/flask_app:3.0           d
ocker-desktop      Running             Running 10 minutes ago
9v3q0hfmirxc        login_app_web.3        venky8283/flask_app:3.0           d
ocker-desktop      Running             Running 10 minutes ago
```

- `services`: Lists all the services in the stack

```
$ docker stack services login_app
ID                  NAME                 MODE                REPLICAS
            IMAGE                               PORTS
iv1nkq62i6fx        login_app_database   replicated          2/2
                 mysql/mysql-server:5.7
rpuagx2yj9zg        login_app_web        replicated          3/3
                 venky8283/flask_app:3.0           *:5000->5000/tcp
yo7o89dr5s9b        login_app_viz        replicated          1/1
                 dockersamples/visualizer:latest   *:8080->8080/tcp
```

- `rm`: Removes stack from the swarm nodes, including network

```
$ docker stack rm login_app
Removing service login_app_database
Removing service login_app_viz
Removing service login_app_web
Removing network login_app_app
```

I guess that’s enough to get started with stack. You can always explore more as long as you understand the fundamentals.

In the next lesson, we will review all the commands we have used in this section and look at a cheat sheet with us in case we forget some of them.