Docker Compose

Till now we've spent all our time exploring the Docker client. In the Docker ecosystem, however, there are a bunch of other open-source tools which play very nicely with Docker. A few of them are -

1. Docker Machine - Create Docker hosts on your computer, on cloud providers, and inside your own data center
2. Docker Compose - A tool for defining and running multi-container Docker applications.
3. Docker Swarm - A native clustering solution for Docker
4. Kubernetes - Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

In this section, we are going to look at one of these tools, Docker Compose, and see how it can make dealing with multi-container apps easier.

The background story of Docker Compose is quite interesting. Roughly around January 2014, a company called OrchardUp launched a tool called Fig. The idea behind Fig was to make isolated development environments work with Docker. The project was very well received on Hacker News - I oddly remember reading about it but didn't quite get the hang of it.

Compose is a tool that is used for defining and running multi-container Docker apps in an easy way. It provides a configuration file called docker-compose.yml that can be used to bring up an application and the suite of services it depends on with just one command. Compose works in all environments: production, staging, development, testing, as well as CI workflows, although Compose is ideal for development and testing environments.

Let's see if we can create a docker-compose.yml file for our SF-Foodtrucks app and evaluate whether Docker Compose lives up to its promise.

The first step, however, is to install Docker Compose. If you're running Windows or Mac, Docker Compose is already installed as it comes in the Docker Toolbox. Linux users can easily get their hands on Docker Compose by following the instructions on the docs. Since Compose is written in Python, you can also simply do pip install docker-compose. Test your installation with -

$ docker-compose --version

docker-compose version 1.21.2, build a133471

Now that we have it installed, we can jump on the next step i.e. the Docker Compose file docker-compose.yml. The syntax for YAML is quite simple and the repo already contains the docker-compose file that we'll be using.

version: "3"

services:

es:

image: docker.elastic.co/elasticsearch/elasticsearch:6.3.2

container\_name: es

environment:

- discovery.type=single-node

ports:

- 9200:9200

volumes:

- esdata1:/usr/share/elasticsearch/data

web:

image: yourusername/foodtrucks-web

command: python3 app.py

depends\_on:

- es

ports:

- 5000:5000

volumes:

- ./flask-app:/opt/flask-app

volumes:

esdata1:

driver: local

Let me breakdown what the file above means. At the parent level, we define the names of our services - es and web. The image parameter is always required, and for each service that we want Docker to run, we can add additional parameters. For es, we just refer to the elasticsearch image available on Elastic registry. For our Flask app, we refer to the image that we built at the beginning of this section.

Other parameters such as command and ports provide more information about the container. The volumes parameter specifies a mount point in our web container where the code will reside. This is purely optional and is useful if you need access to logs, etc. We'll later see how this can be useful during development. Refer to the online reference to learn more about the parameters this file supports. We also add volumes for the es container so that the data we load persists between restarts. We also specify depends\_on, which tells docker to start the es container before web.

Great! Now the file is ready, let's see docker-compose in action. But before we start, we need to make sure the ports and names are free. So if you have the Flask and ES containers running, lets turn them off.

$ docker stop es foodtrucks-web

es

foodtrucks-web

$ docker rm es foodtrucks-web

es

foodtrucks-web

Now we can run docker-compose. Navigate to the food trucks directory and run docker-compose up.

$ docker-compose up

Creating network "foodtrucks\_default" with the default driver

Creating foodtrucks\_es\_1

Creating foodtrucks\_web\_1

Attaching to foodtrucks\_es\_1, foodtrucks\_web\_1

es\_1 | [2016-01-11 03:43:50,300][INFO ][node ] [Comet] version[2.1.1], pid[1], build[40e2c53/2015-12-15T13:05:55Z]

es\_1 | [2016-01-11 03:43:50,307][INFO ][node ] [Comet] initializing ...

es\_1 | [2016-01-11 03:43:50,366][INFO ][plugins ] [Comet] loaded [], sites []

es\_1 | [2016-01-11 03:43:50,421][INFO ][env ] [Comet] using [1] data paths, mounts [[/usr/share/elasticsearch/data (/dev/sda1)]], net usable\_space [16gb], net total\_space [18.1gb], spins? [possibly], types [ext4]

es\_1 | [2016-01-11 03:43:52,626][INFO ][node ] [Comet] initialized

es\_1 | [2016-01-11 03:43:52,632][INFO ][node ] [Comet] starting ...

es\_1 | [2016-01-11 03:43:52,703][WARN ][common.network ] [Comet] publish address: {0.0.0.0} is a wildcard address, falling back to first non-loopback: {172.17.0.2}

es\_1 | [2016-01-11 03:43:52,704][INFO ][transport ] [Comet] publish\_address {172.17.0.2:9300}, bound\_addresses {[::]:9300}

es\_1 | [2016-01-11 03:43:52,721][INFO ][discovery ] [Comet] elasticsearch/cEk4s7pdQ-evRc9MqS2wqw

es\_1 | [2016-01-11 03:43:55,785][INFO ][cluster.service ] [Comet] new\_master {Comet}{cEk4s7pdQ-evRc9MqS2wqw}{172.17.0.2}{172.17.0.2:9300}, reason: zen-disco-join(elected\_as\_master, [0] joins received)

es\_1 | [2016-01-11 03:43:55,818][WARN ][common.network ] [Comet] publish address: {0.0.0.0} is a wildcard address, falling back to first non-loopback: {172.17.0.2}

es\_1 | [2016-01-11 03:43:55,819][INFO ][http ] [Comet] publish\_address {172.17.0.2:9200}, bound\_addresses {[::]:9200}

es\_1 | [2016-01-11 03:43:55,819][INFO ][node ] [Comet] started

es\_1 | [2016-01-11 03:43:55,826][INFO ][gateway ] [Comet] recovered [0] indices into cluster\_state

es\_1 | [2016-01-11 03:44:01,825][INFO ][cluster.metadata ] [Comet] [sfdata] creating index, cause [auto(index api)], templates [], shards [5]/[1], mappings [truck]

es\_1 | [2016-01-11 03:44:02,373][INFO ][cluster.metadata ] [Comet] [sfdata] update\_mapping [truck]

es\_1 | [2016-01-11 03:44:02,510][INFO ][cluster.metadata ] [Comet] [sfdata] update\_mapping [truck]

es\_1 | [2016-01-11 03:44:02,593][INFO ][cluster.metadata ] [Comet] [sfdata] update\_mapping [truck]

es\_1 | [2016-01-11 03:44:02,708][INFO ][cluster.metadata ] [Comet] [sfdata] update\_mapping [truck]

es\_1 | [2016-01-11 03:44:03,047][INFO ][cluster.metadata ] [Comet] [sfdata] update\_mapping [truck]

web\_1 | \* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

Head over to the IP to see your app live. That was amazing wasn't it? Just a few lines of configuration and we have two Docker containers running successfully in unison. Let's stop the services and re-run in detached mode.

web\_1 | \* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

Killing foodtrucks\_web\_1 ... done

Killing foodtrucks\_es\_1 ... done

$ docker-compose up -d

Creating es ... done

Creating foodtrucks\_web\_1 ... done

$ docker-compose ps

Name Command State Ports

--------------------------------------------------------------------------------------------

es /usr/local/bin/docker-entr ... Up 0.0.0.0:9200->9200/tcp, 9300/tcp

foodtrucks\_web\_1 python3 app.py Up 0.0.0.0:5000->5000/tcp

Unsurprisingly, we can see both the containers running successfully. Where do the names come from? Those were created automatically by Compose. But does *Compose* also create the network automatically? Good question! Let's find out.

First off, let us stop the services from running. We can always bring them back up in just one command. Data volumes will persist, so it’s possible to start the cluster again with the same data using docker-compose up. To destroy the cluster and the data volumes, just type docker-compose down -v.

$ docker-compose down -v

Stopping foodtrucks\_web\_1 ... done

Stopping es ... done

Removing foodtrucks\_web\_1 ... done

Removing es ... done

Removing network foodtrucks\_default

Removing volume foodtrucks\_esdata1

While we're are at it, we'll also remove the foodtrucks network that we created last time.

$ docker network rm foodtrucks-net

$ docker network ls

NETWORK ID NAME DRIVER SCOPE

c2c695315b3a bridge bridge local

a875bec5d6fd host host local

ead0e804a67b none null local

Great! Now that we have a clean slate, let's re-run our services and see if *Compose* does its magic.

$ docker-compose up -d

Recreating foodtrucks\_es\_1

Recreating foodtrucks\_web\_1

$ docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

f50bb33a3242 yourusername/foodtrucks-web "python3 app.py" 14 seconds ago Up 13 seconds 0.0.0.0:5000->5000/tcp foodtrucks\_web\_1

e299ceeb4caa elasticsearch "/docker-entrypoint.s" 14 seconds ago Up 14 seconds 9200/tcp, 9300/tcp foodtrucks\_es\_1

So far, so good. Time to see if any networks were created.

$ docker network ls

NETWORK ID NAME DRIVER

c2c695315b3a bridge bridge local

f3b80f381ed3 foodtrucks\_default bridge local

a875bec5d6fd host host local

ead0e804a67b none null local

You can see that compose went ahead and created a new network called foodtrucks\_default and attached both the new services in that network so that each of these are discoverable to the other. Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by them at a hostname identical to the container name.

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

8c6bb7e818ec docker.elastic.co/elasticsearch/elasticsearch:6.3.2 "/usr/local/bin/dock…" About a minute ago Up About a minute 0.0.0.0:9200->9200/tcp, 9300/tcp es

7640cec7feb7 yourusername/foodtrucks-web "python3 app.py" About a minute ago Up About a minute 0.0.0.0:5000->5000/tcp foodtrucks\_web\_1

$ docker network inspect foodtrucks\_default

[

{

"Name": "foodtrucks\_default",

"Id": "f3b80f381ed3e03b3d5e605e42c4a576e32d38ba24399e963d7dad848b3b4fe7",

"Created": "2018-07-30T03:36:06.0384826Z",

"Scope": "local",

"Driver": "bridge",

"EnableIPv6": false,

"IPAM": {

"Driver": "default",

"Options": null,

"Config": [

{

"Subnet": "172.19.0.0/16",

"Gateway": "172.19.0.1"

}

]

},

"Internal": false,

"Attachable": true,

"Ingress": false,

"ConfigFrom": {

"Network": ""

},

"ConfigOnly": false,

"Containers": {

"7640cec7feb7f5615eaac376271a93fb8bab2ce54c7257256bf16716e05c65a5": {

"Name": "foodtrucks\_web\_1",

"EndpointID": "b1aa3e735402abafea3edfbba605eb4617f81d94f1b5f8fcc566a874660a0266",

"MacAddress": "02:42:ac:13:00:02",

"IPv4Address": "172.19.0.2/16",

"IPv6Address": ""

},

"8c6bb7e818ec1f88c37f375c18f00beb030b31f4b10aee5a0952aad753314b57": {

"Name": "es",

"EndpointID": "649b3567d38e5e6f03fa6c004a4302508c14a5f2ac086ee6dcf13ddef936de7b",

"MacAddress": "02:42:ac:13:00:03",

"IPv4Address": "172.19.0.3/16",

"IPv6Address": ""

}

},

"Options": {},

"Labels": {

"com.docker.compose.network": "default",

"com.docker.compose.project": "foodtrucks",

"com.docker.compose.version": "1.21.2"

}

}

]