Create A Compose File

create a Docker Compose file to programmatically deploy an application with code. . A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Compose is a tool for defining and running multi-container applications with Docker. With Compose, you define a multi-container application in a single file, then spin your application up in a single command which does everything that needs to be done to get it running.

Lab setup

Same as Lab 1.

Verify that you have the following software installed:

1. A 64-bit OS

2. [Docker Toolbox](https://www.docker.com/toolbox)

Docker Compose is included in the Docker Toolbox.

## Create an Application

We will follow the steps as listed on [Docker Compose Quickstart](https://docs.docker.com/compose/)

Create a new directory to host the compose file called ```composetest```.

Inside this directory, create ```app.py```, a simple web app that uses the Flask framework and increments a value in Redis. You don't need Python installed since we are going to run the app inside a container later.

Copy and paste the following into `app.py` and save the file:

```py

from flask import Flask

from redis import Redis

app = Flask(\_\_name\_\_)

redis = Redis(host='redis', port=6379)

@app.route('/')

def hello():

redis.incr('hits')

return 'Hello World! I have been seen %s times.' % redis.get('hits')

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", debug=True)

```

Next, define the Python dependencies in a file called `requirements.txt`:

```

flask

redis

```

Copy and paste the following into `Dockerfile`:

```

FROM python:2.7

ADD . /code

WORKDIR /code

RUN pip install -r requirements.txt

CMD python app.py

Next, define a set of services using `docker-compose.yml`. The `docker-compose.yml` file defines two containers `web` and `redis` that are connected via specifying the `links` parameter. The content of the container is specified in two ways. First int the `web` container, it is told to `build: .` which ensures the `Dockerfile` specified above with the Python app is built. The `redis` container specifies `image: redis` which ensures an official image of `redis` is pulled. The `web` container has a `ports:` parameter specified which exposes the `5000` port to the outside world. The last portion is related to the exposing this working directory to the `web` container, this is done through the `volumes` parameter with `.:/code`. Check out all the other attributes at the [docker-compose.yml reference](https://docs.docker.com/compose/yml/).

Copy and paste:

```

web:

build: .

ports:

- "5000:5000"

volumes:

- .:/code

links:

- redis

redis:

image: redis

```

Now lets run it!:

```

$ docker-compose up

```

You are presented with the logging output of both containers. Access the application by going to `http://PUBLICIP:5000` Refresh the page and watch the log.

Now, we can't render our host useless by looking at logs. So stop the container by pressing `CTRL+c`. A `docker ps` will show that these containers have been stopped. Bring them up again in a detached state.:

```

docker-compose up -d

```

Go back to your web application at `http://PUBLICIP:5000` and you'll see that it's all working.