Agenda.

1. Pre-requisites
2. Install Docker Compose on Centos
3. Create the example application
4. Test the app with Compose

Prerequisites

Ensure that you met the following prerequisites before continuing with this tutorial:

- Logged in as a **user with sudo privileges**.
- Have **Docker installed on your CentOS 7** system.

Install Docker Compose on CentOS

The recommended method for installing Docker Compose on CentOS 7 is by downloading the Compose binary from the Docker's GitHub repository.

At the time of writing this article, the latest stable version of Docker Compose is version 1.23.1. Before downloading the Compose binary visit the Compose repository release page on GitHuband check if there is a new version available for download.

Complete the following steps to install Docker Compose on CentOS 7:

1. Start by downloading the Docker Compose binary into the /usr/local/bin directory using the following curl command:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Once the download is complete, make the binary executable by typing:

```
sudo chmod +x /usr/local/bin/docker-compose
```

3. To verify the installation type the following command to print the Compose version:

```
docker-compose --version
```

The output will look something like this:

```
docker-compose version 1.23.1, build b02f1306
```

Create the example application

The app used in this guide is based on the hit counter app in the Get started with Docker Compose guide. It consists of a Python app which maintains a counter in a Redis instance and increments the counter whenever you visit it.

1. Create a directory for the project:

```
$ mkdir stackdemo
$ cd stackdemo
```

2. Create a file called `app.py` in the project directory and paste this in:

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! I have been seen {} times.\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, debug=True)
```

3. Create a file called `requirements.txt` and paste these two lines in:

```
flask
redis
```

4. Create a file called `Dockerfile` and paste this in:

```
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
```

```
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

5. Create a file called `docker-compose.yml` and paste this in:

```
version: '3'

services:
  web:
    image: vishwacloudlab/stackdemo
    build: .
    ports:
      - "8000:8000"
  redis:
    image: redis:alpine
```

The image for the web app is built using the Dockerfile defined above. It's also tagged with `127.0.0.1:5000` - the address of the registry created earlier. This is important when distributing the app to the swarm.

Test the app with Compose

1. Start the app with `docker-compose up`. This builds the web app image, pulls the Redis image if you don't already have it, and creates two containers.

   You see a warning about the Engine being in swarm mode. This is because Compose doesn't take advantage of swarm mode, and deploys everything to a single node. You can safely ignore this.

```
$ 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 are scheduled on the current node.

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

Creating network "stackdemo_default" with the default driver
Building web
...(build output)...
Creating stackdemo_redis_1
Creating stackdemo_web_1
```

2. Check that the app is running with `docker-compose ps`:

```
$ docker-compose ps

      Name                     Command               State           Ports
-------------------------------------------------------------------------------
stackdemo_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
stackdemo_web_1     python app.py                    Up      0.0.0.0:8000-
>8000/tcp
```

   You can test the app with `curl`:

```
$ curl http://localhost:8000
Hello World! I have been seen 1 times.

$ curl http://localhost:8000
Hello World! I have been seen 2 times.

$ curl http://localhost:8000
Hello World! I have been seen 3 times.
```

9. Bring the app down:

```
$ docker-compose down --volumes

Stopping stackdemo_web_1 ... done
Stopping stackdemo_redis_1 ... done
Removing stackdemo_web_1 ... done
Removing stackdemo_redis_1 ... done
Removing network stackdemo_default
```

Scaling of the containers

Now that we have to scale the containers.

We would need to change the docker compose file such that we can map multiple “web” containers to the port on the host machine.

```
version: '3'
services:
  web:
    image: vishwacloudlab/stackdemo
    build: .
    ports:
      - "8000-8010:8000"
  redis:
    image: redis:alpine
```

```
[root@linux-client stackdemo]# docker-compose scale web=3
```

```
[root@linux-client stackdemo]# docker-compose scale web=3
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
WARNING: The "web" service specifies a port on the host. If multiple containers for this service are created on
, the port will clash.
Starting stackdemo_web_1_51c4df2b9c68 ... done
Creating stackdemo_web_2_d2d0a52347f9 ... done
Creating stackdemo_web_3_45c0b903f569 ... done
[root@linux-client stackdemo]#

[root@linux-client stackdemo]# docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS
          NAMES
b15aaa351b5a        vishwacloudlab/stackdemo   "python app.py"          47 seconds ago      Up 43 seconds       0.0.0.0:8002-
>8000/tcp   stackdemo_web_2_9a54ba9b8eab
4ab7739673ba        vishwacloudlab/stackdemo   "python app.py"          47 seconds ago      Up 44 seconds       0.0.0.0:8001-
>8000/tcp   stackdemo_web_3_e881ac47be36
c514cf977086        redis:alpine             "docker-entrypoint..."   41 minutes ago      Up 41 minutes       6379/tcp
          stackdemo_redis_1_a1fbb8dc52ce
3a73a0942ddc        vishwacloudlab/stackdemo   "python app.py"          41 minutes ago      Up 41 minutes       0.0.0.0:8000-
>8000/tcp   stackdemo_web_1_51c4df2b9c68
[root@linux-client stackdemo]#
```

Output

Try the output with

http:// <linux-client>:8000

http:// <linux-client>:8001

http:// <linux-client>:8002