Objective

Run an Python-Redis app, and scale, Observe the way docker swarm orchestrates it.

Steps

1. Create the example app and test it with docker compose
2. Push the image to the hub.docker.com
3. Deploy it via the docker swarm
4. Chk the output of the app and troubleshoot it
5. Scale the app and chk the output
6. To remove the App Env and Cluster

1.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:
   Make sure the below “intendation” are followed as the Python file is very sensitive to this.
   If the copy paste did not, take some effort and rectify it below proceeding further.

```
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 `vishwacloudlab`.

Later we would be pushing the docker image to central repository , so that all the nodes in the docker swarm cluster can use it.

Note: -- I'm pushing it to my docker hub ID. Make sure you put your ID there.

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.
```

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
```

2.Push the generated image to the central repo

Before you push the image to the remote registry, login to the hub.docker.com on the manager nodes.

$ docker push vishwacloudlab/stackdemo

To distribute the web app's image across the swarm, it needs to be pushed to the registry you set up earlier. With Compose, this is very simple:

```
$ docker-compose push


Pushing web (vishwacloudlab/stackdemo:latest)...
The push refers to a repository [vishwacloudlab/stackdemo]
5b5a49501a76: Pushed
be44185ce609: Pushed
bd7330a79bcf: Pushed
c9fc143a069a: Pushed
011b303988d2: Pushed
latest: digest:
sha256:a81840ebf5ac24b42c1c676cbda3b2cb144580ee347c07e1bc80e35e5ca76507 size: 1372
```

The stack is now ready to be deployed.

3.Deploy the stack to the swarm

1. Create the stack with `docker stack deploy`:

```
$ docker stack deploy --compose-file docker-compose.yml stackdemo01


Ignoring unsupported options: build


Creating network stackdemo_default
Creating service stackdemo_web
Creating service stackdemo_redis
```

The last argument is a name for the stack. Each network, volume and service name is prefixed with the stack name.

2. Check that it's running with `docker stack services stackdemo`:

```
$ docker stack services stackdemo
```

```
[root@docker-mas01 stackdemo]# docker stack services stackdemo01
ID            NAME                MODE         REPLICAS  IMAGE
9h00ax3fcjlv  stackdemo01_redis   replicated   1/1       redis:alpine
sqgpaevy2yud  stackdemo01_web     replicated   1/1       vishwacloudlab/python-custb16:latest
[root@docker-mas01 stackdemo]#
```

Once it's running, you should see `1/1` under `REPLICAS` for both services. This might take some time if you have a multi-node swarm, as images need to be pulled.

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.213.136  netmask 255.255.255.0  broadcas
        inet6 fe80::db92:abb5:690c:23a9  prefixlen 64  scopei
        ether 00:0c:29:12:d2:cf  txqueuelen 1000  (Ethernet)
        RX packets 53358  bytes 6550745 (6.2 MiB)
```

Use the ip of the underline Host VM on which the "web" is running.

```
$ docker stack ps stackdemo01
```

```
[root@docker-mas01 stackdemo]# docker stack ps stackdemo01
ID            NAME                 IMAGE                                NODE          DESIRED STATE  CURRENT
TATE           ERROR  PORTS
bdyiu8x6zcad  stackdemo01_web.1    vishwacloudlab/python-custb16:latest  docker-mas01  Running        Running
3 minutes ago
115m013lyu2p  stackdemo01_redis.1  redis:alpine                         docker-mas02  Running        Running
3 minutes ago
[root@docker-mas01 stackdemo]#
```

The above command, shows which container is running on which NODE.

Here, **"web"** container is running **"docker-mas01"** node and
"redis" container is running **"docker-mas02"** node

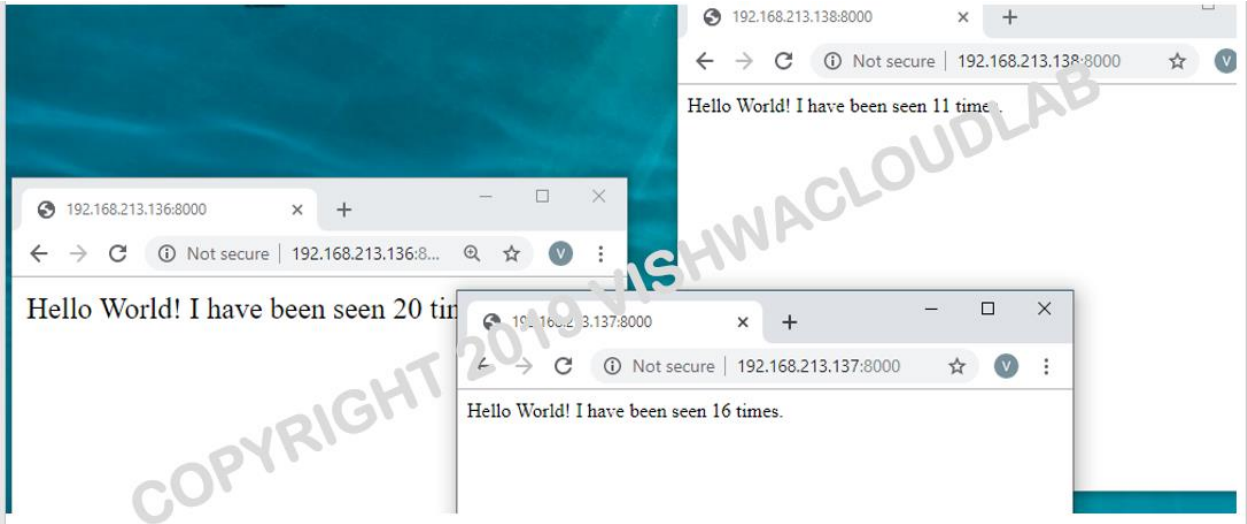
As before, you can test the app with `curl`:

```
$ curl http://192.168.213.136:8000
Hello World! I have been seen 1 times.

$ curl http://192.168.213.136:8000
Hello World! I have been seen 2 times.

$ curl http://192.168.213.136:8000
Hello World! I have been seen 3 times.
```

Thanks to Docker's built-in routing mesh, you can access any node in the swarm on port 8000 and get routed to the app:

```
$ curl http://address-of-other-node:8000
Hello World! I have been seen 4 times.
```

4.Chk the output of the app and troubleshoot it



Here I hve used all the ip address of all the NODES in the cluster.

Troubleshooting:

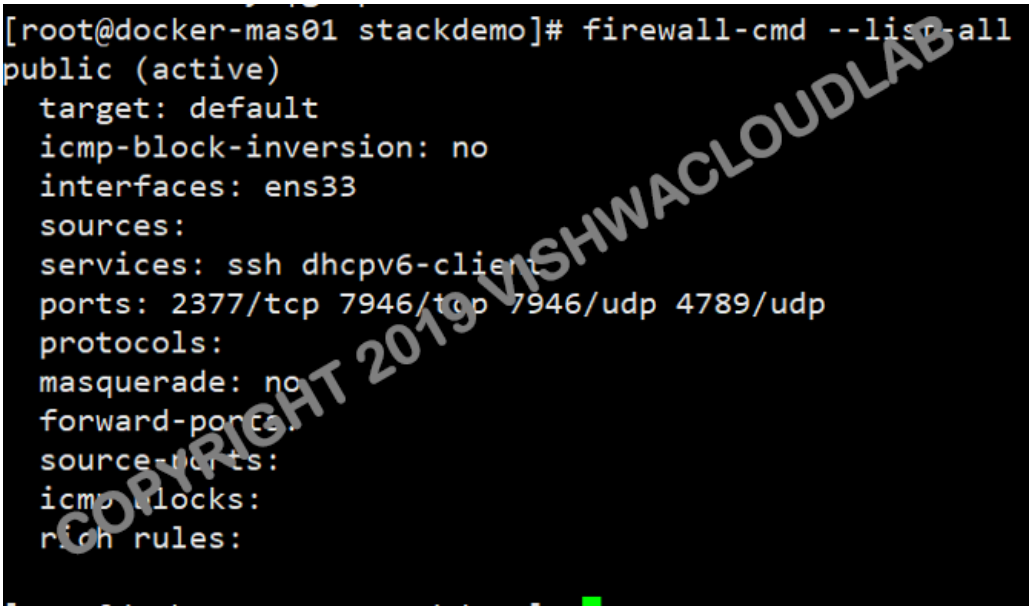
IF the output is not coming up and gives an REDIS error, which means the container in Master node01 is not able to communicate to the **redis** container in the "**Master-node02"**.
To solve this problem,

We hve to allow, overlay communication between all the NODES in the cluster.
Open the below ports on all the NODES in the cluster.

1. Port 7946 TCP/UDP for container network discovery
2. Port 4789 UDP for the container overlay network

Commands to be executed on all the VM's are

```
$ firewall-cmd --zone=public --permanent --add-port=7946/tcp
$ firewall-cmd --zone=public --permanent --add-port=7946/udp
$ firewall-cmd --zone=public --permanent --add-port=4789/tcp

$ firewall-cmd --reload
```


[root@docker-mas01 stackdemo]# firewall-cmd --list-all
public (active)
target: default
icmp-block-inversion: no
interfaces: ens33
sources:
services: ssh dhcpv6-client
ports: 2377/tcp 7946/tcp 7946/udp 4789/udp
protocols:
masquerade: no
forward-ports:
source-ports:
icmp-blocks:
rich rules:
COPYRIGHT 2019 VISHWACLOUDLAB

5.Scale the app and chk the output

Check the service details and we would want to scale only the **web** service

```
$ docker stack services stackdemo01
```

```
[root@docker-mas01 stackdemo]# docker stack services  stackdemo01
ID            NAME                MODE         REPLICAS  IMAGE
9h00ax3fcjlv  stackdemo01_redis    replicated   1/1       redis:alpine
sqgpaevy2yud  stackdemo01_web      replicated   1/1       vishwacloudlab/python-custb16:latest
[root@docker-mas01 stackdemo]#
```

Now, lets scale the web service.

```
$ docker service scale stackdemo01_web=4
```

```
[root@docker-mas01 stackdemo]# docker service scale stackdemo01_web=4
stackdemo01_web scaled to 4
```

```
. $ docker stack services stackdemo01
```

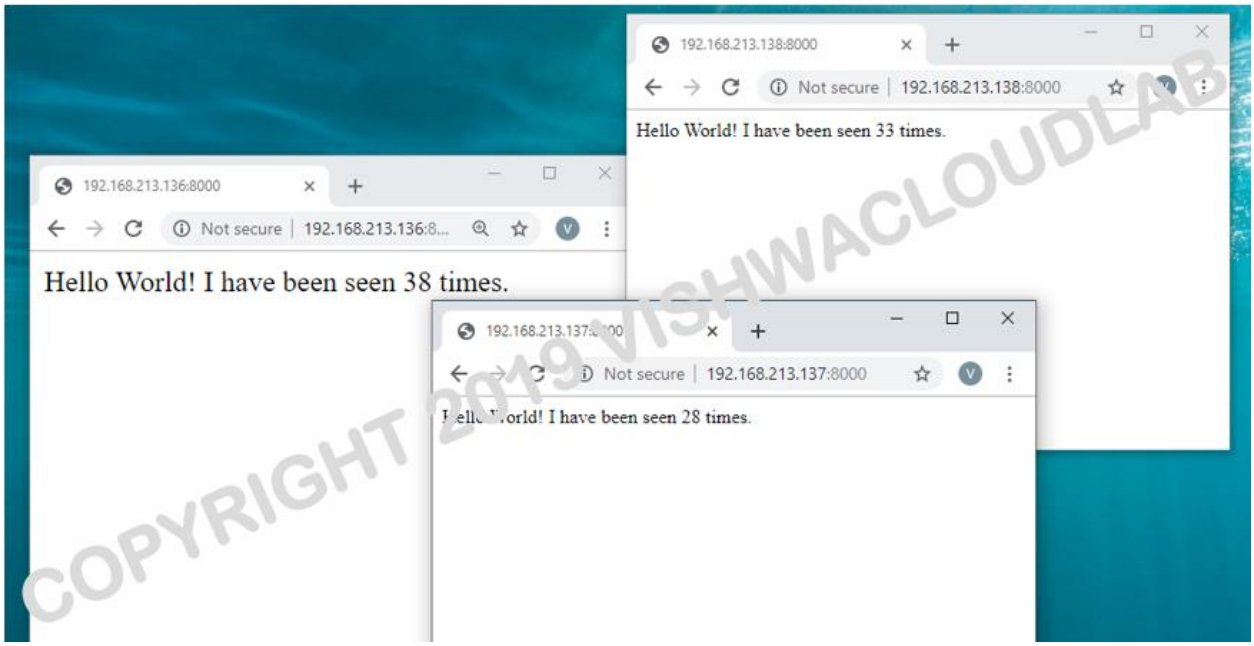
```
[root@docker-mas01 stackdemo]# docker stack services  stackdemo01
ID            NAME                MODE         REPLICAS  IMAGE
9h00ax3fcjlv  stackdemo01_redis    replicated   1/1       redis:alpine
sqgpaevy2yud  stackdemo01_web      replicated   4/4       vishwacloudlab/python-custb16:latest
[root@docker-mas01 stackdemo]#
```

```
. $ docker stack ps stackdemo01
```

```
[root@docker-mas01 stackdemo]# docker stack ps stackdemo01
ID            NAME                  IMAGE                                  NODE           DESIRED STATE  CURRENT
 STATE             ERROR  PORTS
bdyiu8x6zcad  stackdemo01_web.1      vishwacloudlab/python-custb16:latest   docker-mas01   Running        Running
 about an hour ago
115m0131yu2p  stackdemo01_redis.1    redis:alpine                           docker-mas02   Running        Running
 about an hour ago
pdsf3sapm4m3  stackdemo01_web.2      vishwacloudlab/python-custb16:latest   docker-wnode01 Running        Running
 2 seconds ago
n3j1yyfk94dc  stackdemo01_web.3      vishwacloudlab/python-custb16:latest   docker-mas02   Running        Running
 3 seconds ago
pnikqsm98g6o  stackdemo01_web.4      vishwacloudlab/python-custb16:latest   docker-mas01   Running        Running
 3 seconds ago
[root@docker-mas01 stackdemo]#
```


Lets check the output now.

The front end, really no changes, We can still access the app, using any of the IP address of the NODES in the cluster.



6.Remove the stack and cluster

1. Bring the stack down with `docker stack rm`:

```
$ docker stack rm stackdemo01


Removing service stackdemo_web
Removing service stackdemo_redis
Removing network stackdemo_default
```

2. If you're just testing things out on a local machine and want to bring your Docker Engine out of swarm mode, use `docker swarm leave`:

```
$ docker swarm leave --force
Node left the swarm.
```