Report

Kubernetes, Docker Swarm & Microservices.

INFO 7390

Advance Data Science & Architecture

Professor: Srikanth Krishnamurthy

By: Team 5

Akash Jagtap

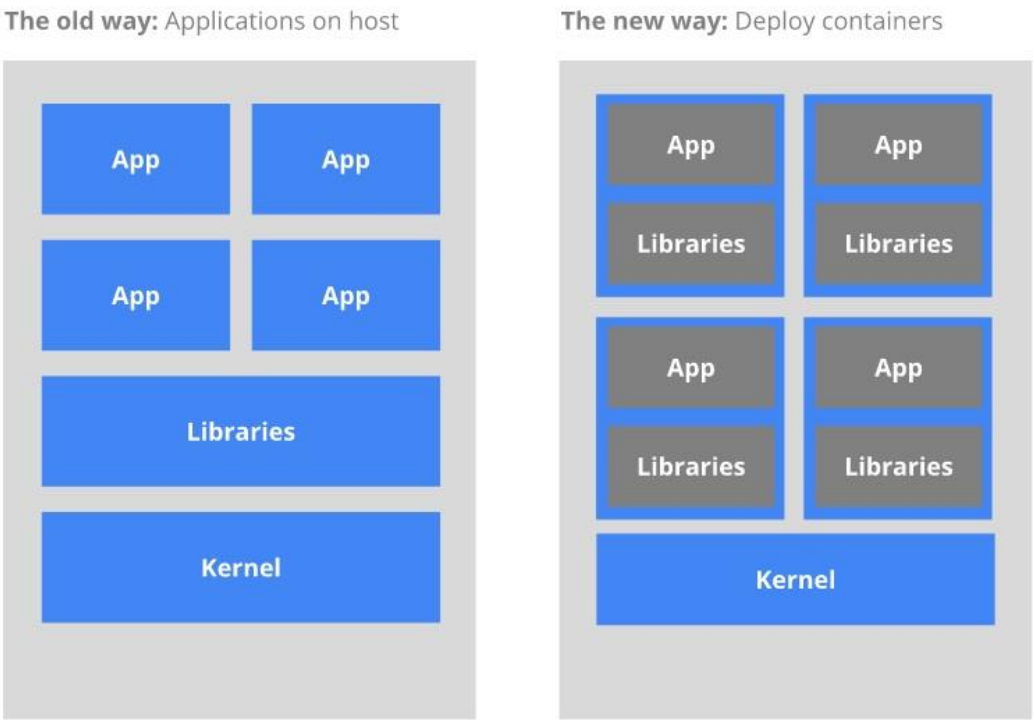
Jerin Rajan

Nitin Prince Reuben

By this report, we can learn about online hosting applications like Kubernetes, docker swarm and basic concepts of microservices.

Kubernetes.

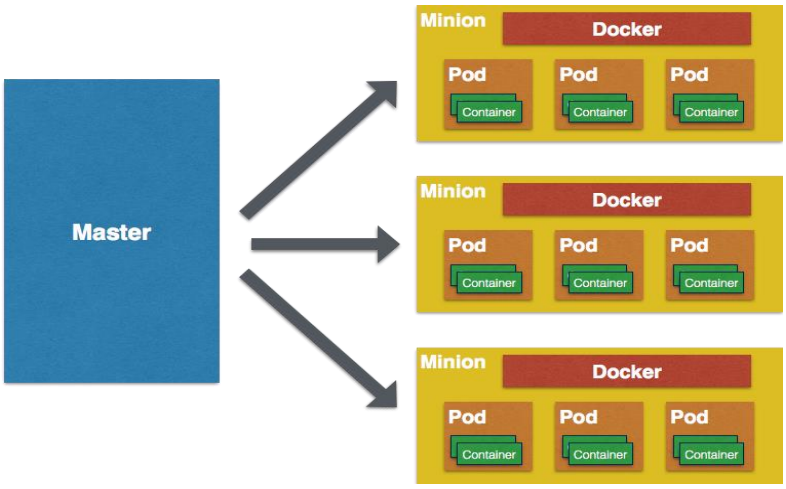
Kubernetes, the word is derived from Greek which means ‘helmsman’ or pilot of a ship. It is an open source project that was started by Google and derived from Borg, which is used inside Google for several years now, for container management, automating deployment and scaling. Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines. It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior. Currently, it is hosted by Cloud Native Computing Foundation(CNCF). There are lot of things along with kubernetes which is being hosted by CNCF for various purposes. Major ones are Prometheus for monitoring and CNI for networking API. They are trying to gather as many software/applications as possible which can provide smooth transition for any company towards cloud business. You can learn more about these software from https://www.cncf.io/.



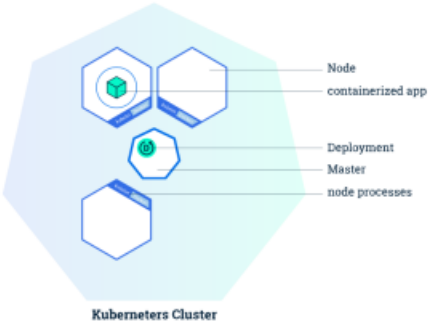
By this picture, we can understand the difference between older and new technique of deploying the applications. Earlier, the containers to which the applications are being deployed has some libraries already present in it. That had few outcomes. Not all applications have same requirements. Some preferred older libraries and some preferred new. This created lots of problems. This made people change the deployment technique. Now the libraries which are required by the applications are being deployed with it. This decreases the problem of compatibility and increased the performance and speed. This also made upgradations easier.

Kubernetes have lots of features. Some of them are as follows:

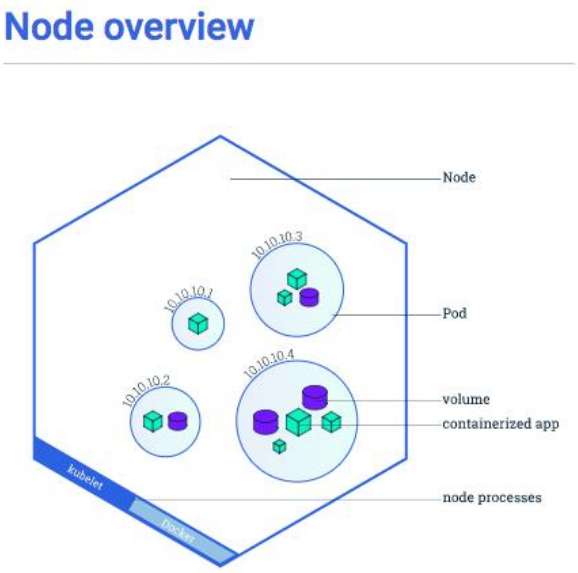
- Does not limit the types of application supported – It supports wide varieties of application and this make it preferable over other deployment application
- Load Balancing – if the traffic to the network is more, it helps has to balance the load easily.
- Rolling update or rollback – it maintains logs. So, it is easy to roll back to a previous version or to make an update easy.
- Auto – scalability
- Using pods for grouping container
- API management – It provides lots API to work upon. All these API will be disabled at the beginning. We can start using the ones which we need by enabling them. And can later disable them after we are done using it as google charges money for using these API depending upon the time of usage. So, we don't want to keep unused API enabled. Also, google thinks that by disabling the APIs, it can increase security as people can attack your account to steal the code or cause a problem to your application by using the APIs which we are not using.



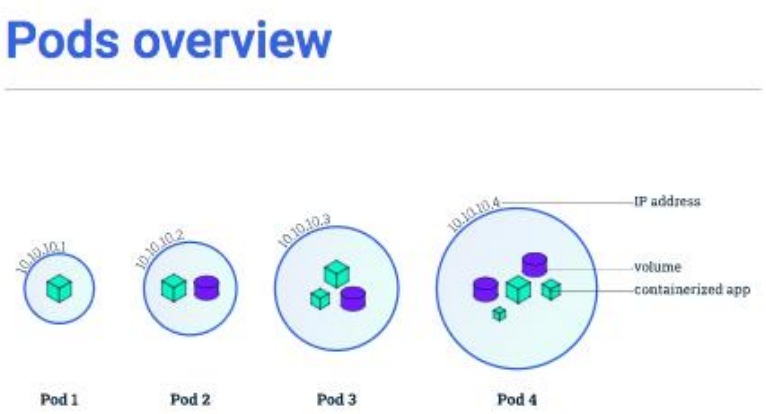
The above picture describes the architecture of Kuberneters. For a clearer, understanding, we would check next image.



Kubernetes is made up of clusters. Each cluster has a master surrounded by nodes.

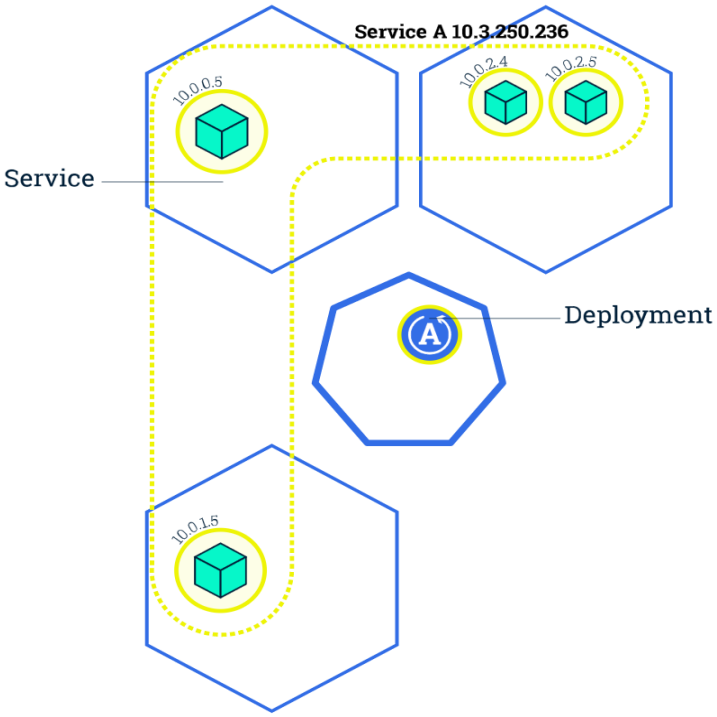


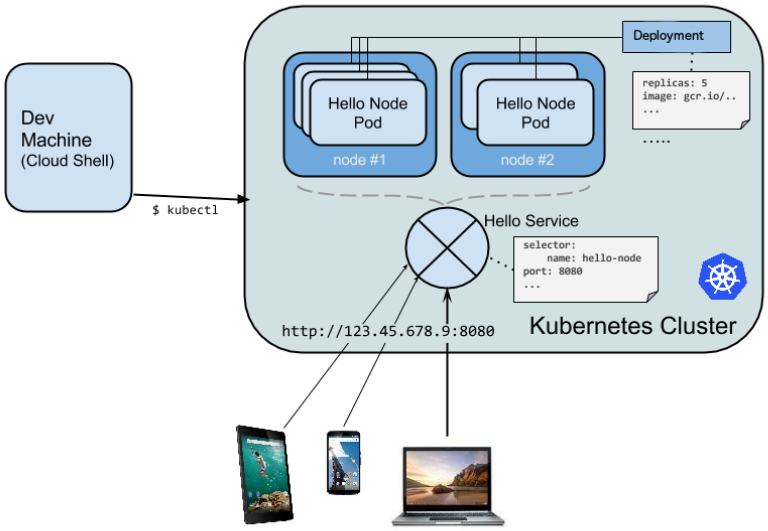
Each node consists of lots of pods inside which the application is deployed and every pod have it’s set of libraries already present in it.



The basic scheduling unit in Kubernetes is called a "pod". It adds a higher level of abstraction to containerized components. A pod consists of one or more containers that are guaranteed to be co-located on the host machine and can share resources. Each pod in Kubernetes is assigned a unique (within the cluster) IP address, which allows applications to use ports without the risk of conflict.] A pod can define a volume, such as a local disk directory or a network disk, and expose it to the containers in the pod. Pods can be managed manually through the Kubernetes API, or their management can be delegated to a controller.

A Kubernetes service is a set of pods that work together, such as one tier of a multi-tier application. The set of pods that constitute a service are defined by a label selector.] Kubernetes provides service discovery and request routing by assigning a stable IP address and DNS name to the service, and load balances traffic in a round-robin manner to network connections of that IP address among the pods matching the selector (even as failures cause the pods to move from machine to machine). By default a service is exposed inside a cluster (e.g. back end pods might be grouped into a service, with requests from the front-end pods load-balanced among them), but a service can also be exposed outside a cluster (e.g. for clients to reach frontend pods).





With the above picture, we can understand the business prospective of Kubernetes. Through our developer machine, we transfer the code. *Kubectl* is the function which helps google SDK to communicate with kubernetes. With the help of auto-scaling, the code is distributed and we are provided with an IP address with which we can access the application in our devices.

Deploying a Simple Hello World App on Kubernetes.

Technology used are: Docker, Python, Flask, Kubernetes, Google Cloud Platform

1. Create a Google Cloud Account
2. Install gcloud SDK (client cli)
3. After installing SDK there are two ways to setups kubernetes
    a. Minicube – Single Node Kubernetes
        i. For installation – https://www.youtube.com/watch?v=_vHTaIJm9uY
4. Kubernetes multi-node cluster can be performed on AWS or GCP.
   We would be proceeding with GCP i.e. Google Cloud Platform.
5. Commands to execute are :
        i. gcloud components list

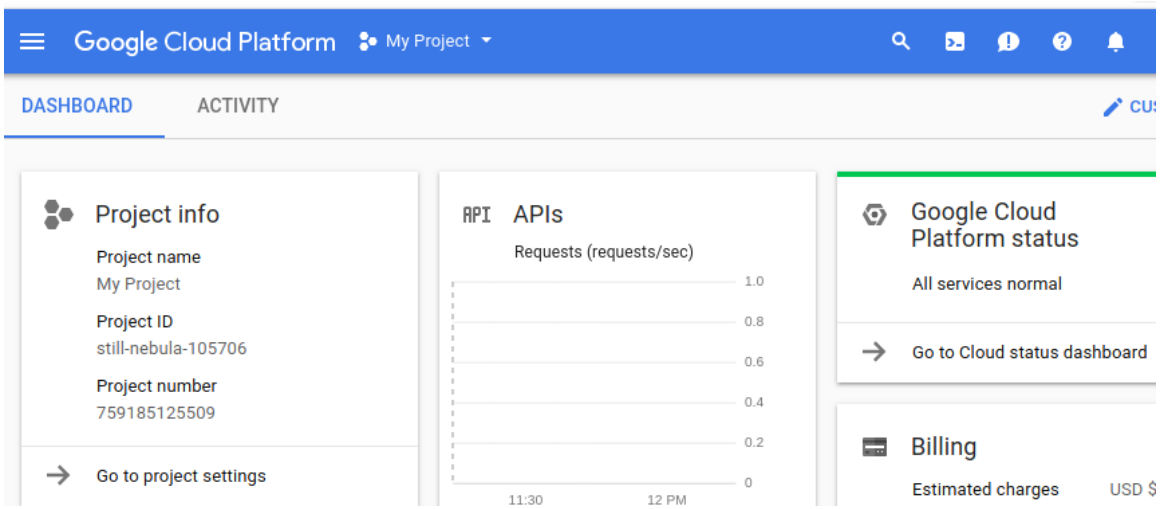
```
frivolouspantheon jerry # gcloud components list

Your current Cloud SDK version is: 188.0.1
The latest available version is: 189.0.0

                                                  Components
┌──────────────────┬──────────────────────────────────────────────────────┬──────────────────────────┬───────────┐
│      Status      │                         Name                         │            ID            │    Size   │
├──────────────────┼──────────────────────────────────────────────────────┼──────────────────────────┼───────────┤
│ Update Available │ BigQuery Command Line Tool                           │ bq                       │   < 1 MiB │
│ Update Available │ Cloud SDK Core Libraries                             │ core                     │   7.2 MiB │
│ Update Available │ Cloud Storage Command Line Tool                      │ gsutil                   │   3.3 MiB │
│ Not Installed    │ App Engine Go Extensions                             │ app-engine-go            │  98.1 MiB │
│ Not Installed    │ Cloud Bigtable Command Line Tool                     │ cbt                      │   4.5 MiB │
│ Not Installed    │ Cloud Bigtable Emulator                              │ bigtable                 │   3.7 MiB │
│ Not Installed    │ Cloud Datalab Command Line Tool                      │ datalab                  │   < 1 MiB │
│ Not Installed    │ Cloud Datastore Emulator                             │ cloud-datastore-emulator │  17.9 MiB │
│ Not Installed    │ Cloud Datastore Emulator (Legacy)                    │ gcd-emulator             │  38.1 MiB │
│ Not Installed    │ Cloud Pub/Sub Emulator                               │ pubsub-emulator          │  33.4 MiB │
│ Not Installed    │ Emulator Reverse Proxy                               │ emulator-reverse-proxy   │  14.5 MiB │
│ Not Installed    │ gcloud Alpha Commands                                │ alpha                    │   < 1 MiB │
│ Not Installed    │ gcloud Beta Commands                                 │ beta                     │   < 1 MiB │
│ Not Installed    │ gcloud app Java Extensions                           │ app-engine-java          │ 118.8 MiB │
│ Not Installed    │ gcloud app PHP Extensions                            │ app-engine-php           │           │
│ Not Installed    │ gcloud app Python Extensions                         │ app-engine-python        │   6.1 MiB │
│ Not Installed    │ gcloud app Python Extensions (Extra Libraries)       │ app-engine-python-extras │  27.8 MiB │
│ Installed        │ Google Container Local Builder                       │ container-builder-local  │   3.8 MiB │
│ Installed        │ Google Container Registry's Docker credential helper │ docker-credential-gcr    │   2.8 MiB │
│ Installed        │ kubectl                                              │ kubectl                  │  12.3 MiB │
└──────────────────┴──────────────────────────────────────────────────────┴──────────────────────────┴───────────┘
To install or remove components at your current SDK version [188.0.1], run:
  $ gcloud components install COMPONENT_ID
  $ gcloud components remove COMPONENT_ID

To update your SDK installation to the latest version [189.0.0], run:
  $ gcloud components update
```

        ii. If kubectl is not installed then install it using the command

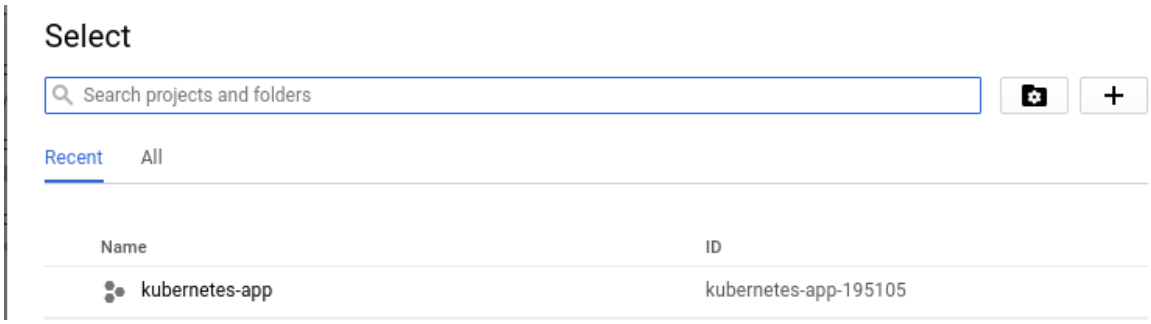
            gcloud component install kubectl

Ref:https://cloud.google.com/sdk/gcloud/reference/components/

6. Login with CLI:
    a. gcloud auth login
    b. Enter your username and password
    c. Go to your gcloud console and create a project.



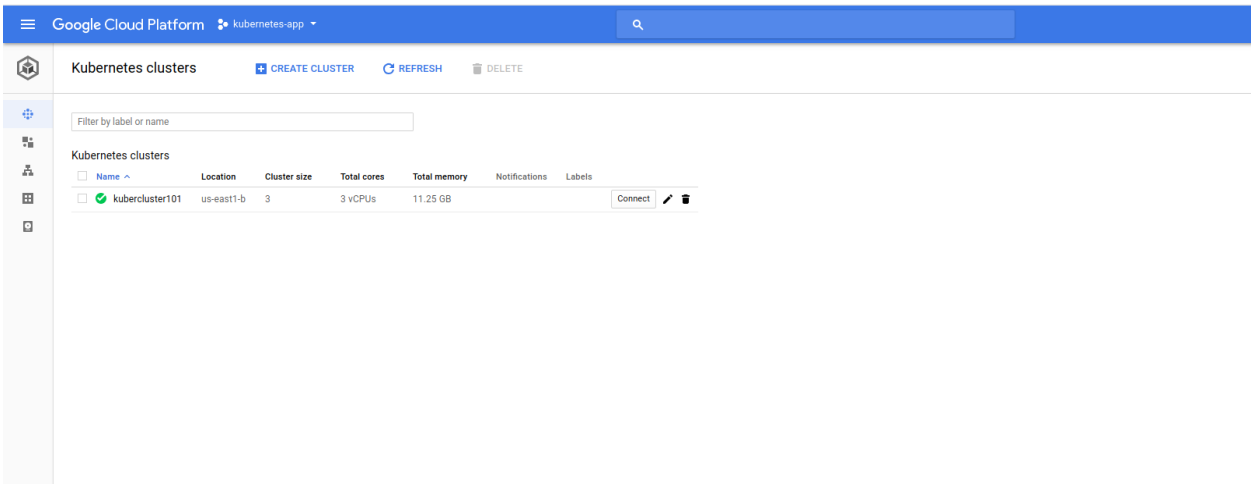
Next, Click + sign And assign your project name

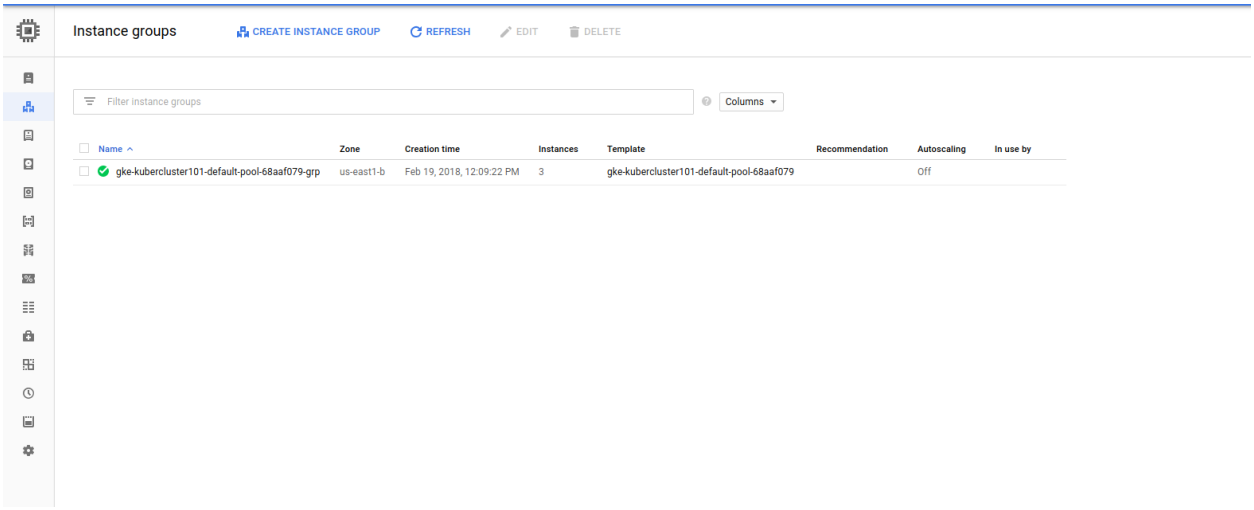


7. Goto the CLI
    a. gcloud config set project[PROJECT_ID]
    b. gcloud config set compute/zone [COMPUTE ZONE]
    OR
    c. gcloud init
        i. Enter Option 1.
        ii. Login with your username
        iii. Select the project you just created.
        iv. Select the configure zone.
8. Create Kubernetes Cluster

A cluster consists of at least one cluster master machine and multiple machines called nodes. Nodes are Compute Engine Virtual Machine(VM) instances that run the Kubernetes processes necessary to make them part of the cluster. You deploy applications to clustersm and the applications run on the nodes

Command to create cluster is : gcloud container clusters create kubecluster101





Ref: https://kubernetes.io/docs/setup/pick-right-solution/#hosted-solutions

9. Push Docker images to Container Registry
    a. Create a Docker File

```
# Import Python runtime and set up working directory
FROM python:2.7-slim
WORKDIR /app
ADD . /app
# Install any necessary dependencies
RUN pip install -r requirements.txt
# Open port 80 for serving the webpage
EXPOSE 80
# Run app.py when the container launches
CMD ["python", "app.py"]
```

b. Create requirements.txt << Flask

c. Create app.py

```
from flask import Flask
import os
import socket
app = Flask(__name__)
@app.route("/")
def hello():
        html = "<h3>Hello, World!</h3>"
        return html
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

```
Kubernetes # vi Dockerfile
Kubernetes # vi requirements.txt
Kubernetes # vi app.py
Kubernetes #
```

d. Build docker images

```
frivolouspantheon Kubernetes # docker build -t quickstart-image .
Sending build context to Docker daemon 4.096 kB
Step 1/6 : FROM python:2.7-slim
 ---> 52ad41c7aea4
Step 2/6 : WORKDIR /app
 ---> 89c9072e73dc
Removing intermediate container ed19bad66666
Step 3/6 : ADD . /app
 ---> 639567118ceb
Removing intermediate container edf864f1ed60
Step 4/6 : RUN pip install -r requirements.txt
 ---> Running in a20e7b1459d6
Collecting Flask (from -r requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting itsdangerous>=0.21 (from Flask->-r requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.4 (from Flask->-r requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting Werkzeug>=0.7 (from Flask->-r requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting click>=2.0 (from Flask->-r requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->Flask->-r requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
  Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/fc/a8/66/24d655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
  Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/88/a7/30/e39a54a87bcbe25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, Flask
Successfully installed Flask-0.12.2 Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 itsdangerous-0.24
 ---> 11b3e9be36e2
Removing intermediate container a20e7b1459d6
Step 5/6 : EXPOSE 80
 ---> Running in 06195fd43105
 ---> b9cb682c42f1
Removing intermediate container 06195fd43105
Step 6/6 : CMD python app.py
 ---> Running in 90bbcc9d4ec5
 ---> 745503856a7f
Removing intermediate container 90bbcc9d4ec5
Successfully built 745503856a7f
```

e. Tagging your image

Before you push your Docker image, you need to tag it with its registry name. Tagging your Docker image with a registry configuration the docker push command to push the image to a specific location.

The registry name format is below:

```
[HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

where

- **[HOSTNAME]**is the **gcr.io** hostname
- **[PROJECT-ID]** is your Google Cloud Platform Console project ID
- **[IMAGE]** is your image's name

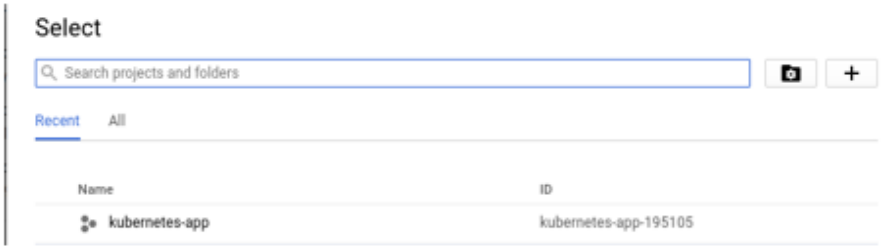
To tag your Docker image for Container Registry, run the following command:

```
docker tag [IMAGE] [HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

For example

```
docker tag quickstart-image gcr.io/my-project/quickstart-image
```

```
Kubernetes # docker tag quickstart-image gcr.io/kubernetes-app-195105/quickstart-image
Kubernetes #
```



f. Pushing your image

Pushing your image

To push your Docker image to Container Registry, run the following command:

```
gcloud docker -- push [HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

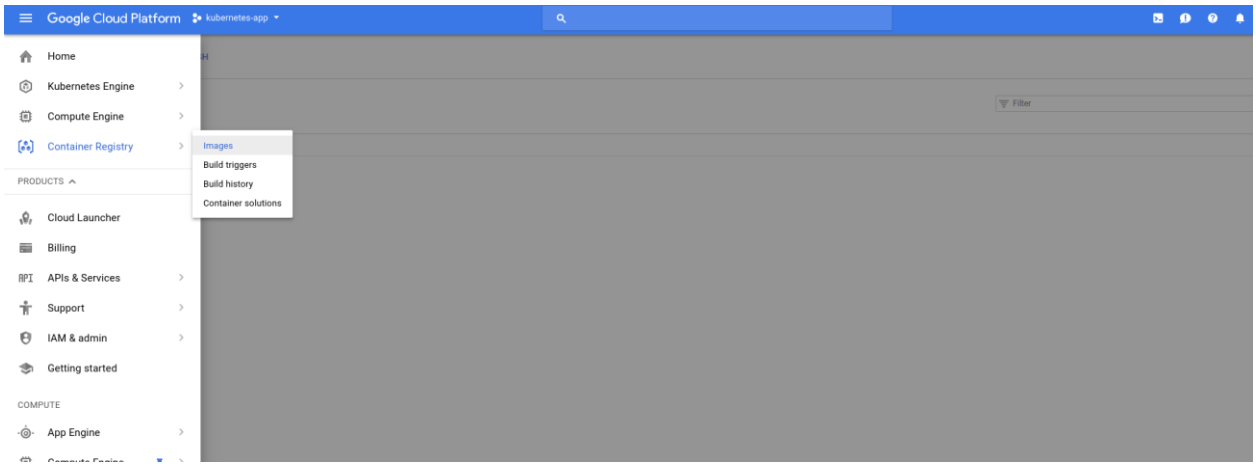
For example:

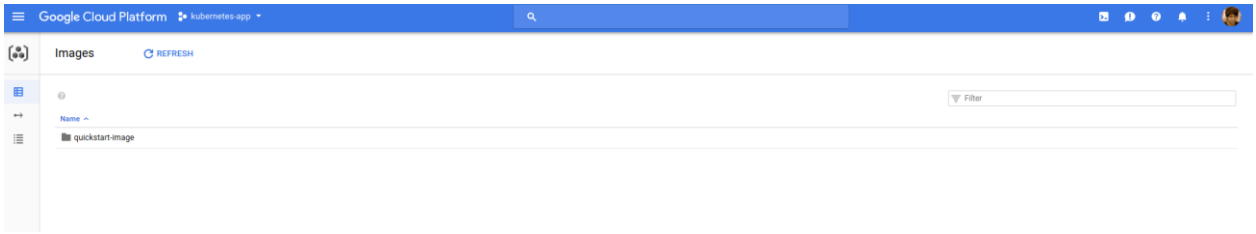
```
gcloud docker -- push gcr.io/my-project/quickstart-image
```

```
frivolouspantheon Kubernetes # docker build -t quickstart-image .
Sending build context to Docker daemon 4.096 kB
Step 1/6 : FROM python:2.7-slim
 ---> 52ad41c7aea4
Step 2/6 : WORKDIR /app
 ---> 89c9072e73dc
Removing intermediate container ed19bad66666
Step 3/6 : ADD . /app
 ---> 639567118ceb
Removing intermediate container edf864f1ed60
Step 4/6 : RUN pip install -r requirements.txt
 ---> Running in a20e7b1459d6
Collecting Flask (from -r requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting itsdangerous>=0.21 (from Flask->-r requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.4 (from Flask->-r requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting Werkzeug>=0.7 (from Flask->-r requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting click>=2.0 (from Flask->-r requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->Flask->-r requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
  Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/fc/a8/66/24d655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
  Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/88/a7/30/e39a54a87bcbe25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, Flask
Successfully installed Flask-0.12.2 Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 itsdangerous-0.24
 ---> 11b3e9be36e2
Removing intermediate container a20e7b1459d6
Step 5/6 : EXPOSE 80
 ---> Running in 06195fd43105
 ---> b9cb682c42f1
Removing intermediate container 06195fd43105
Step 6/6 : CMD python app.py
 ---> Running in 90bbcc9d4ec5
 ---> 745503856a7f
Removing intermediate container 90bbcc9d4ec5
Successfully built 745503856a7f
frivolouspantheon Kubernetes # docker tag quickstart-image gcr.io/kubernetes-app-195105/quickstart-image
frivolouspantheon Kubernetes # docker images
REPOSITORY                                    TAG                 IMAGE ID            CREATED             SIZE
gcr.io/kubernetes-app-195105/quickstart-image   latest              745503856a7f        5 minutes ago       149 MB
quickstart-image                              latest              745503856a7f        5 minutes ago       149 MB
python                                        2.7-slim            52ad41c7aea4        4 days ago          139 MB
```

```
frivolouspantheon Kubernetes # gcloud docker -- push gcr.io/kubernetes-app-195105/quickstart-image
The push refers to a repository [gcr.io/kubernetes-app-195105/quickstart-image]
e12e0d7d6245: Pushed
3e0f650d74d1: Pushed
92e03a00a883: Pushed
03cd3fb86dd2: Pushed
630d02da980e: Pushed
b2f046b20847: Pushed
cf051be4e149: Layer already exists
latest: digest: sha256:e92cf93f95b1842175db06d42095856b267d25e611baf03e05c660d551b9786f size: 1787
frivolouspantheon Kubernetes #
```

Ref: https://cloud.google.com/container-registry/docs/quickstart





10. Deploying an application to the cluster
    Now that we have created a cluster, we can deploy a containerized application to it.

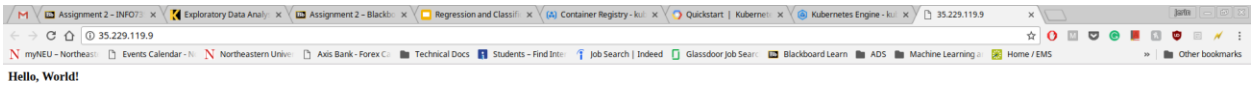
    The kubectl run creates a new deployment name qkimage1. The deployment's pod runs the quickstart images in its container.

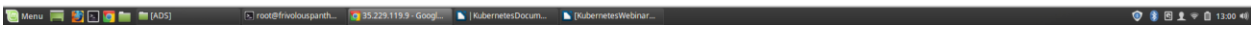
```
frivolouspantheon Kubernetes # kubectl run qkimage1 --image gcr.io/kubernetes-app-195105/quickstart-image --port 80
deployment "qkimage1" created
frivolouspantheon Kubernetes # kubectl expose deployment qkimage1 --type "LoadBalancer"
service "qkimage1" exposed
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   <pending>     80:30962/TCP   6s
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   <pending>     80:30962/TCP   12s
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   <pending>     80:30962/TCP   14s
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   <pending>     80:30962/TCP   15s
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   <pending>     80:30962/TCP   17s
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   <pending>     80:30962/TCP   28s
frivolouspantheon Kubernetes # kubectl get services qkimage1
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
qkimage1   LoadBalancer   10.59.251.37   35.229.119.9    80:30962/TCP   1m
frivolouspantheon Kubernetes #
```

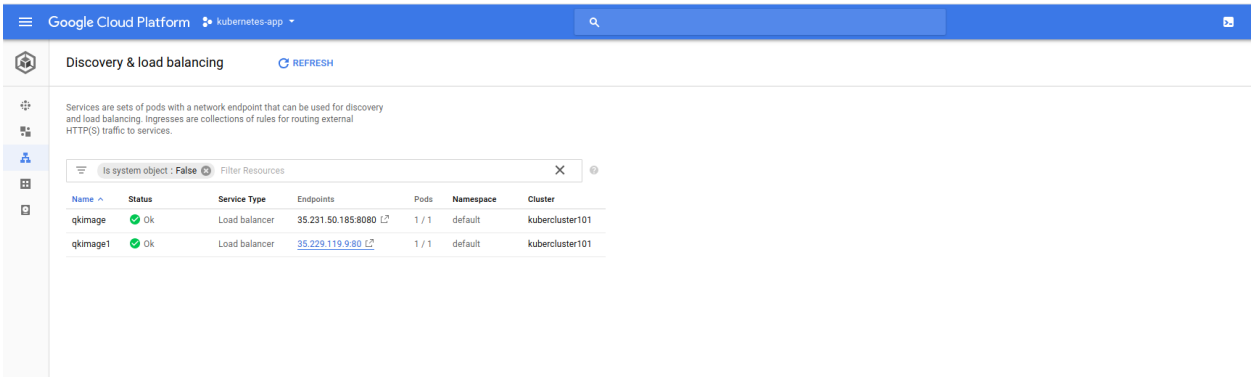
In this command:

- --images specifies a container image to deploy. In this case, the command pulls the example image from Google Container Registry bucket, gcr.io/kubernetes-app-195105/quickstart-image indicated the specific image version to pull. If a version is not specified the latest version is used.
- --port specifies the port that the container exposes.

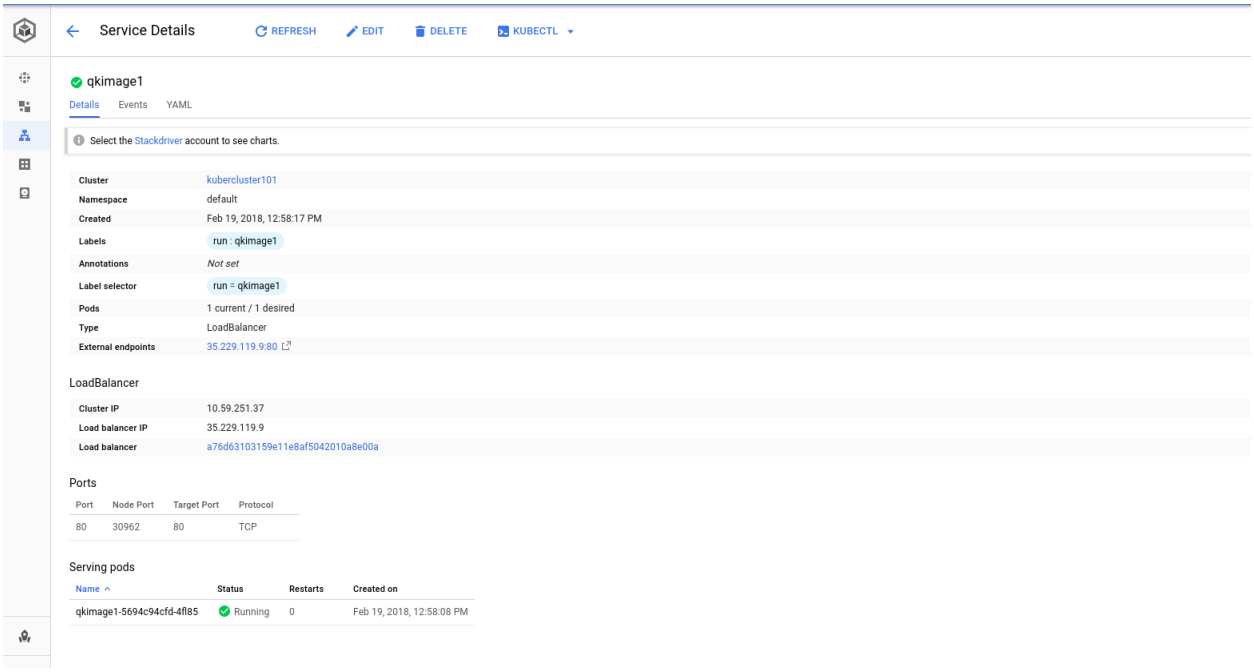
11. Exposing the Deployment
    a. After deploying the application, we need to expose it to the internet so that users can access it. We can expose the application by creating a Service, a Kubernetes resource that exposes your application to external traffic.
    b. Passing in the –type “LoadBalancer” flag creates a Compute Engine load balancer for your container.





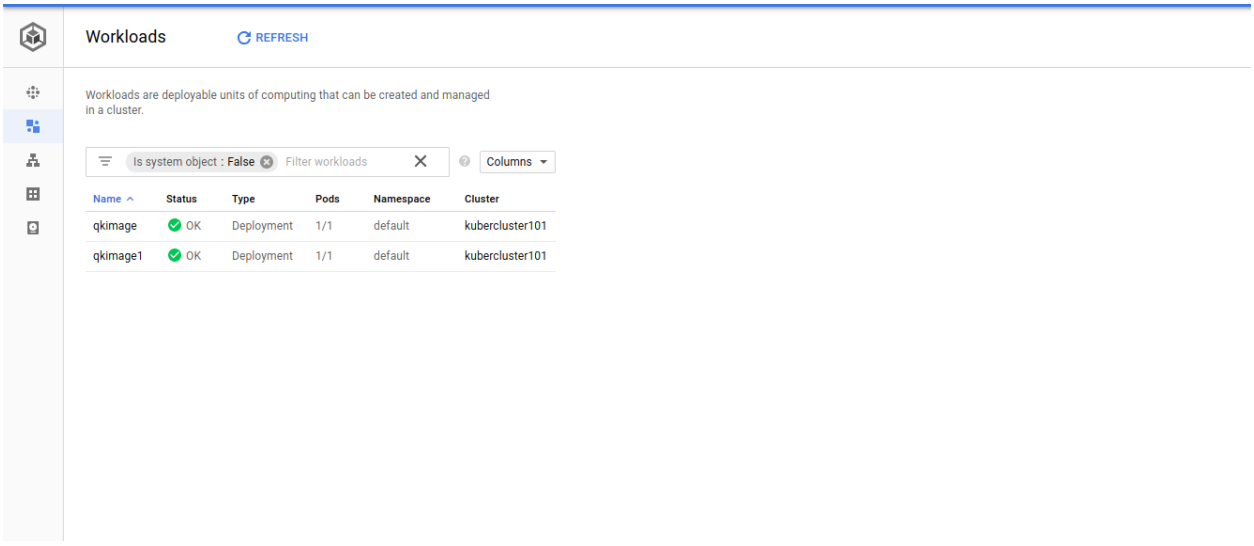


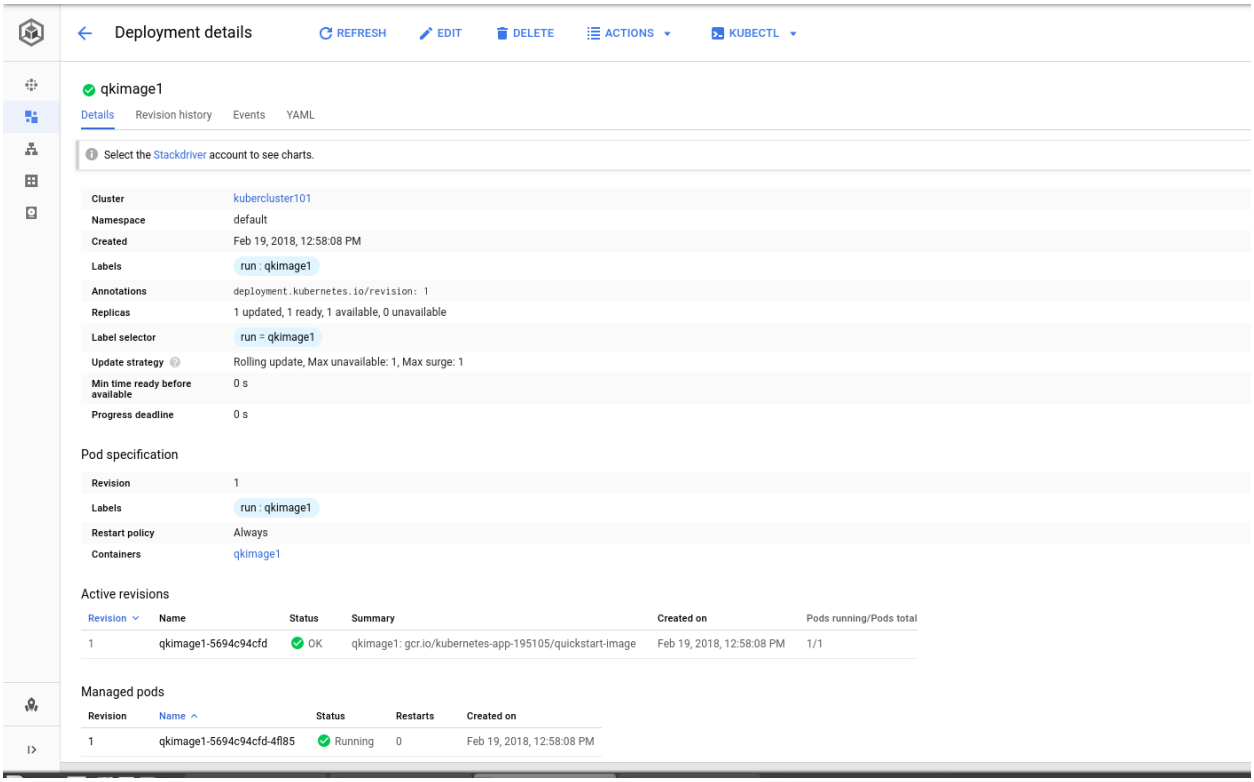
For Load Balancing, we right now use only 1 pod to run the service as per the picture above.

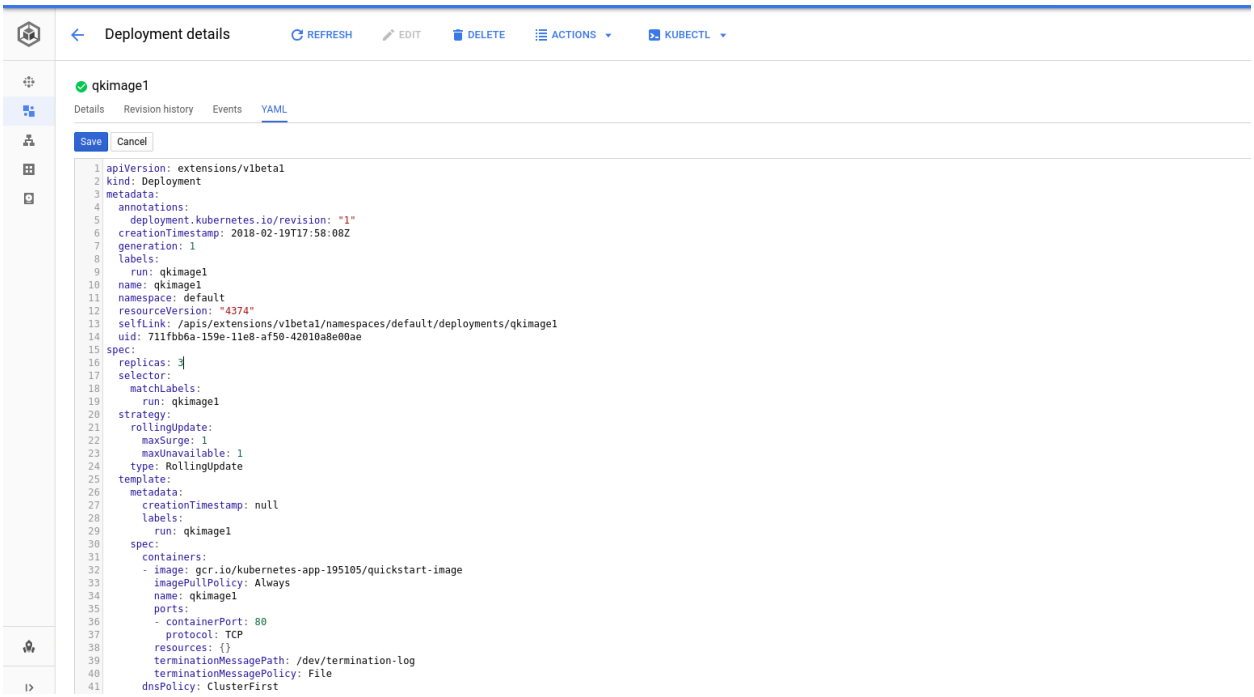


To increase the serving pods

Goto kubernetes Engine > Workloads > Click qkimage > Edit

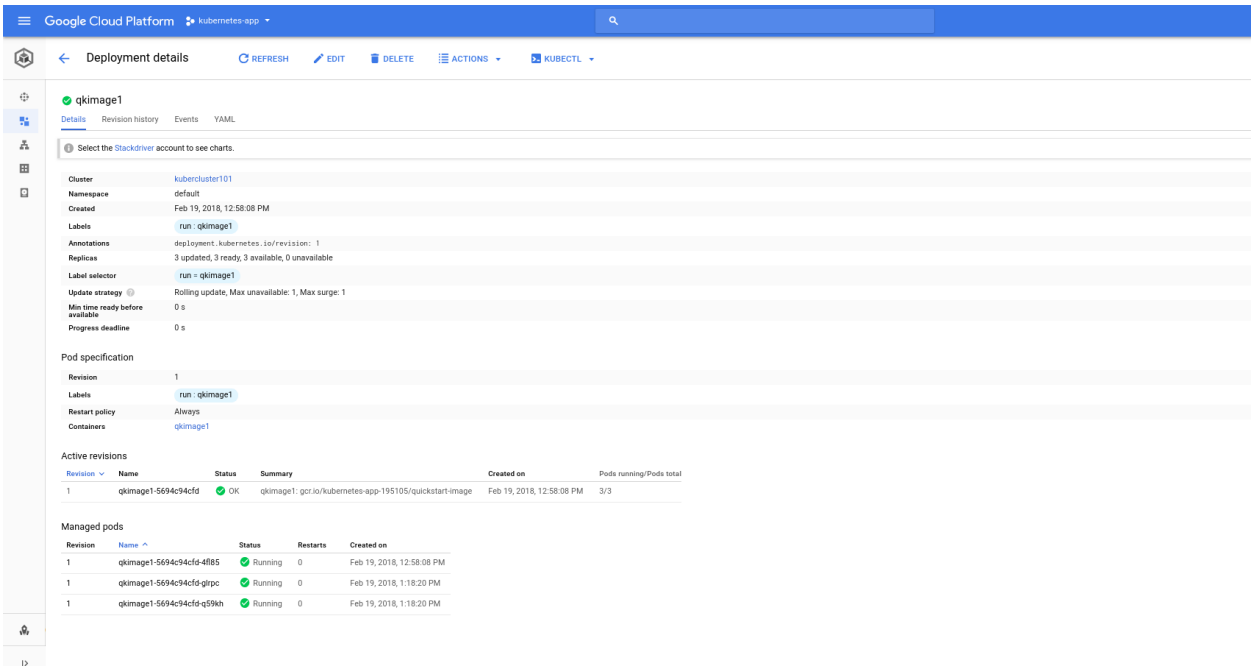






Here I have changed the replicas from 1 to 3.

The immediate effect which has taken place on the kubernetes cluster node.



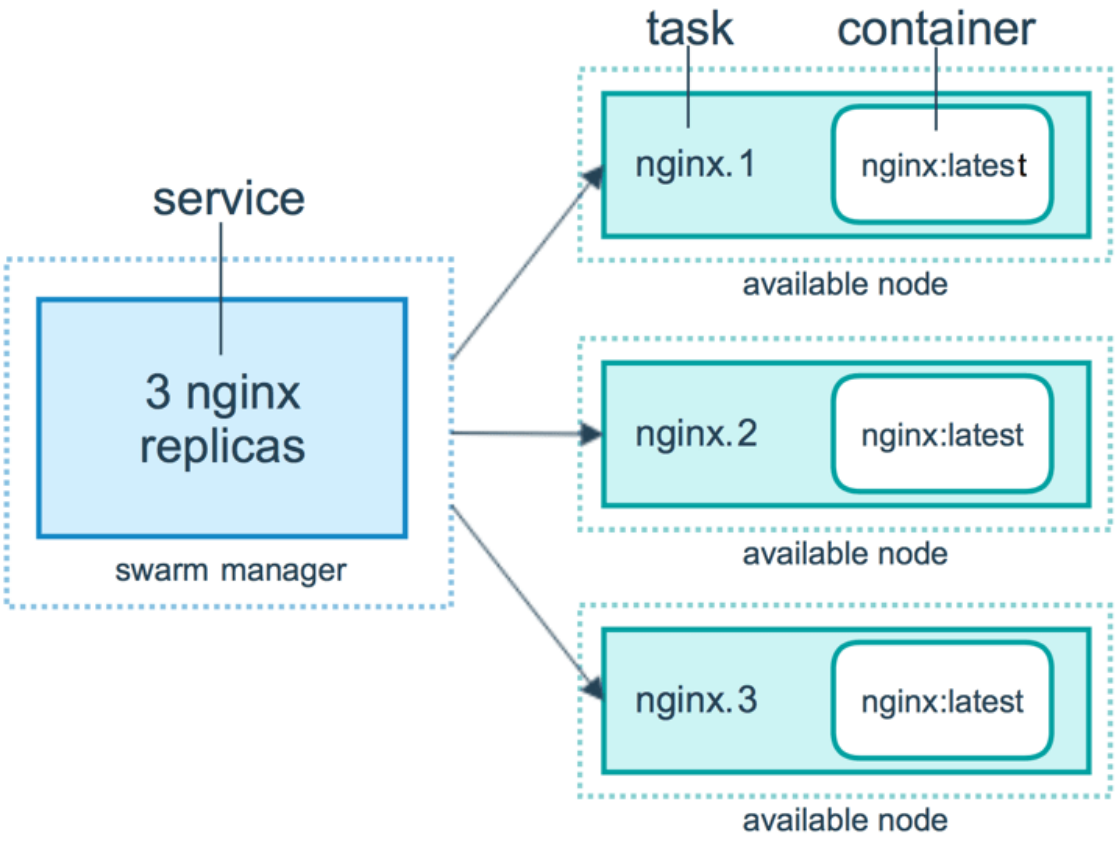
Now the service will be available in all the three pods which will be running the service qkimage1 and image quickstart-image.

Docker Swarm

Docker Swarm is a clustering and scheduling tool for Docker containers. With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.

Swarms are a cluster of nodes which comprise of:

- Manager nodes: Control orchestration, cluster management, and the distribution of tasks.
- Worker nodes: The sole purpose of workers is to run containers and services as assigned by a manager node.
- Services: A service describes how you'd like an individual container to distribute itself across your nodes. To create a service, specify the exact information as in an ordinary 'docker run', plus new parameters (i.e., # of container replicas).
- Tasks: Single containers place work within these "slots" according to the Swarm manager.



Steps to create a swarm :

To create a set of Docker machines that will act as nodes in our Docker Swarm.

For creating Manager Node:

docker-machine create --driver hyperv manager1 //For OS with Hyper-V manager

Or

docker-machine create --driver virtualbox manager1 //For Docker Toolbox

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine create --driver virtualbox manager1
Running pre-create checks...
Creating machine...
(manager1) Copying C:\Users\Akash\.docker\machine\cache\boot2docker.iso to C:\Users\Akash\.docker\machine\machines\manager1\boot2docker.iso...
(manager1) Creating VirtualBox VM...
(manager1) Creating SSH key...
(manager1) Starting the VM...
(manager1) Check network to re-create if needed...
(manager1) Windows might ask for the permission to configure a dhcp server. Sometimes, such confirmation window is minimized in the taskbar.
(manager1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: C:\Program Files\Docker Toolbox\docker-machine.exe env manager1

Akash@DESKTOP-BBTTUHF MINGW64 ~
```

For creating Worker Nodes:

docker-machine create --driver virtualbox worker1

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine create --driver virtualbox worker1
Running pre-create checks...
Creating machine...
(worker1) Copying C:\Users\Akash\.docker\machine\cache\boot2docker.iso to C:\Users\Akash\.docker\machine\machines\worker1\boot2docker.iso...
(worker1) Creating VirtualBox VM...
(worker1) Creating SSH key...
(worker1) Starting the VM...
(worker1) Check network to re-create if needed...
(worker1) Windows might ask for the permission to configure a dhcp server. Sometimes, such confirmation window is minimized in the taskbar.
(worker1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: C:\Program Files\Docker Toolbox\docker-machine.exe env worker1
```

docker-machine create --driver virtualbox worker2

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine create --driver virtualbox worker2
Running pre-create checks...
Creating machine...
(worker2) Copying C:\Users\Akash\.docker\machine\cache\boot2docker.iso to C:\Users\Akash\.docker\machine\machines\worker2\boot2docker.iso...
(worker2) Creating VirtualBox VM...
(worker2) Creating SSH key...
(worker2) Starting the VM...
(worker2) Check network to re-create if needed...
(worker2) Windows might ask for the permission to configure a dhcp server. Sometimes, such confirmation window is minimized in the taskbar.
(worker2) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: C:\Program Files\Docker Toolbox\docker-machine.exe env worker2
```

docker-machine ls //to get list of docker machines

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
default    *        virtualbox   Running   tcp://192.168.99.100:2376           v18.02.0-ce
manager1   -        virtualbox   Running   tcp://192.168.99.101:2376           v18.02.0-ce
worker1    -        virtualbox   Running   tcp://192.168.99.102:2376           v18.02.0-ce
worker2    -        virtualbox   Running   tcp://192.168.99.103:2376           v18.02.0-ce
```

Ip's of docker machines.

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ip manager1
192.168.99.101

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ip worker1
192.168.99.102

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ip worker2
192.168.99.103
```

To SSH into any of the machines.

docker-machine ssh <machine-name>

docker-machine ssh manager1

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ssh manager1
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \_______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.02.0-ce, build HEAD : 99245f4 - Thu Feb  8 17:43:39 UTC 2018
Docker version 18.02.0-ce, build fc4de44
```

To set up the Swarm.

docker swarm init --advertise-addr MANAGER_IP

```
docker@manager1:~$ docker swarm init --advertise-addr 192.168.99.101
Swarm initialized: current node (vod2qiqcocjesh62dohqopxep) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0zz0o252oa1ajaezixm3ggnb8h4noxcwx4yo561f7w2av85dje-99fqm8zp1ykm5tpw74esypxlg 192.168.99.101:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

To find out what docker swarm command to use to join as a node, you will need to use the join-token <role> command.

docker swarm join-token worker

```
docker@manager1:~$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0zz0o252oa1ajaezixm3ggnb8h4noxcwx4yo561f7w2av85dje-99fqm8zp1ykm5tpw74esypxlg 192.168.99.101:2377
```

docker swarm join-token manager

```
docker@manager1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0zz0o252oa1ajaezixm3ggnb8h4noxcwx4yo561f7w2av85dje-aysng5gsdsyj1hpw1y8uzfwcr 192.168.99.101:2377
```

To get the list of nodes in the swarm.

docker node ls

```
docker@manager1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
vod2qiqcocjesh62dohqopxep *   manager1            Ready               Active              Leader
```

SSH into the worker1 machine.

Open a new docker terminal.

docker-machine ssh worker1

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ssh worker1
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \_____ o           __/
            \    \         __/
             \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.02.0-ce, build HEAD : 99245f4 - Thu Feb  8 17:43:39 UTC 2018
Docker version 18.02.0-ce, build fc4de44
docker@worker1:~$ docker swarm join-token worker
Error response from daemon: This node is not a swarm manager. Use "docker swarm init" or "docker swarm join" to connect this node to swarm and try again.
```

To join worker1 to the swarm.

Use the join-token generated for worker

```
docker@worker1:~$ docker swarm join --token SWMTKN-1-0zz0o252oa1ajaezixm3ggnb8h4noxcwx4yo561f7w2av85dje-99fqm8zp1ykm5tpw74esypxlg 192.168.99.101:2377
This node joined a swarm as a worker.
docker@worker1:~$
```

docker node ls //on the manager1

```
docker@manager1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
vod2qiqcocjesh62dohqopxep *   manager1            Ready               Active              Leader
ufh55mibj423fnikw3fzc3nse     worker1             Ready               Active
```

SSH into the worker2 machine.

Open a new docker terminal.

docker-machine ssh worker2

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ssh worker2
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/

 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.02.0-ce, build HEAD : 99245f4 - Thu Feb  8 17:43:39 UTC 2018
Docker version 18.02.0-ce, build fc4de44
```

To join worker2 to the swarm

Use the join-token generated for worker

```
docker@worker2:~$ docker swarm join --token SWMTKN-1-0zz0o252oa1ajaezixm3ggnb8h4noxcwx4yo561f7w2av85dje-99fqm8zp1ykm5tpw74esypxlg 192.168.99.101:2377
This node joined a swarm as a worker.
docker@worker2:~$
```

docker node ls //on the manager1

```
docker@manager1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
vod2qiqcocjesh62dohqopxep *   manager1            Ready               Active              Leader
ufh55mibj423fnikw3fzc3nse     worker1             Ready               Active
9uzkaso7yb9ttlq9cuj0bae6u     worker2             Ready               Active
docker@manager1:~$
```

To get swarm information.

docker info //on the manager1 and zoom into the swarm section

```
docker@manager1:~$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.02.0-ce
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: active
 NodeID: vod2qiqcocjesh62dohqopxep
 Is Manager: true
 ClusterID: r0j7h9o9uieil5qn024vm7yy0
 Managers: 1
 Nodes: 3
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 192.168.99.101
 Manager Addresses:
  192.168.99.101:2377
```

To create a Service.

```
docker@manager1:~$ docker service create --replicas 2 -p 8080:80 --name web nginx
uma8aodbr2bbb4wy8uf2zjqi8
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
docker@manager1:~$
```

Check status of service.

```
docker@manager1:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
uma8aodbr2bb        web                 replicated          2/2                 nginx:latest        *:8080->80/tcp
docker@manager1:~$
```

```
docker@manager1:~$ docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR               PORTS
9tepjllqjzft        web.1               nginx:latest        worker1             Running             Running 3 minutes ago
rl0pvefo1b9t        web.2               nginx:latest        manager1            Running             Running 3 minutes ago
docker@manager1:~$
```

```
docker@manager1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
a63f21833ba7        nginx:latest        "nginx -g 'daemon of…"   5 minutes ago       Up 5 minutes        80/tcp              web.2.rl0pvefo1b9tygq86jotqepre
docker@manager1:~$
```

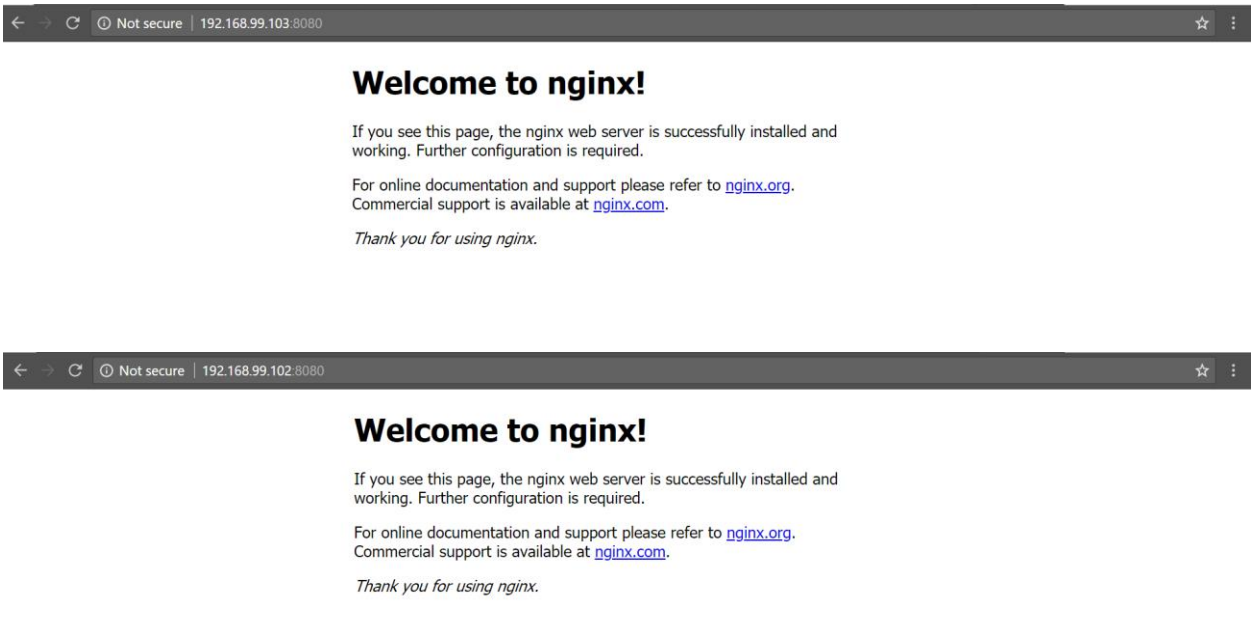
list of docker machines and their ip’s //on new docker terminal

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
default    *        virtualbox   Running   tcp://192.168.99.100:2376           v18.02.0-ce
manager1   -        virtualbox   Running   tcp://192.168.99.101:2376           v18.02.0-ce
worker1    -        virtualbox   Running   tcp://192.168.99.102:2376           v18.02.0-ce
worker2    -        virtualbox   Running   tcp://192.168.99.103:2376           v18.02.0-ce
```

To access the Service.

http://<machine ip>:8080 in the browser.

You should be able to get the standard NGINX Home page.



Scaling up.

To Scale up to 5 replicas from 2 replicas

```
docker@manager1:~$ docker service scale web=5
web scaled to 5
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
docker@manager1:~$
```

To check status of service

```
docker@manager1:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
uma8aodbr2bb        web                 replicated          5/5                 nginx:latest        *:8080->80/tcp
docker@manager1:~$ docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE               ERROR               PORTS
9tepjllqjzft        web.1               nginx:latest        worker1             Running             Running 20 minutes ago
rl0pvefo1b9t        web.2               nginx:latest        manager1            Running             Running 20 minutes ago
njcihgned39i        web.3               nginx:latest        worker1             Running             Running about a minute ago
oo4modmbezbb        web.4               nginx:latest        worker2             Running             Running about a minute ago
7ag1uaqj8p48        web.5               nginx:latest        worker2             Running             Running about a minute ago
docker@manager1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
a63f21833ba7        nginx:latest        "nginx -g 'daemon of…"   20 minutes ago      Up 20 minutes       80/tcp              web.2.rl0pvefo1b9tygq86jotqepre
docker@manager1:~$
```

Checking tasks running on manager1

```
docker@manager1:~$ docker node ps self
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR               PORTS
rl0pvefo1b9t        web.2               nginx:latest        manager1            Running             Running 23 minutes ago
docker@manager1:~$
```

Checking tasks running worker1 and worker2

```
docker@manager1:~$ docker node ps worker1
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR               PORTS
9tepjllqjzft        web.1               nginx:latest        worker1             Running             Running 24 minutes ago
njcihgned39i        web.3               nginx:latest        worker1             Running             Running 5 minutes ago
docker@manager1:~$ docker node ps worker2
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR               PORTS
oo4modmbezbb        web.4               nginx:latest        worker2             Running             Running 5 minutes ago
7ag1uaqj8p48        web.5               nginx:latest        worker2             Running             Running 5 minutes ago
docker@manager1:~$
```

Inspecting a node.

docker node inspect [--pretty] Nodename

```
docker@manager1:~$ docker node inspect --pretty worker1
ID:                     ufh55mibj423fnikw3fzc3nse
Hostname:               worker1
Joined at:              2018-02-14 06:05:07.318961913 +0000 utc
Status:
 State:                 Ready
 Availability:          Active
 Address:               192.168.99.102
Platform:
 Operating System:      linux
 Architecture:          x86_64
Resources:
 CPUs:                  1
 Memory:                995.9MiB
Plugins:
 Log:           awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk, syslog
 Network:               bridge, host, macvlan, null, overlay
 Volume:                local
Engine Version:         18.02.0-ce
Engine Labels:
 - provider=virtualbox
TLS Info:
 TrustRoot:
-----BEGIN CERTIFICATE-----
MIIBajCCARCgAwIBAgIUda+SP/yFFCxEicgVQnEP1oKCfSowCgYIKoZIzj0EAwIw
EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTgwMjE0MDU0MzAwWhcNMzgwMjA5MDU0
MzAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABJ5kqJhb8xsa0D+dzX1DxjYM8q5xHR6J2topEIpLHf9qZwmffBuigI9nfICz
MNeMGoTxy/U+vo/3SjsCWMlxSAijQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBQKkKr9P4tgKcggNBk5l7LJJe+0QDAKBggqhkjO
PQQDAgNIADBFAiEA4FQaxsuDjiuCQS8l70eD1ZJExtRSJuDtSI4hW7sZZnQCIFhT
YdEewMrWHdIc7OLRBBMSne5JTjFqbucRnrDbHs11
-----END CERTIFICATE-----

 Issuer Subject:        MBMxETAPBgNVBAMTCHN3YXJtLWNh
 Issuer Public Key:     MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEnmSomFvzGxrQP53NfUPGNgzyrnEdHona2ikQiksd/2pnCZ98G6KAj2d8gLMw14wahPHL9T6+j/dKOwJYyXFICA==
```

Draining a node.

docker node update --availability drain Nodename

```
docker@manager1:~$ docker node update --availability drain worker1
worker1
docker@manager1:~$ docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
orasm2f1bwjz        web.1               nginx:latest        manager1            Running             Running 17 seconds ago
9tepjllqjzft         \_ web.1           nginx:latest        worker1             Shutdown            Shutdown 16 seconds ago
rl0pvefo1b9t        web.2               nginx:latest        manager1            Running             Running 35 minutes ago
x6x76g9h7kos        web.3               nginx:latest        manager1            Running             Running 17 seconds ago
njcihgned39i         \_ web.3           nginx:latest        worker1             Shutdown            Shutdown 16 seconds ago
oo4modmbezbb        web.4               nginx:latest        worker2             Running             Running 16 minutes ago
7ag1uaqj8p48        web.5               nginx:latest        worker2             Running             Running 16 minutes ago
```

Checking replicas running on worker1,worker2 and manager1

```
docker@manager1:~$ docker node ps worker1
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
9tepjllqjzft        web.1               nginx:latest        worker1             Shutdown            Shutdown 47 seconds ago
njcihgned39i        web.3               nginx:latest        worker1             Shutdown            Shutdown 47 seconds ago
docker@manager1:~$ docker node ps worker2
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
oo4modmbezbb        web.4               nginx:latest        worker2             Running             Running 16 minutes ago
7ag1uaqj8p48        web.5               nginx:latest        worker2             Running             Running 16 minutes ago
docker@manager1:~$ docker node ps manager1
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE                ERROR               PORTS
orasm2f1bwjz        web.1               nginx:latest        manager1            Running             Running about a minute ago
rl0pvefo1b9t        web.2               nginx:latest        manager1            Running             Running 35 minutes ago
x6x76g9h7kos        web.3               nginx:latest        manager1            Running             Running about a minute ago
docker@manager1:~$
```

Inspecting the Service.

```
docker@manager1:~$ docker service inspect --pretty web

ID:             uma8aodbr2bbb4wy8uf2zjqi8
Name:           web
Service Mode:   Replicated
 Replicas:      5
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         nginx:latest@sha256:285b49d42c703fdf257d1e2422765c4ba9d3e37768d6ea83d7fe2043dad6e63d
Resources:
Endpoint Mode:  vip
Ports:
 PublishedPort = 8080
  Protocol = tcp
  TargetPort = 80
  PublishMode = ingress
```

Removing the Service.

docker service rm ServiceName

```
docker@manager1:~$ docker service rm web
web
docker@manager1:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
docker@manager1:~$ docker service inspect --pretty web
Error: no such service: web
docker@manager1:~$ docker node ps worker1
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE       ERROR               PORTS
docker@manager1:~$ docker node ps worker2
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE       ERROR               PORTS
docker@manager1:~$ docker node ps manager1
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE       ERROR               PORTS
docker@manager1:~$ docker service ps web
no such service: web
docker@manager1:~$
```

Stopping docker machine.

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
default    *        virtualbox   Running   tcp://192.168.99.100:2376           v18.02.0-ce
manager1   -        virtualbox   Running   tcp://192.168.99.101:2376           v18.02.0-ce
worker1    -        virtualbox   Running   tcp://192.168.99.102:2376           v18.02.0-ce
worker2    -        virtualbox   Running   tcp://192.168.99.103:2376           v18.02.0-ce

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine stop worker1
Stopping "worker1"...
Machine "worker1" was stopped.

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine stop worker2
Stopping "worker2"...
Machine "worker2" was stopped.

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine stop manager1
Stopping "manager1"...
Machine "manager1" was stopped.

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-manager ls
bash: docker-manager: command not found

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
default    *        virtualbox   Running   tcp://192.168.99.100:2376           v18.02.0-ce
manager1   -        virtualbox   Stopped                                       Unknown
worker1    -        virtualbox   Stopped                                       Unknown
worker2    -        virtualbox   Stopped                                       Unknown
```

Deleting docker-machine.

```
Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine rm worker1 worker2 manager1
About to remove worker1, worker2, manager1
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed worker1
Successfully removed worker2
Successfully removed manager1

Akash@DESKTOP-BBTTUHF MINGW64 ~
$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
default   *        virtualbox   Running   tcp://192.168.99.100:2376           v18.02.0-ce

Akash@DESKTOP-BBTTUHF MINGW64 ~
$
```

Kubernetes Vs Docker Swarm.

https://platform9.com/blog/compare-kubernetes-vs-docker-swarm/

Microservices

- Software systems become more complex as their scale
  - scope
  - Volume
  - user interactions
- Sometimes system that grow in size beyond the boundaries initially define pose a particular problems when it comes to changing them

What is Microservices?

- Microservice architecture are used to achieve faster delivery and greater safety as the scale of their systems increase
- Microservices are goal-oriented
- Microservices are focused on replaceability
- This idea that driving toward replacement of components rather than maintaining existing components get to the very heart of what makes the microservices approach special

Microservices

- More specifically, the real value of microservices is realized when we focus on two key aspects - **speed and safety**
- **The Speed of Change** - The desire for speed is a desire for immediate change and ultimately a desire for adaptability
- **At Scale** - Systems that can work at scale don't break when under pressure; instead they incorporate built-in mechanisms to increase capacity in a safe way
- **In Harmony** - Organizations that succeed with microservice architecture are able to maintain their system stability while increasing their change velocity. In other words, they created a harmony of speed and safety that works for their own context

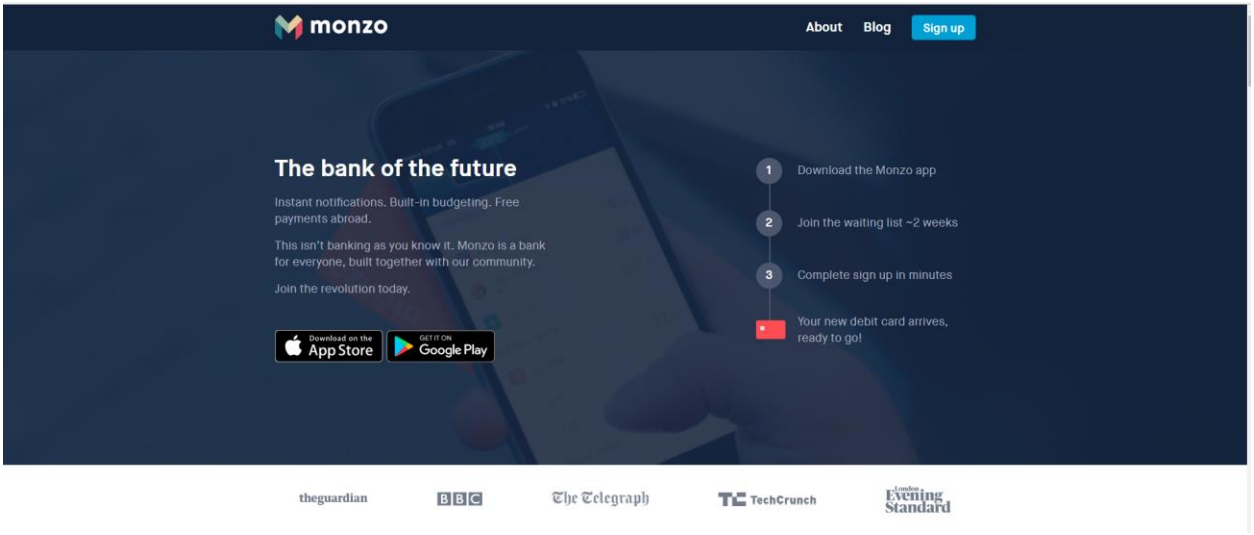
How it impact Business ?

- **Agility** allows organizations to deliver new products, functions, and features more quickly and pivot more easily if needed
- **Comprehensibility of the software system** simplifies development planning, increases accuracy, and allows new resources to come up to speed more quickly
- **Independent deployment** of components gets new features into production more quickly and provides more flexible options for piloting and prototyping
- Organizational alignment of services to teams reduces ramp-up time and encourages teams to **build more complex products** and features iteratively
- **Polyglotism** permits the use of the right tools for the right task, thus accelerating technology introduction and increasing solution options

Case Studies

MONZO: Build a Modern Bank Backend.

Ref:https://monzo.com/blog/2016/09/19/building-a-modern-bank-backend/



Objective: At Monzo, we're building a banking system from scratch. Our systems must be available 24x7 no matter what happens, scalable to hundreds of millions of customers around the world, and very extensible. This first post in a series about our platform explains how we're building systems to meet these demands using modern, open-source technology.

Strategy: Building the backend as a collection of distributed **microservices**. For an early-stage startup, this architecture is quite unusual; most companies start with a centralised application using well-known frameworks and relational databases.

But with an ambition to build the best current account in the world and scale it to hundreds of millions of customers, we knew that we had to be the bank with the best technology.

Implementation: When we launched the Monzo **beta**, our backend had grown to nearly 100 services (now about 150), and our engineering team had grown considerably too. Few areas of focus: Cluster management, polyglot services, RPC transport, Asynchronous messages.

After a year or so of using Mesos and Marathon, we decided to switch to **Kubernetes**, which we run on a fleet of **CoreOS** machines. It is very tightly integrated with Docker, and is the product of Google's **long experience** running containers in production at global scale. Deploying Kubernetes in a **highly available** configuration on AWS is not for the faint of heart and requires you to get familiar with its internals, but we are very pleased with the results. We regularly kill hosts in production and Kubernetes quickly reschedules applications to accommodate.

There are increasingly large number of such startups and large comporation wanting to implement microservice strategy in developing their product through technology. This advancement can improve their efficiency and transactions can be monitored easily.

Conclusion: Today Fintech is rising at an alarming pace with huge investment from major companies competing for the best technology for driving their business needs and goals. Kubernetes and Docker Swarm are the tools and techniques on how systems operate in a micro service manner.