**Kubernetes Cluster App Deployment**

**Prerequisites:**

1. Existing node.js application

Refer to this application for rest of the tutorial:

<https://github.com/handaanmol/product-service>

1. Refer to DockerHub Image already built - **anmolhanda/product-master-service**

(Note: Try creating a Ubuntu 16.04 VM as it will go well with the tutorials)

**Steps:**

1. **VM Creation**

Create a linux virtual machine or if you already have, connect to it via putty.

1. **Cloning the code in VM**

Install GIT inside VM and configure it and clone the following code there using the following command:

git clone [git@github.com:handaanmol/product-service.git](mailto:git@github.com:handaanmol/product-service.git)

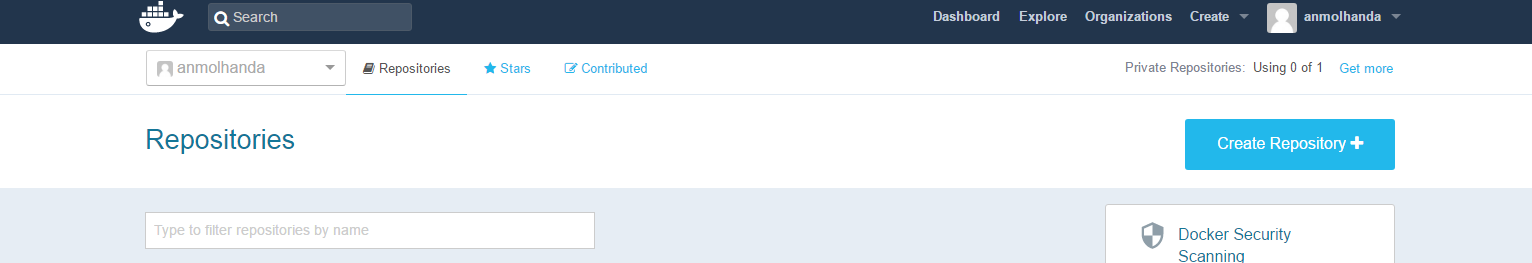
or

git clone <https://github.com/handaanmol/product-service.git>

*(Refer to this link for git installation -* [*https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-16-04*](https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-16-04) *)*

1. **Create DockerHub Account**

Go to <https://hub.docker.com>, create your account and login.



1. **Install Docker in VM**

For installing docker, first check the distribution and release of the linux by following command : **lsb\_release –a**

After checking, find the steps to install docker for that distribution and release.

*(The steps to install and run docker on Ubuntu 16.04 are given here:* [*https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04*](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04)*)*

1. Skip steps 6 to 11 if you already have a dockerhub image available.
2. **Change current working Directory to Node.js Code Directory**

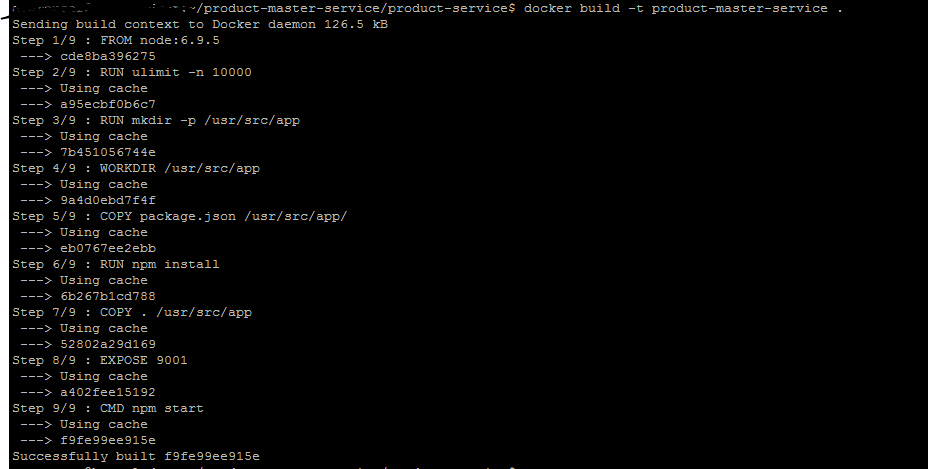
Check the current directory should have the following files as shown below:



1. **Build Docker Image of Code**

Run command:

**docker build –t product-master-service .**



It will build the image in VM

1. **View Docker Image**

Run command:

**docker images**



1. **Connect to Your DockerHub Account from VM**

Run commands:

**export DOCKER\_ID\_USER="<your dockerhub username >"**

**docker login**

(Enter your dockerhub credentials for authentication)

1. **Tagging and Pushing Docker Image**

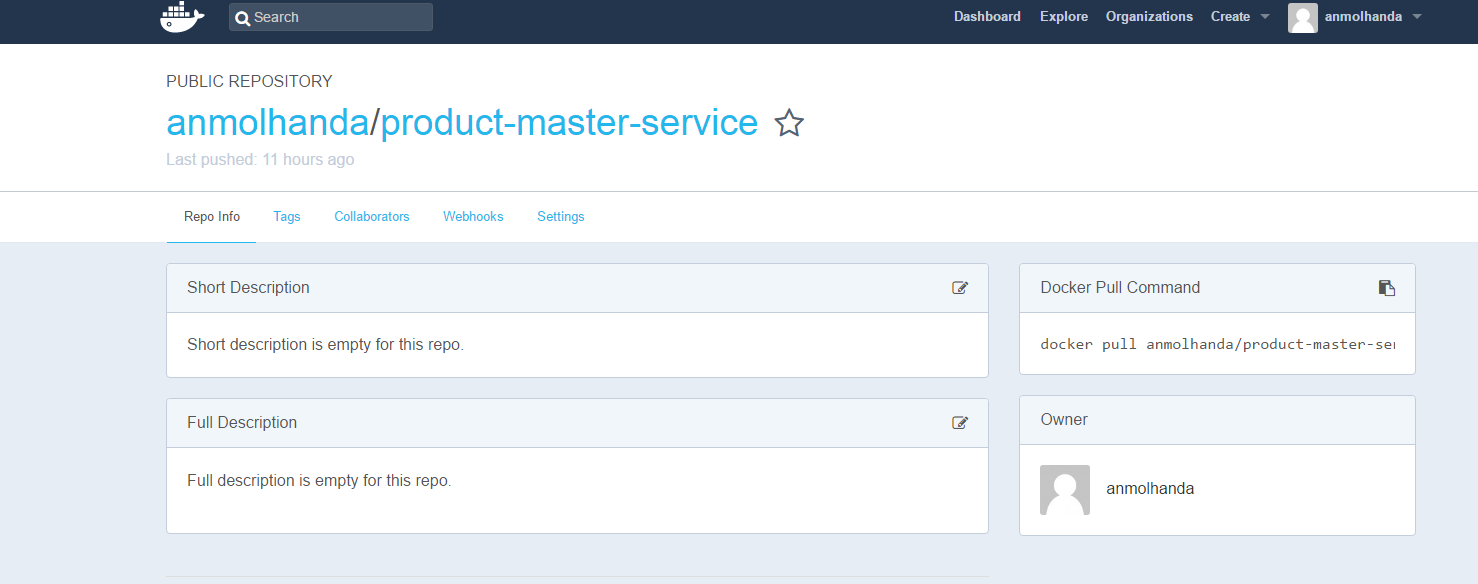
Run command:

**docker tag product-master-service $DOCKER\_ID\_USER/ product-master-service**

After this push the image to dockerhub Registry by following command

**docker push $DOCKER\_ID\_USER/ product-master-service**

This command will push the image to your dockerhub account from where it can be accessed by anyone as shown in the below screenshot



1. **Check Docker Container**

Check whether the image is running as docker container or not by following command in the VM:

**docker run -d -p 9090:9001 --name product-master-services anmolhanda/product-master- service**  
(9090 [exposed port]: 9001[internal port])

If something fails, check Dockerfile.

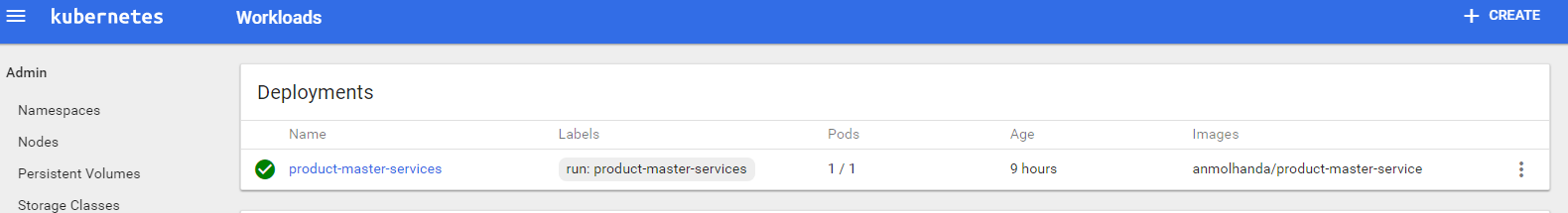
Delete this container from the VM after checking. Run this command to delete the container:

**docker rm product-master-service**

1. **Deploying DockerHub Image on Kubernetes**

From Azure CLI or local CMD, run the following command:

**kubectl run product-master-services --image anmolhanda/product-master-service --port=9001**Successful Deployment can be viewed on Kubernetes Dashboard running locally as shown below:



Alternatively, write the following command on CLI or CMD to see whether deployment has been done successfully or not:

**kubectl get deployments**



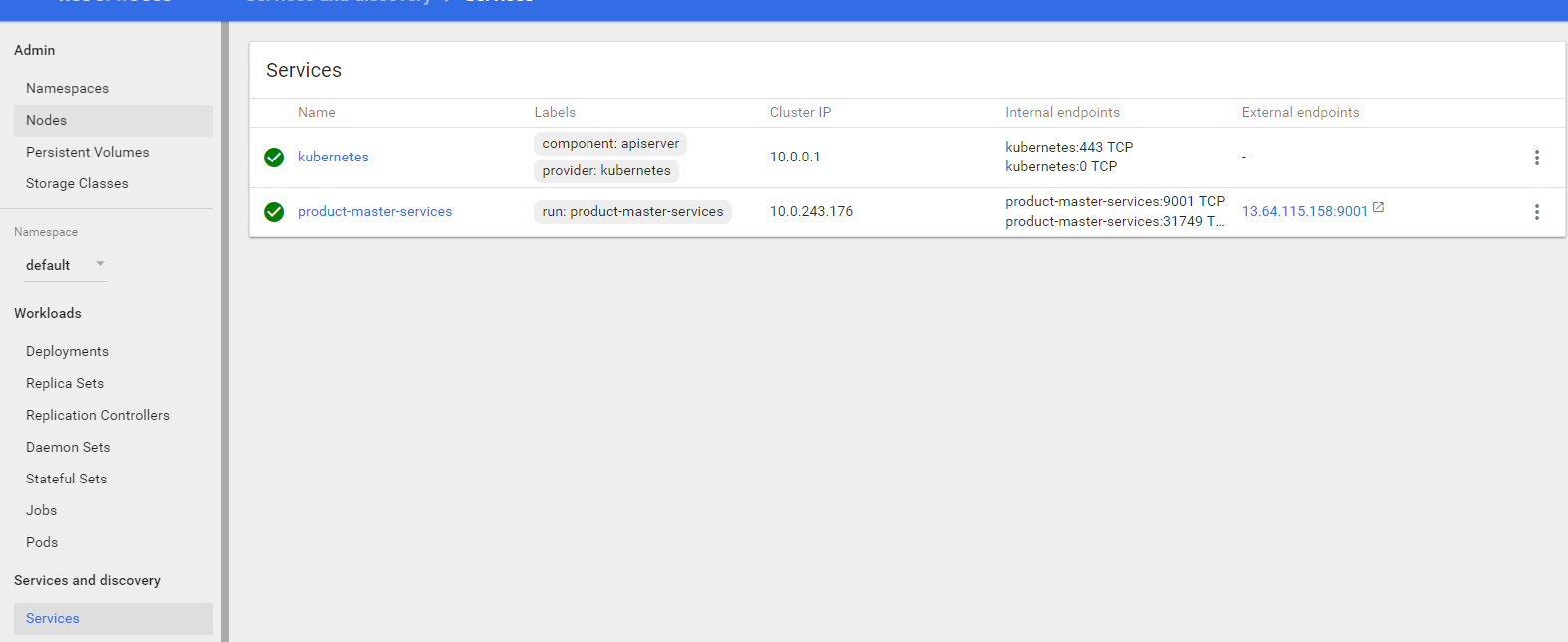
1. **Exposing Deployment as Service**

Run the service using the following command:

**kubectl expose deployment product-master-services --type=LoadBalancer**

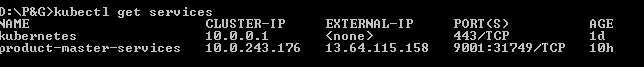
*(This command causes Kubernetes to create an Azure load balancer rule with a public IP address)*

Access the services on the dashboard as shown below and note the external endpoint for future reference.



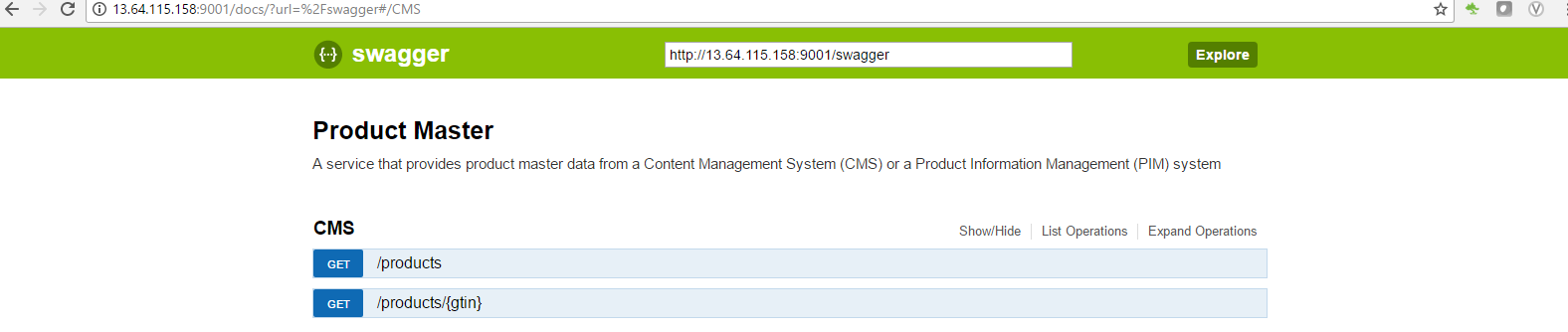
Use the following command using CLI/CMD to view the services

**kubectl get services**



1. **Access the Service on Kubernetes**

Try to access the External IP/ EndPoint from last step via below link:

<http://13.64.115.158:9001/docs> - Swagger Link

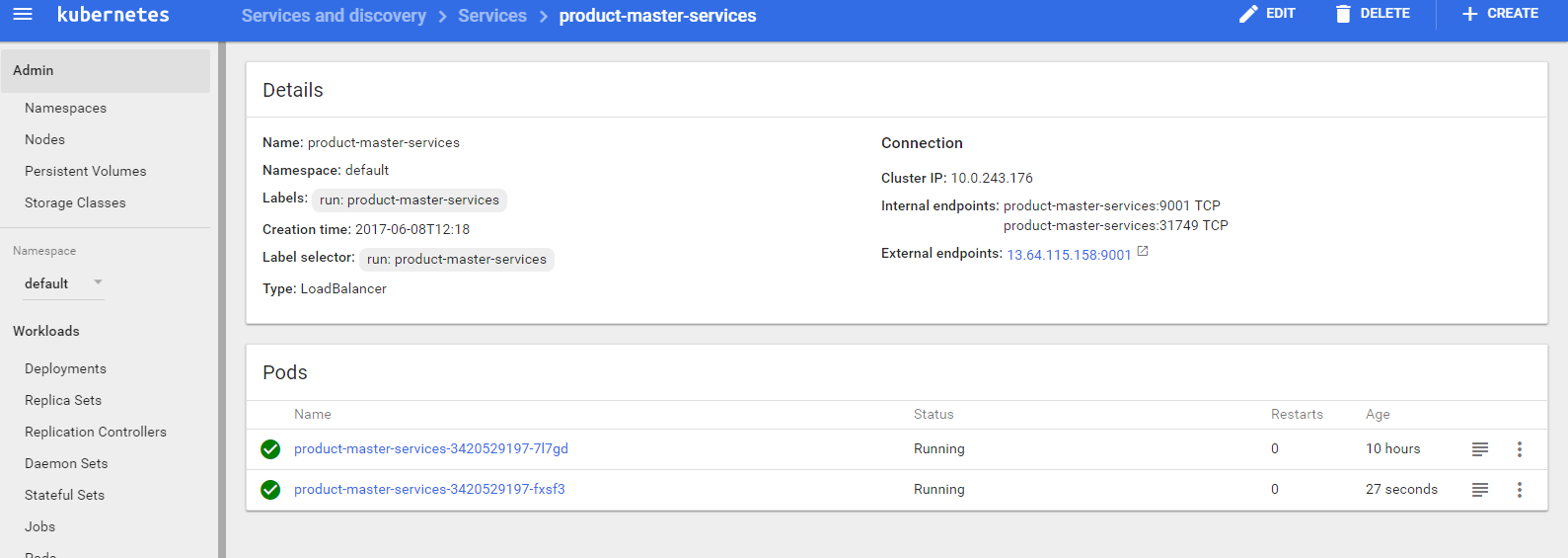
Above screenshot means the app has been successfully deployed to Kubernetes Cluster

1. **Scale Service**

Use the following command:

**kubectl scale deployment product-master-services --replicas=2**

This command would bring up two replication sets or 2 pods for same service in order to promote higher availability and zero downtime as shown below



Thanks