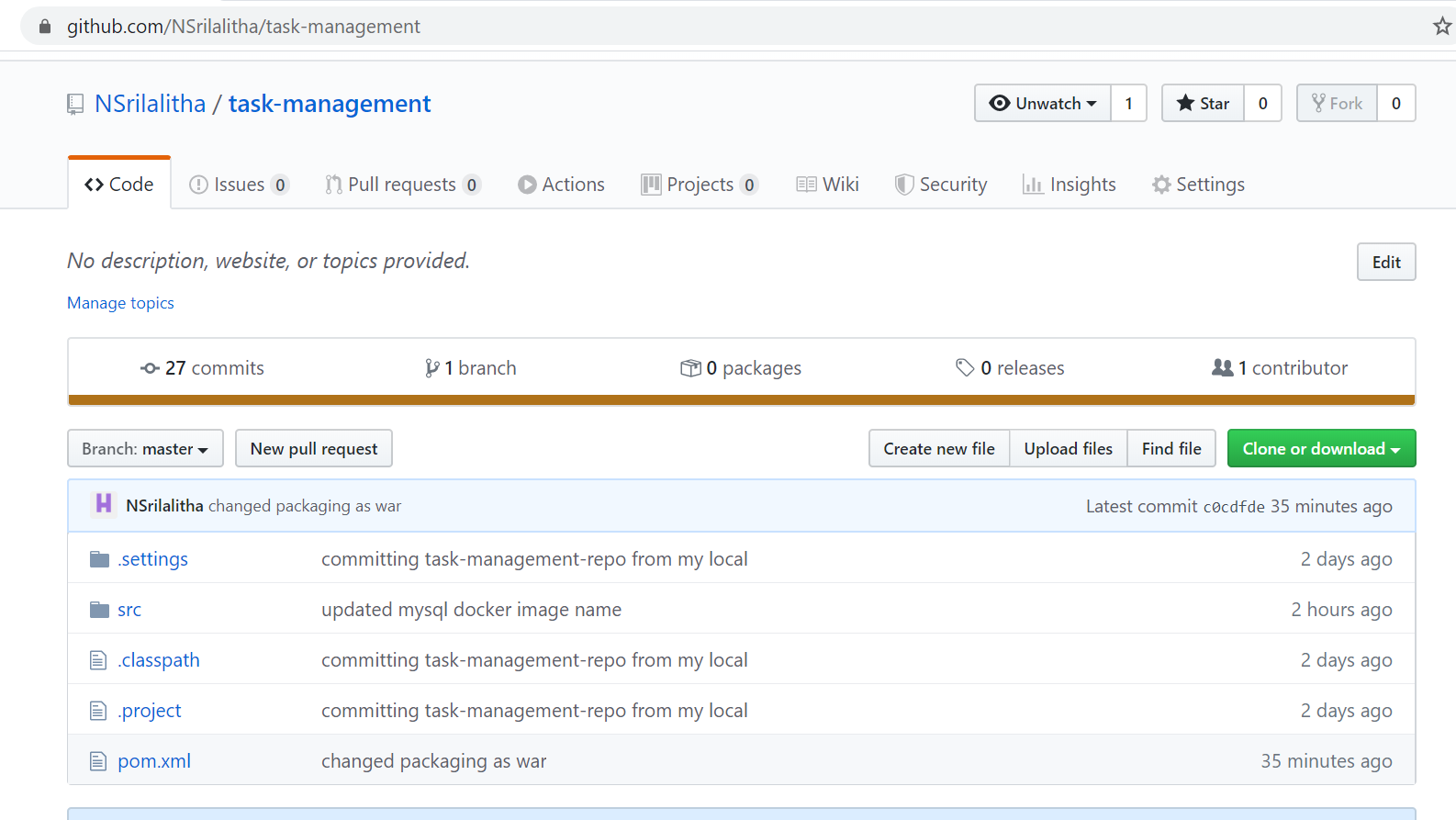
**Kubernetes - Case Study**

* Write a micro services based Java / Node application or download a sample from Github (use can use spring boot based app).

Created spring boot application with MySQL database for managing tasks. The application allows us to add task details. These task details will be persisted into MySQL database using spring boot data jpa and hibernate. Pushed the application to my github account. The application is available in below link, which can be cloned in linux cloud lab in further steps.

<https://github.com/NSrilalitha/task-management>

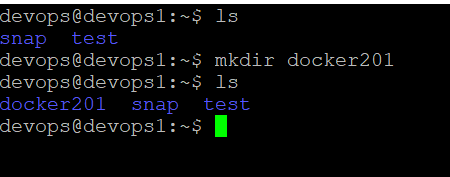


In order to manage containers with Kubernetes first dockerize this application. This application has following dependencies namely java 8, maven and git dependencies. We install these dependencies and make application to run in any environment using Dockerfile.

* Write a Dockerfile for creating an image of the application & Containerize the application

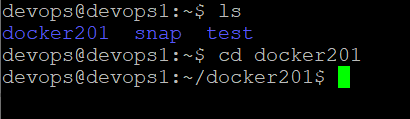
1. Now create a folder called docker201 to work on docker.

**mkdir docker201**



1. Move to docker201 folder

**cd docker201**

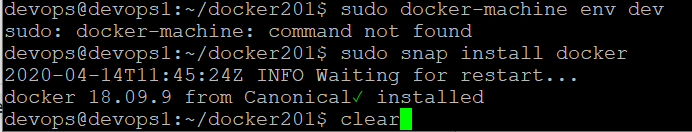


1. Install docker machine with below commands

**sudo apt update**

**sudo apt install snapd**

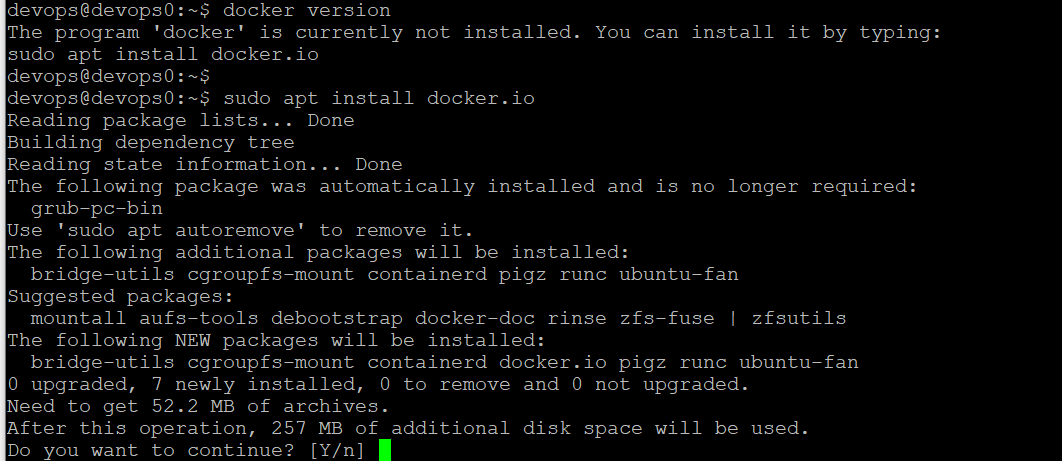
**sudo snap install docker**



(Or)

Install docker using below command

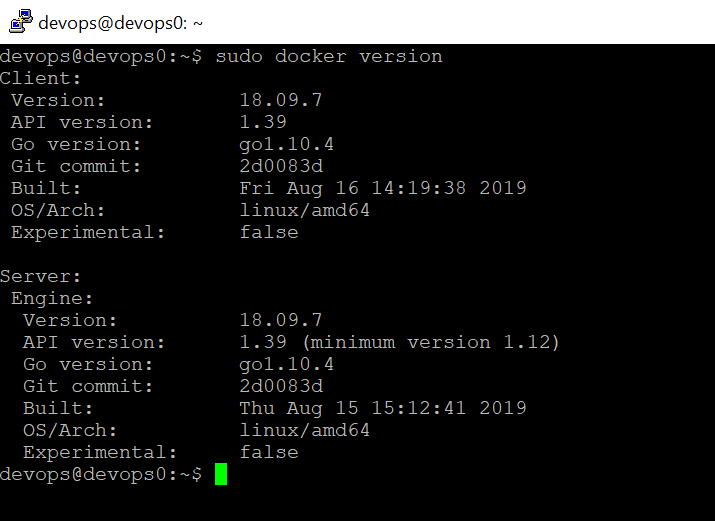
**sudo apt install docker.io**



Now docker is successfully installed. Now verify docker version

1. Verify docker version using below command

**sudo docker version**

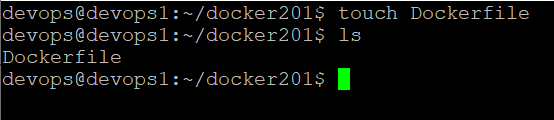


1. Now create Dockerfile.

Dockerfile: A Dockerfile is a text document that contains all the commands which assemble to create a Docker Image. Now we need to create a Dockerfile to containerize this application. Docker can build images by reading instruction from a Dockerfile.

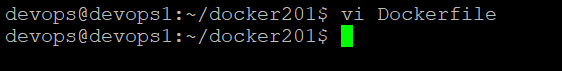
Use below command to create Dockerfile

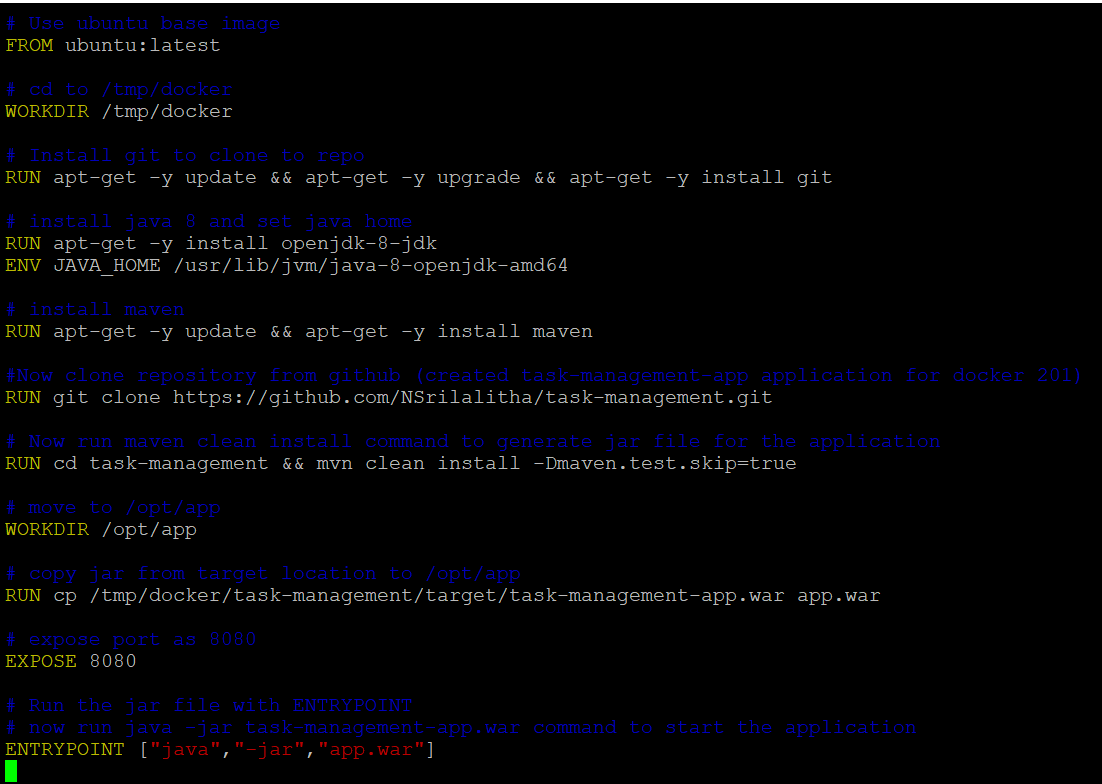
**touch Dockerfile**



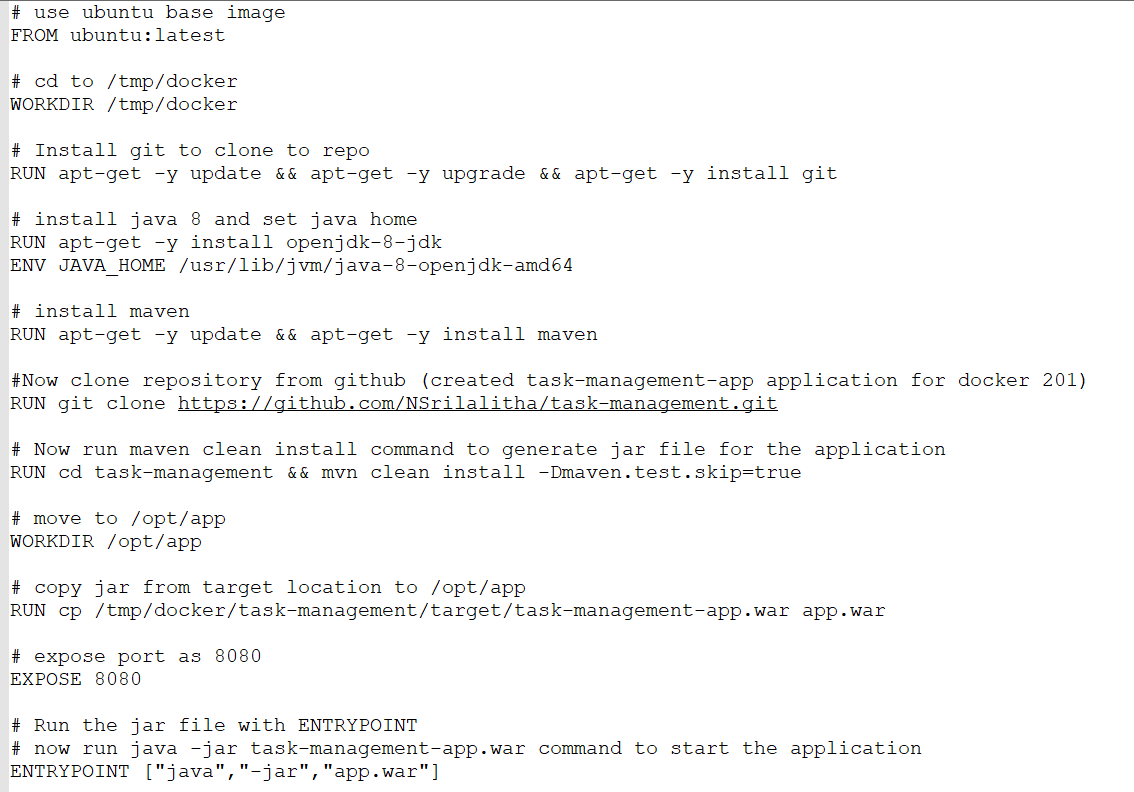
1. Now edit Dockerfile. Use below command to edit Dockerfile

**vi Dockerfile**





Dockerfile should contain the dependencies required to build the docker image. For the task management application, we need jdk8, maven, git dependencies. Git is to clone the task-management repository from GitHub. Below screenshot show Dockerfile contents.



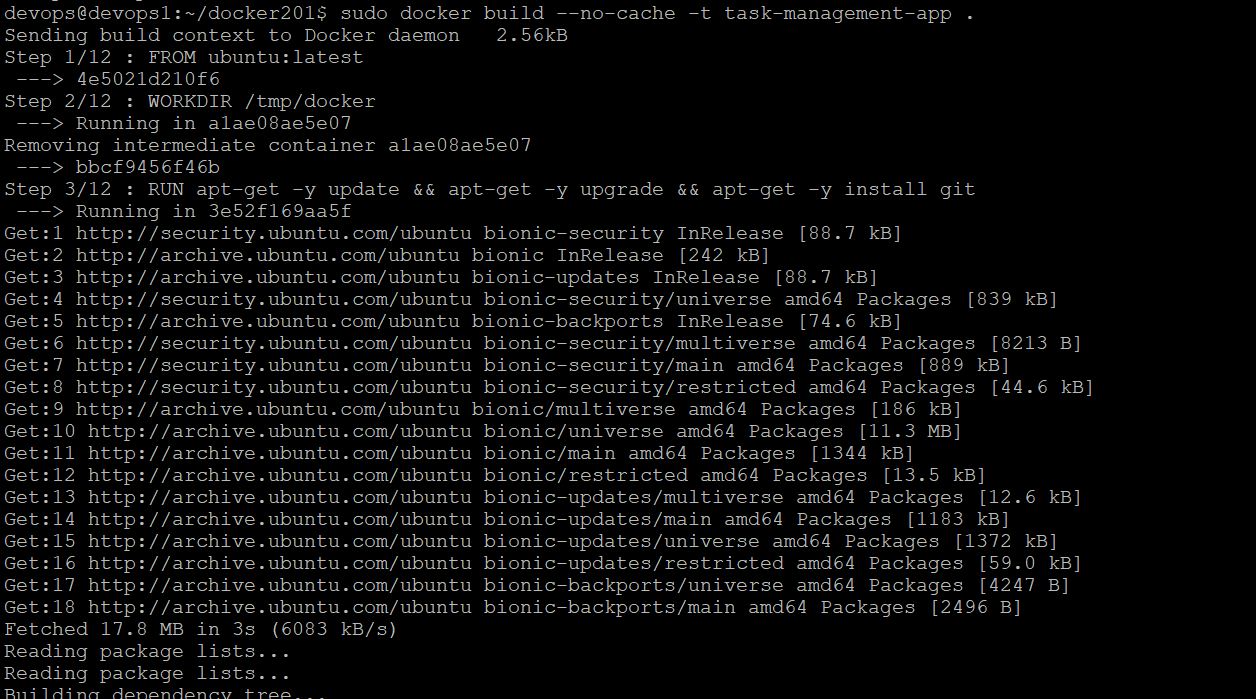
Let’s get into the docker file and understand each and every piece of it.

* **FROM**: This command pulls a public image from Docker Hub and get that added to the build. Here, to build docker image for task-management application, using Ubuntu instance with latest configuration as base image since we are dockerizing the application in linux environment.
* **WORKDIR**: This command sets the working directory for running all the commands mentioned in the Dockerfile.
* **RUN**: This command executes any command on the top of the image and commits the result. Here, installing jdk 8, maven and git dependencies through RUN command. This will install these dependencies and make these dependencies available to the docker to make application to run. RUN command can be used to execute any commands like installation, copy etc. Here we are installing git to clone the application from GitHub.
* **ENV**: This command sets the home path of the different application. Here we are setting java path using this command.
* **EXPOSE**: This command ensures the container that the application listens to the specific port at runtime. Here, exposed port 8080 for this application.
* **ENTRYPOINT**: ENTRYPOINT is the program that is executed to start the application. Here task management application is java based spring boot application. So to run any java application we use “java –jar app.jar/war”. Since this web application the packaging is war type. So given “java –jar app.war” in ENTRYPOINT.

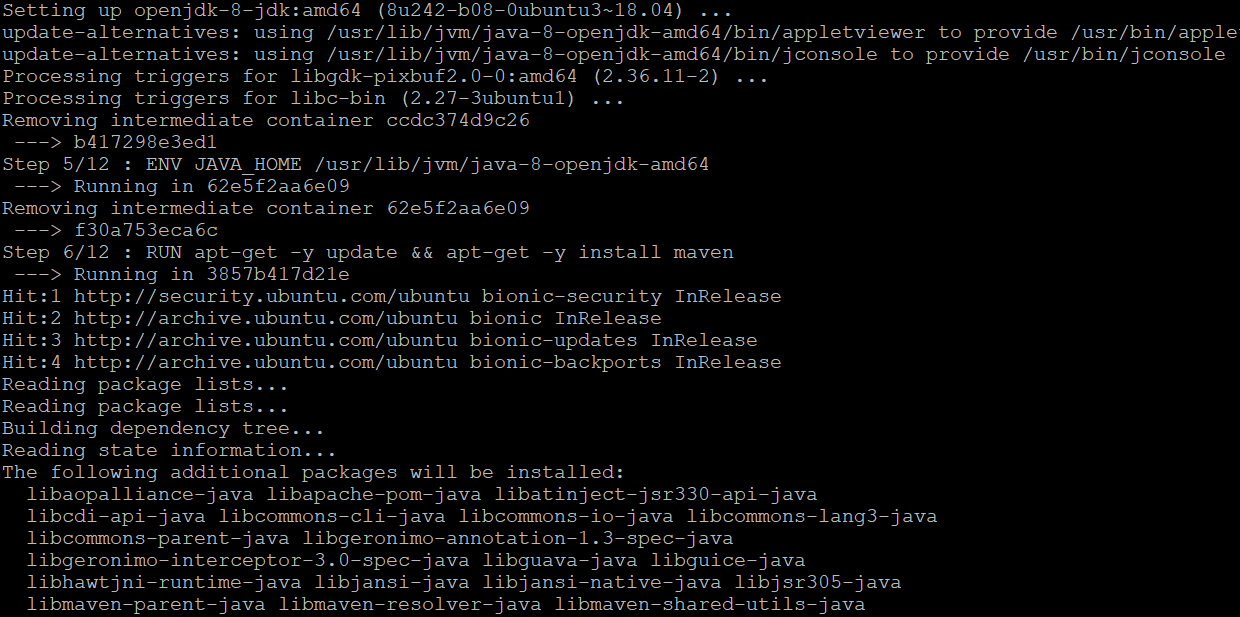
1. Now build docker image using below command

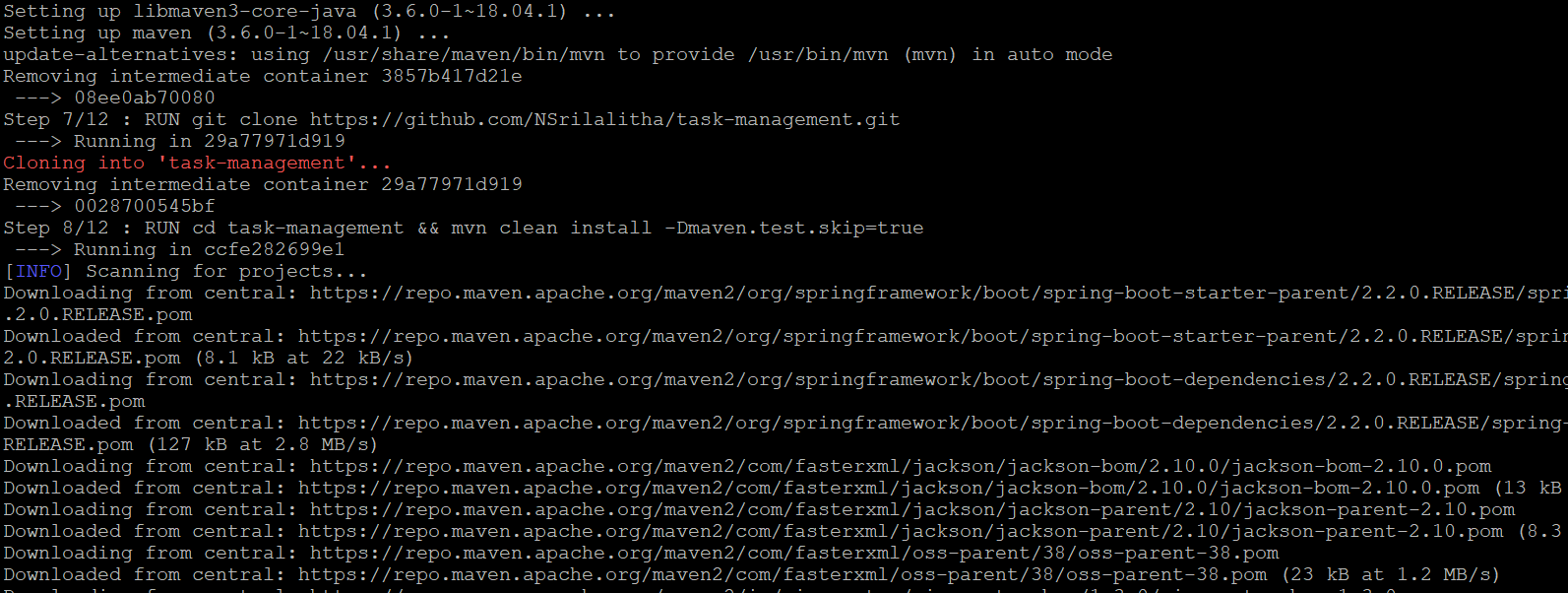
**sudo docker build –t task-management-app .**

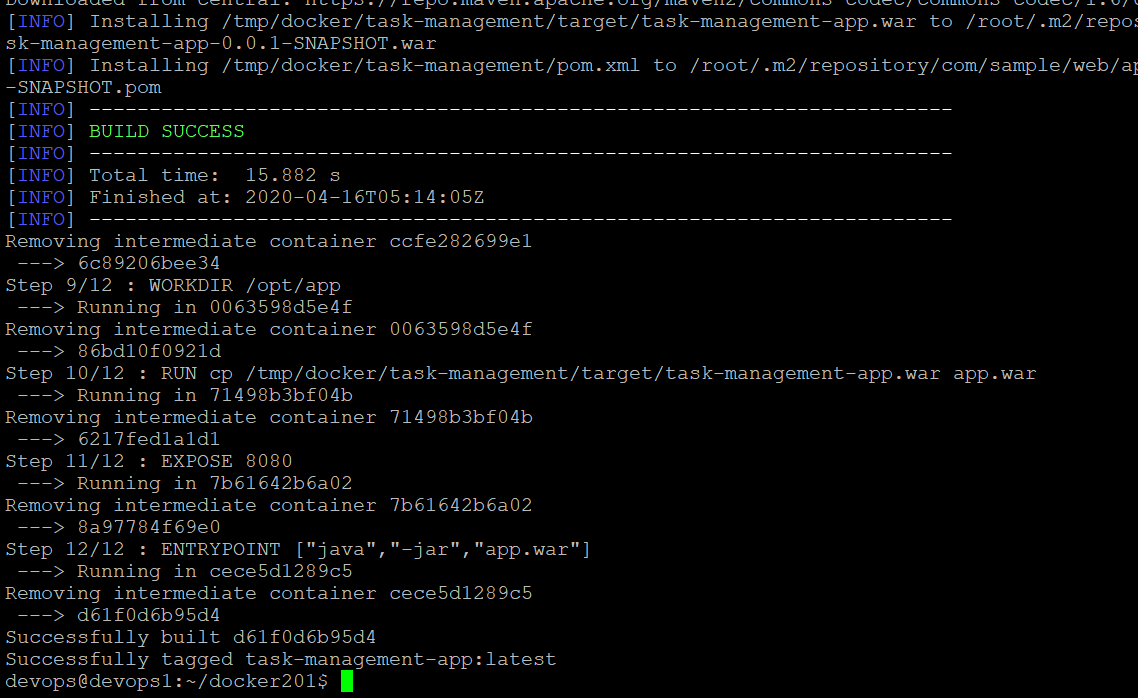
Here “task-management-app” is docker image name given for this application.





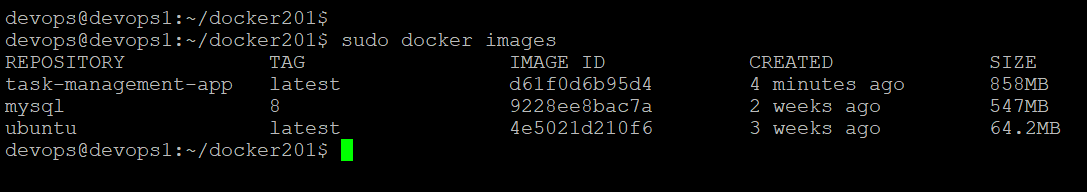






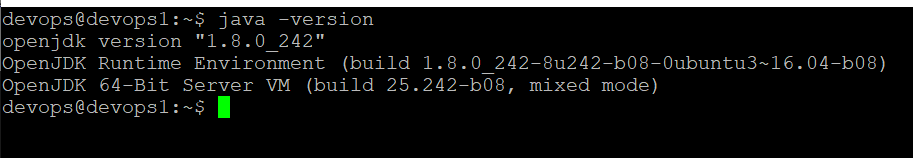
1. Now docker image is successfully build. To see all docker images created use below command.

**sudo docker images**



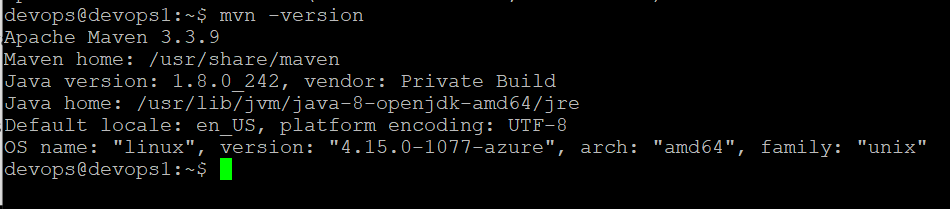
1. All the dependencies (java, maven, git) required to run this application is downloaded through Dockerfile. Now verify all these dependencies installed or not.
   1. Verify java version using below command

**java –version**



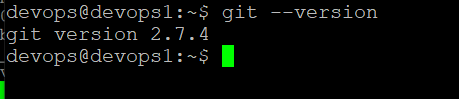
* 1. Verify maven version using below command

**mvn –version**



* 1. Verify git version

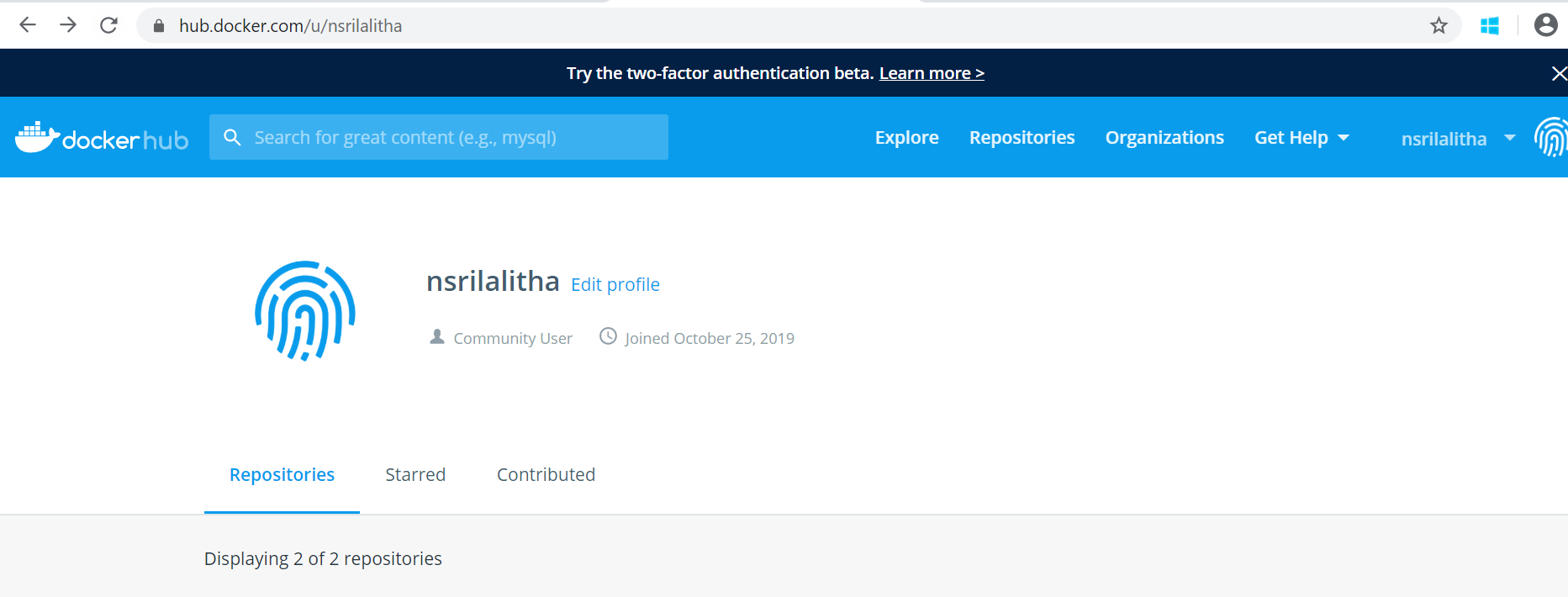
**git –version**



1. In order to run the application I will use kubernetes. To run the application we need two pods, in each pod one container will be there. one for spring boot application and one for MySQL. Here in this application am using MySQL for data persistence. So I will be defining all the services (application, MySQL) in docker compose configuration file. To create deployments and services for this application and MySQL I will make use of this docker-compose file in later sections.

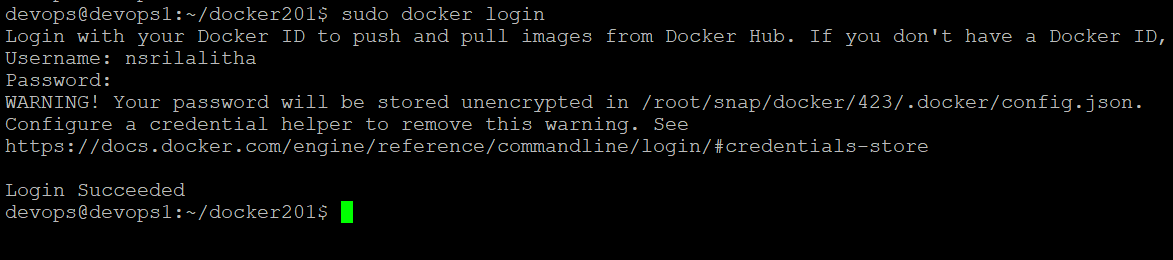
* Push the image to the Docker Hub

1. Inorder to push the docker image to docker hub we need account in docker hub. So created account in Docker hub <https://hub.docker.com>

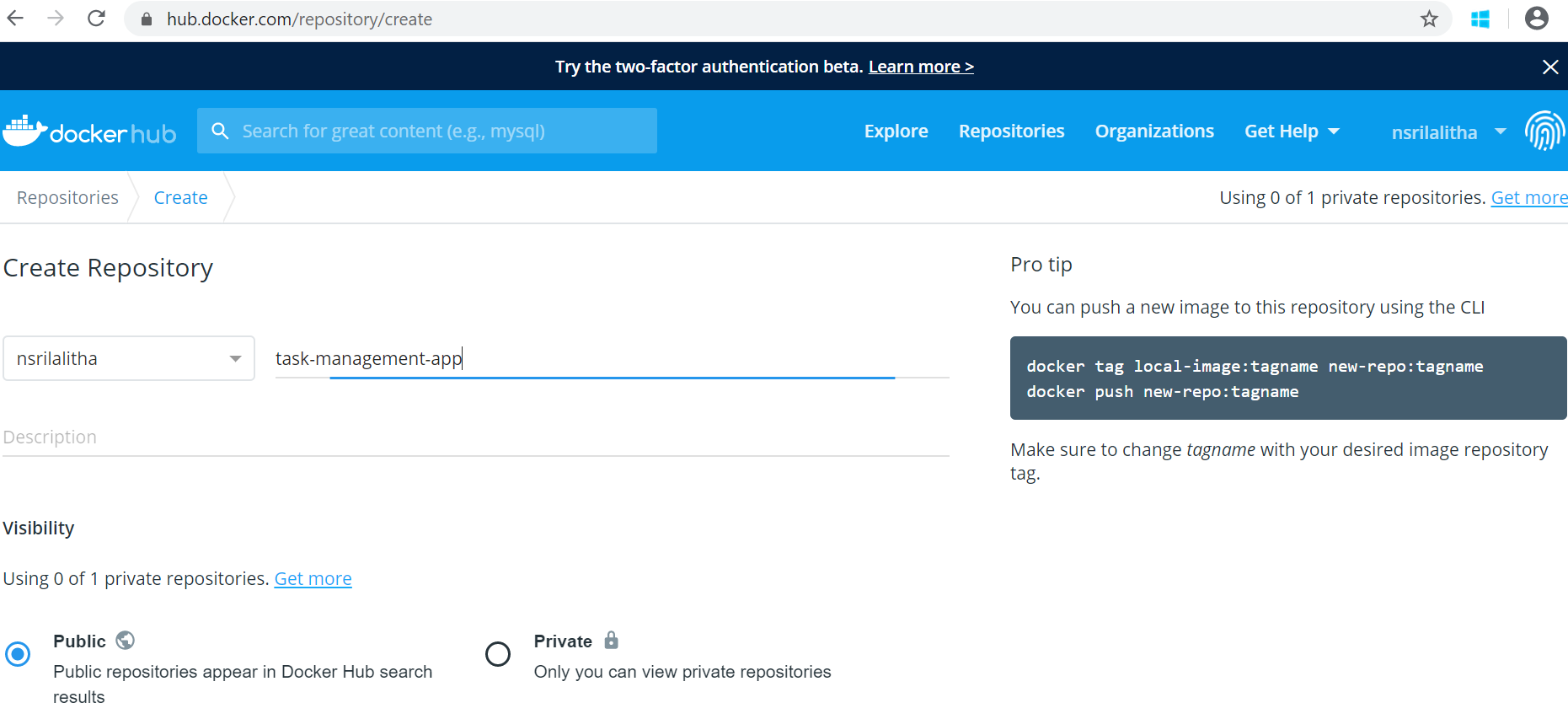


1. Login to docker hub from command prompt using below command

**sudo docker login**



1. To push “task-management-app” image to docker hub ,first Create a repository in Docker hub with name “task-management-app”.



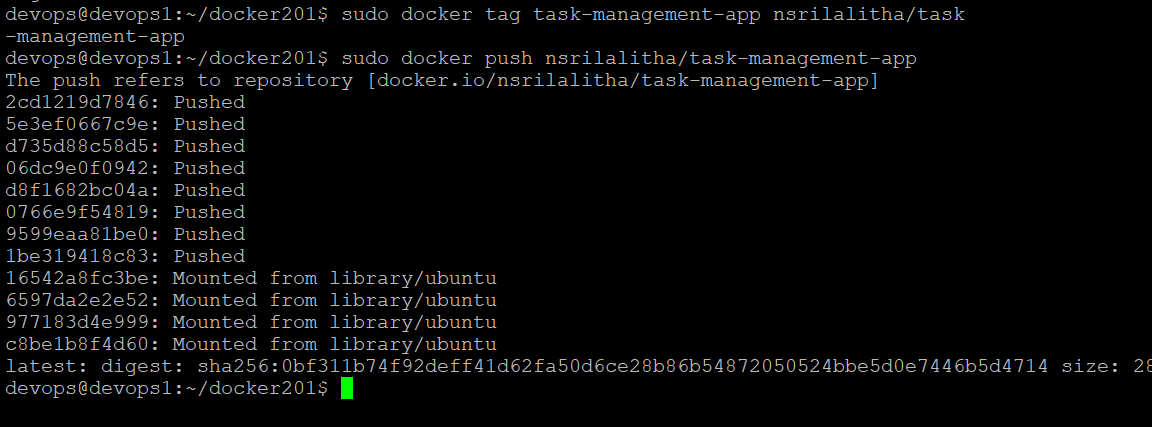


1. Now create a tag to link our image to docker repository. Use below command to create tag for this image

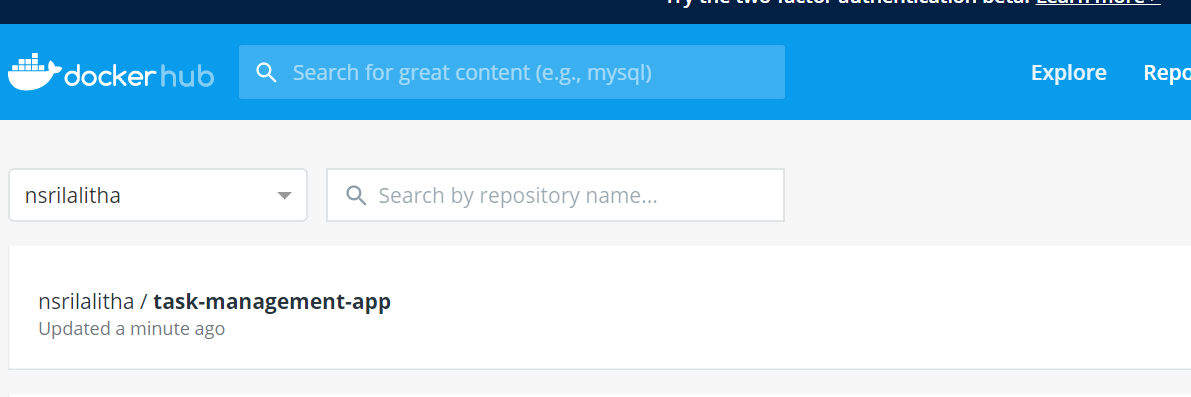
**sudo docker tag task-management-app nsrilalitha/task-management-app**

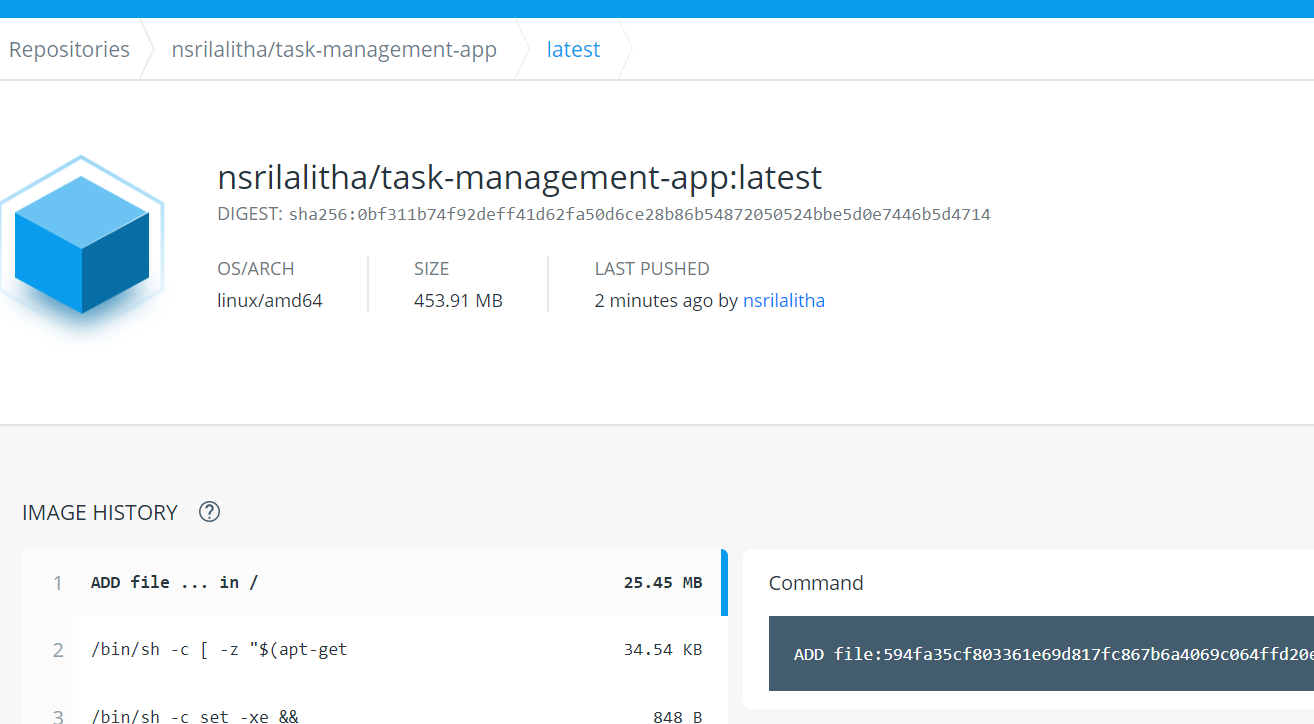
1. Use below command to push the image to created repo in docker hub

**sudo docker push nsrilalitha/task-management-app**



1. Now image is pushed to my docker hub account. Verify it in docker hub.

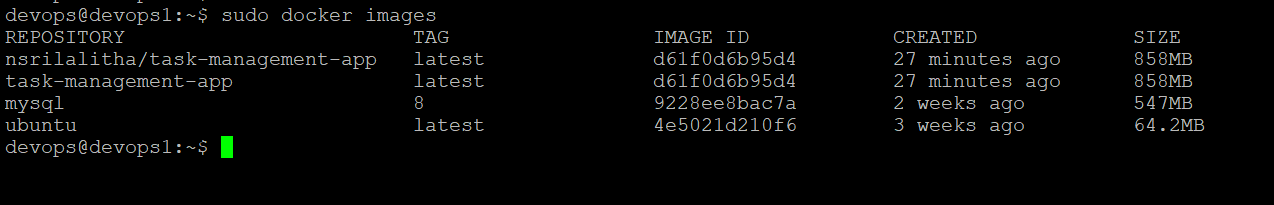




Now any one can access the docker image created for this application in below link

<https://hub.docker.com/repository/docker/nsrilalitha/task-management-app>

Now verify docker images.



* Create Kubernetes cluster or use managed Kubernetes platforms such as GKE, ECS-EKS, AKS etc.

Here I am going to setup Kubernetes cluster using Azure cloud lab. I have two hosts. So I will make one host machine as master and another host machine as worker node. I will join worker node with master node. Inorder to start with kubernetes, first I will be installing Kubernetes.

1. Assign unique names to both hosts machines

I have two host machines namely devops0 and devops1. I will configure devops0 as master node and devops1 as worker node. In order to avoid confusion, first I will give unique names to these hosts.

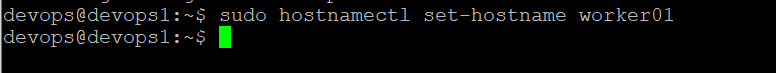
For devops0, use below command to set host name as “master-node”

**sudo hostnamectl set-hostname master-node**



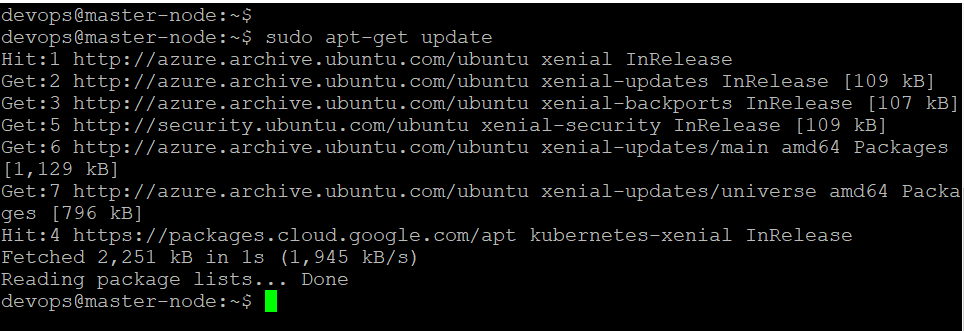
For devops1, use below command to set host name as “worker01”

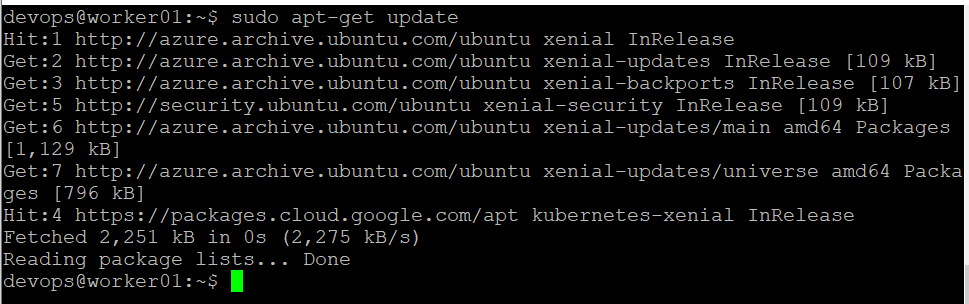
**sudo hostnamectl set-hostname worker01**



1. Update package list with below command

**sudo apt-get update**

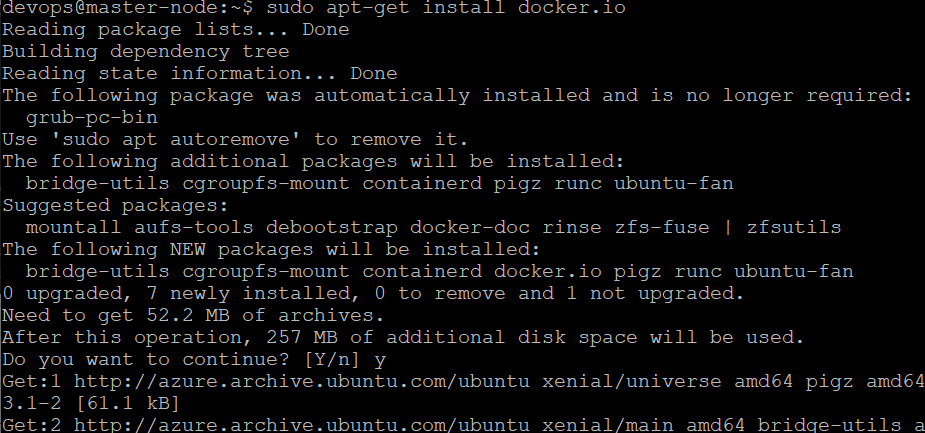


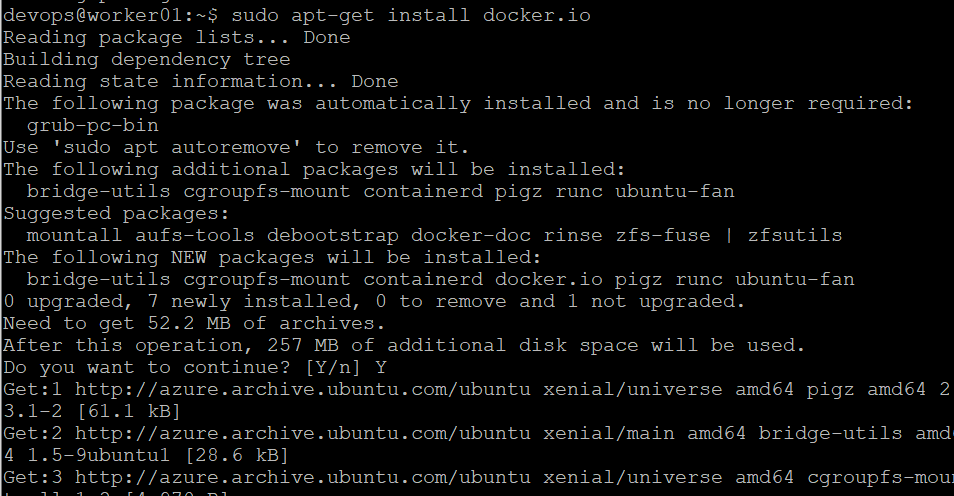


Since kubernetes uses container run time, to run the applications inside containers of pods. We should have docker installed in both master and worker nodes. Make sure docker is installed and if not install docker with below command.

1. Install docker using below command if docker is not present in any of the node

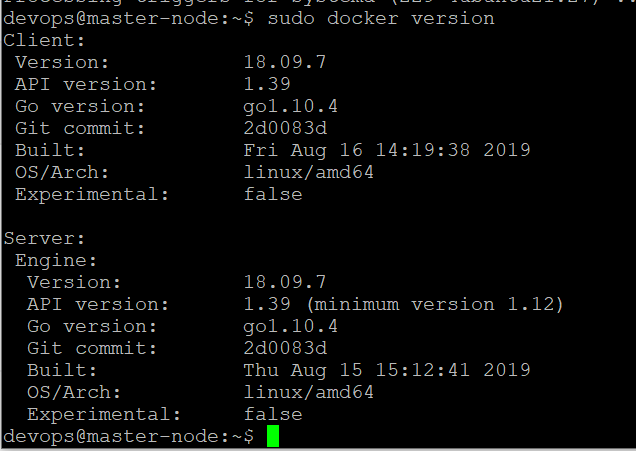
**sudo apt-get install docker.io**

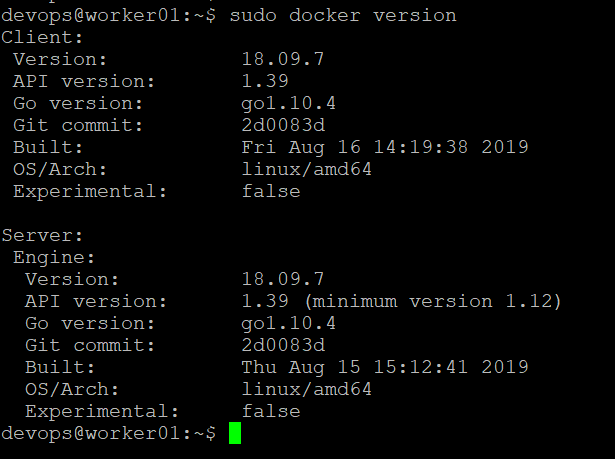




Verify docker is installed using below command

**sudo docker version**

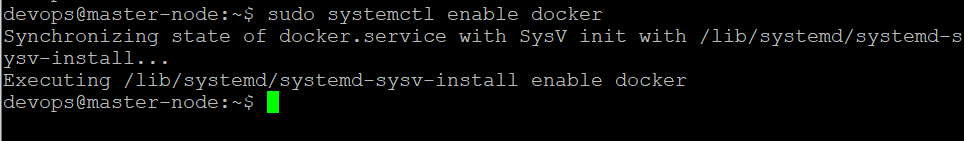


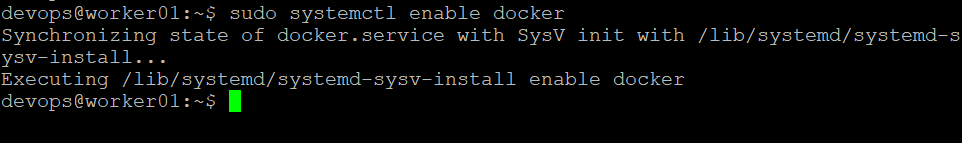


1. Now enable docker in both master and worker nodes.

Set docker to launch at boot by using below command

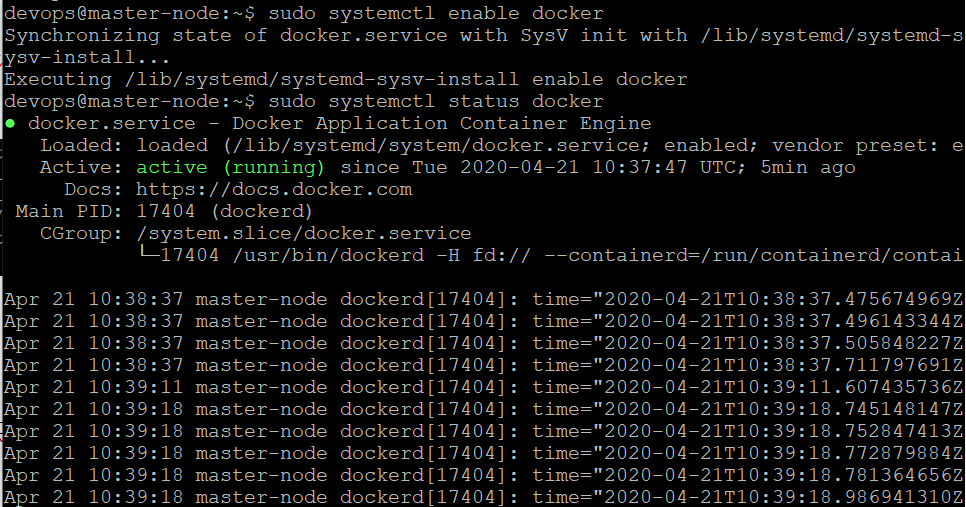
**sudo systemctl enable docker**





1. Now verify docker is running or not using below command

**sudo systemctl status docker**

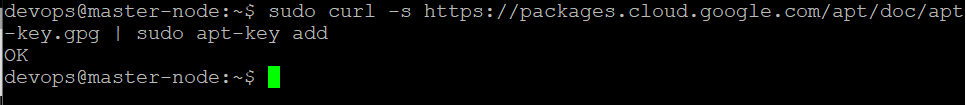


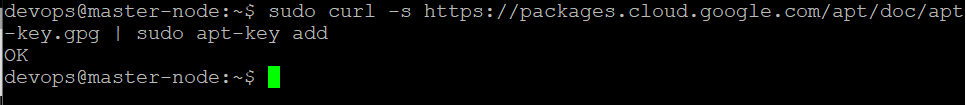


1. Add a signing key

Enter the following command to add a signing key

**curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add**

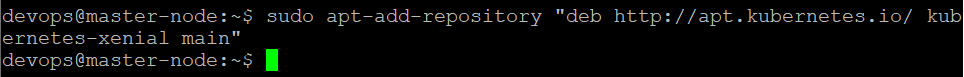


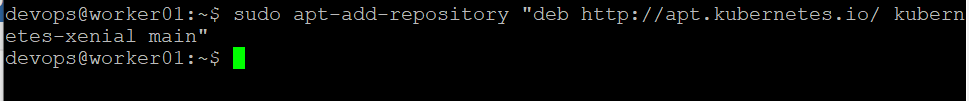


1. Add software repositories

Kubernetes is not included in the default repositories. To add them, enter the following:

**sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"**

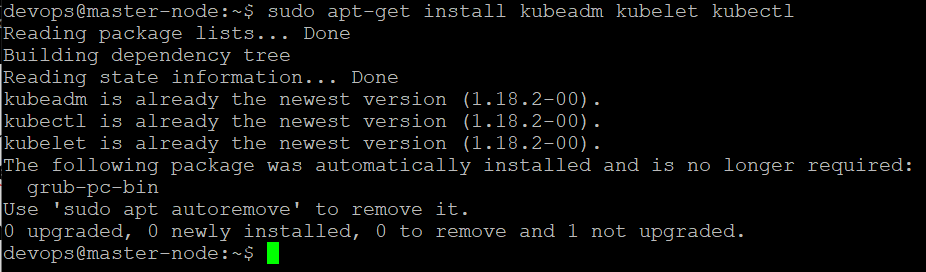


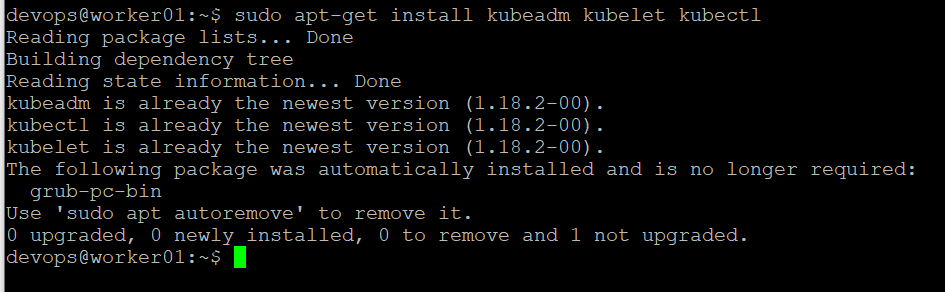


1. Now install kubernetes using **Kubeadm**

**Kubeadm** (Kubernetes Admin) is a tool that helps to initialize a cluster. Here I am using Kubeadm to setup kubernetes cluster with one master node and one worker node. So first install Kubeadm using below command. **Kubectl** is the command line tool to communicate with API server in master node which is entry point for the cluster. So with below command I am going to install Kubeadm, kubectl and kubelet.

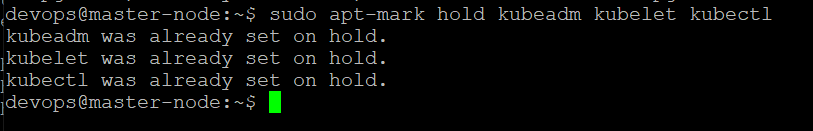
**sudo apt-get install kubeadm kubelet kubectl**

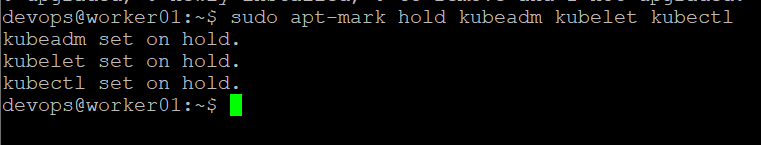




Now run below command to complete installation

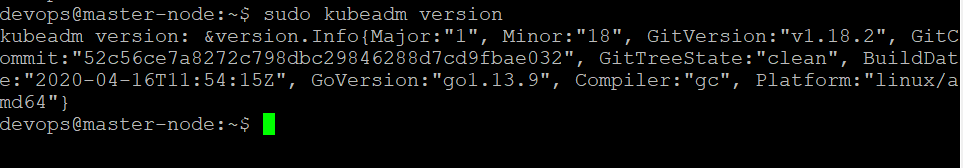
**sudo apt-mark hold kubeadm kubelet kubectl**





1. Now verify installation with below command

**kubeadm version**



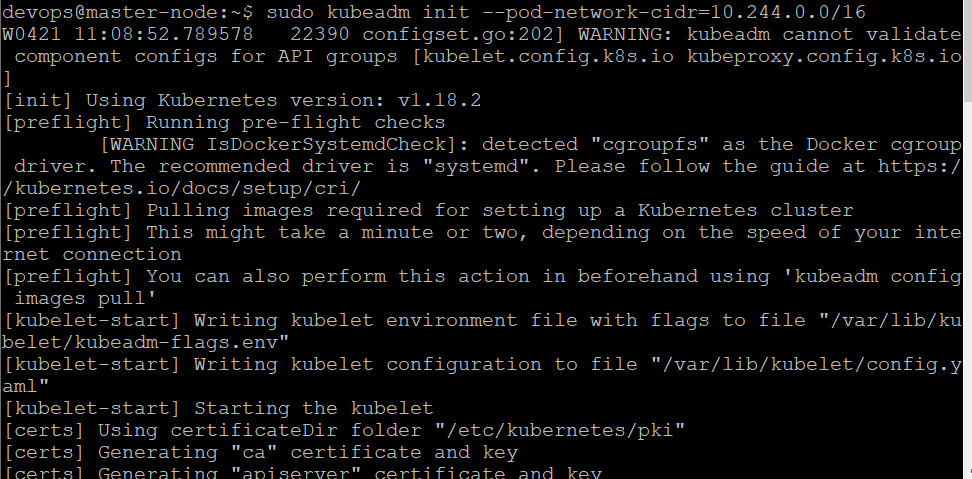


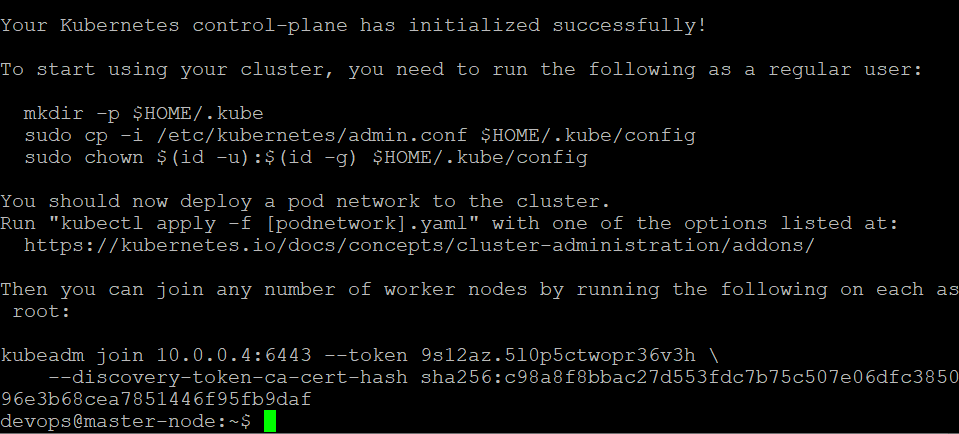
1. Initialize Kubernetes on master node

Now switch to master node and run following command to initialize kubernetes on master.

**sudo kubeadm init --pod-network-cidr=10.244.0.0/16**

Once this command finishes, it will display a **kubeadm join** message at the end. Make a note of the whole entry. This will be used to join the worker nodes to the cluster.





Here is my join command that I got from above command response.

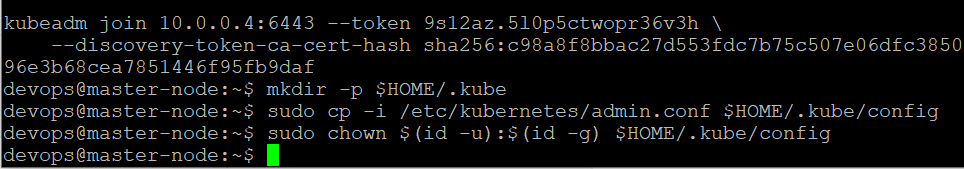
**kubeadm join 10.0.0.4:6443 --token 9s12az.5l0p5ctwopr36v3h --discovery-token-ca-cert- hash sha256:c98a8f8bbac27d553fdc7b75c507e06dfc385096e3b68cea7851446f95fb9daf**

1. Next, enter the following to create a directory for the cluster

**$ mkdir -p $HOME/.kube**

**$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**

**$ sudo chown $(id -u):$(id -g) $HOME/.kube/config**

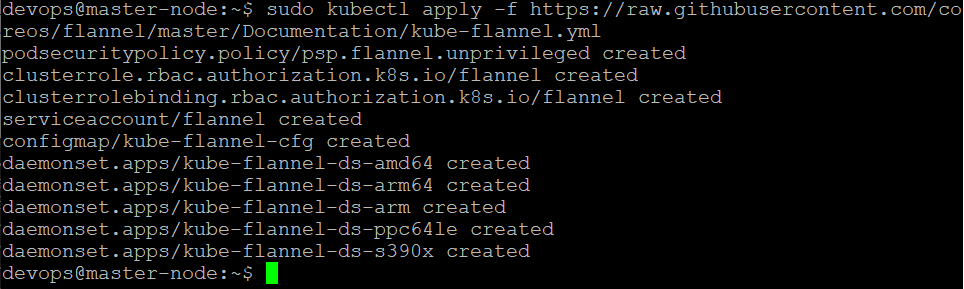


1. Deploy pod network to cluster

In order to establish communication between pods and nodes in a cluster we need pod network. Here I am using flannel virtual network to establish connection between nodes in a cluster.

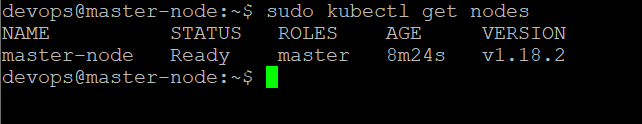
Enter the following command to deploy pod network

**sudo kubectl apply -f** [**https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml**](https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml)



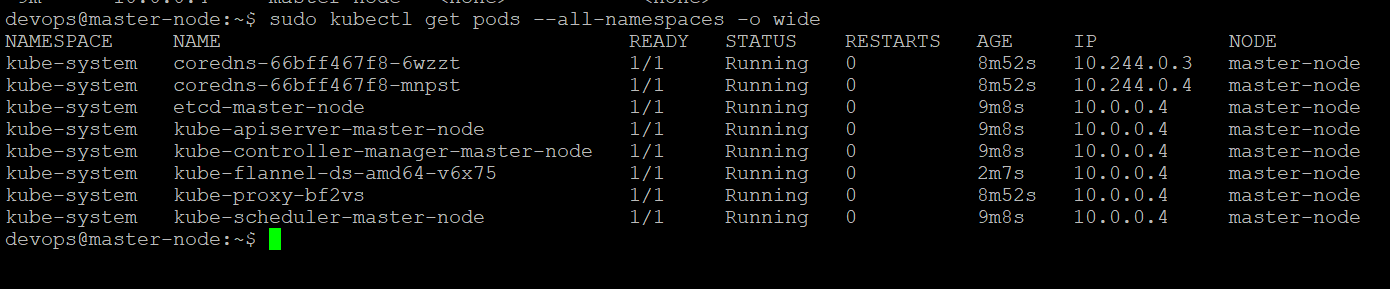
1. Verify that everything is running and communicating

**sudo kubectl get nodes**



1. Since Kubernetes is installed, Kubernetes by default containers will be running inside pods. Will verify those pods using below command

**sudo kubectl get pods --all-namespaces -o wide**



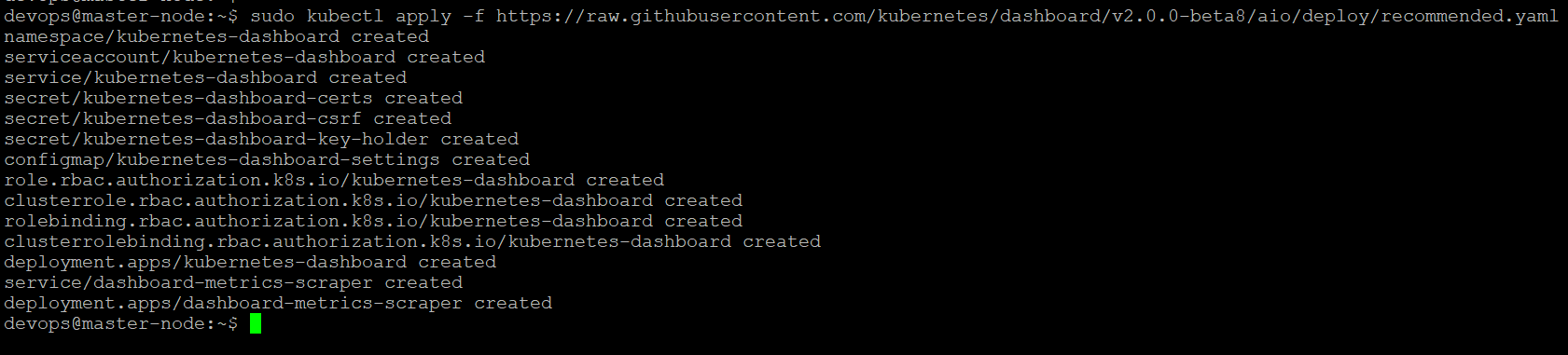
Verify **kube-apiserver**, **kube-proxy**, **kube-scheduler**, **kube-controller-manager** are master node components. These are running in master node.

**Etcd :** key-value storage. Etcd stores information related to kubernetes cluster.

**Etcd** also running in master node.

1. Now make dashboard up using below command

**sudo kubectl apply -f** [**https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml**](https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml)



Now I can access the dashboard using kubectl command line tool using below command

**sudo kubectl proxy**



Now to access dashboard in browser, use below url

<http://127.0.0.1:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>

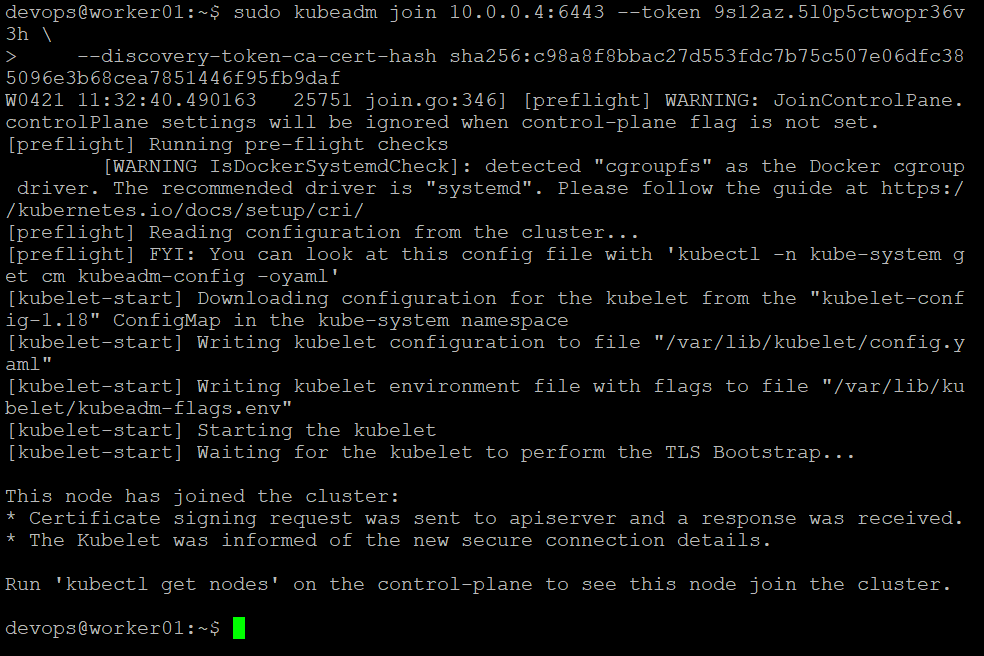
Here 127.0.0.1 is master node ip address.

The UI can only be accessed from the machine where the command is executed. Since I am using kubernetes in cloud lab, UI can’t be accessed in local machine browser.

1. Now join worker node to cluster

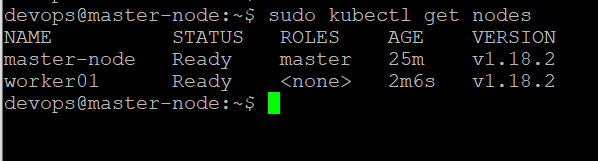
Switch to worker node “worker01” and run join command that we get from kubernetes init command output.

**kubeadm join 10.0.0.4:6443 --token 9s12az.5l0p5ctwopr36v3h --discovery-token-ca-cert-hash sha256:c98a8f8bbac27d553fdc7b75c507e06dfc385096e3b68cea7851446f95fb9daf**



Now switch to master node and verify status of nodes using below command

**sudo kubectl get nodes**



Worker node “worker01” is successfully joined the cluster. Now we are able to see both master and worker nodes by kubectl get nodes command.

Now my cluster is successfully setup with one master node and one worker node. Now I can create deployments and services to access containers. Next section, I will be working on yaml files to create deployments and services to run the application using kubernetes.

* Create a yml file for defining the application service structure and run the same using Kubernetes.

There are multiple ways to create deployment and service configuration files in kubernetes. We can create docker-compose file and use that file to convert it to deployment and service files using **Kompose** or we can manually create deployment and service configuration files and by using kubectl command line tool we can create deployments and access the application. I will be using both approaches in this section.

**Kompose:** Kompose is a tool which is used to convert dockerized application to make it run in kubernetes by simply converting the docker-compose file into deployment and service configuration files.

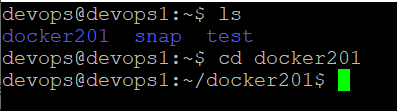
In the docker-compose file we have to specify services for the application and its dependent services like MySQL. So for this application, we need two services - One for application and one for mysql. We can establish connection between both containers and to run application with Docker compose.

To spin both containers at a time, the docker-compose file can be used. While writing the configuration, we need to write the setting for the database first and then the application build since the application is dependent on the database.

Now we will start creating docker-compose.yml file in docker201 folder since this folder has our Dockerfile for this application.

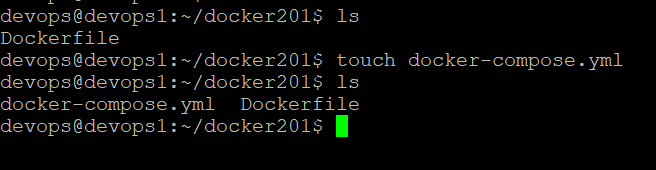
1. Now go to docker201 folder

**cd docker201**



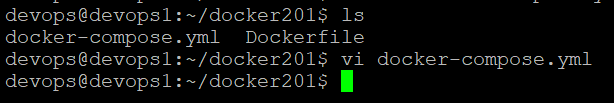
1. Create docker-compose file with name docker-compose.yml

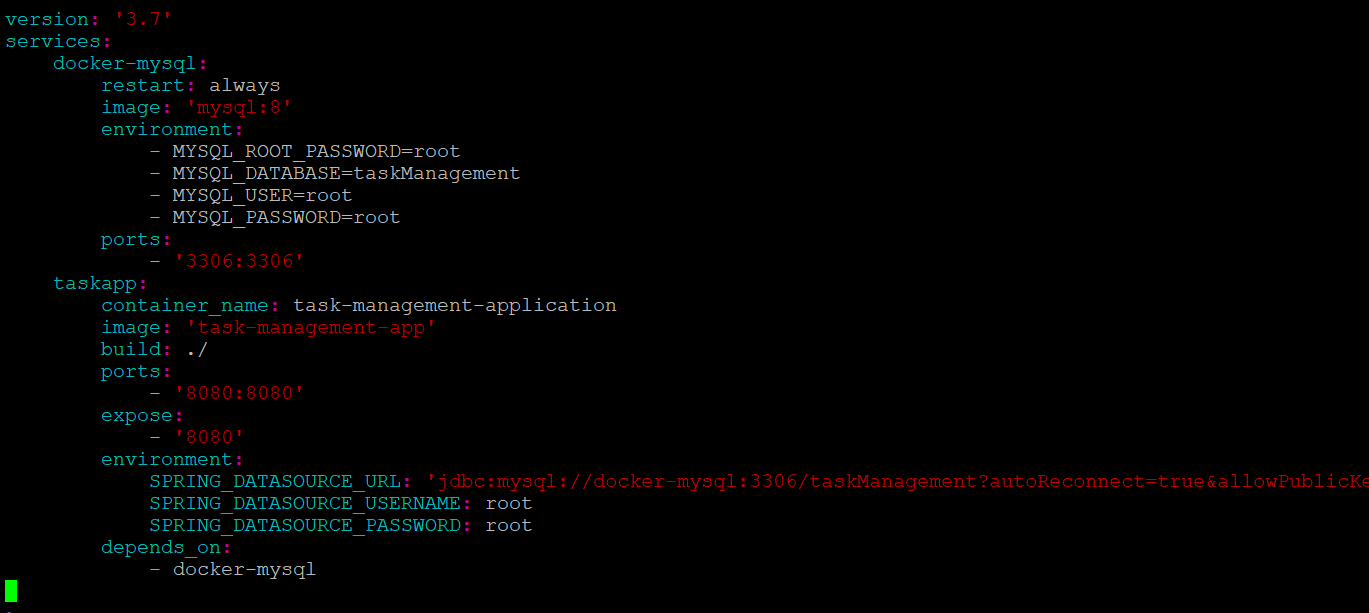
**touch docker-compose.yml**



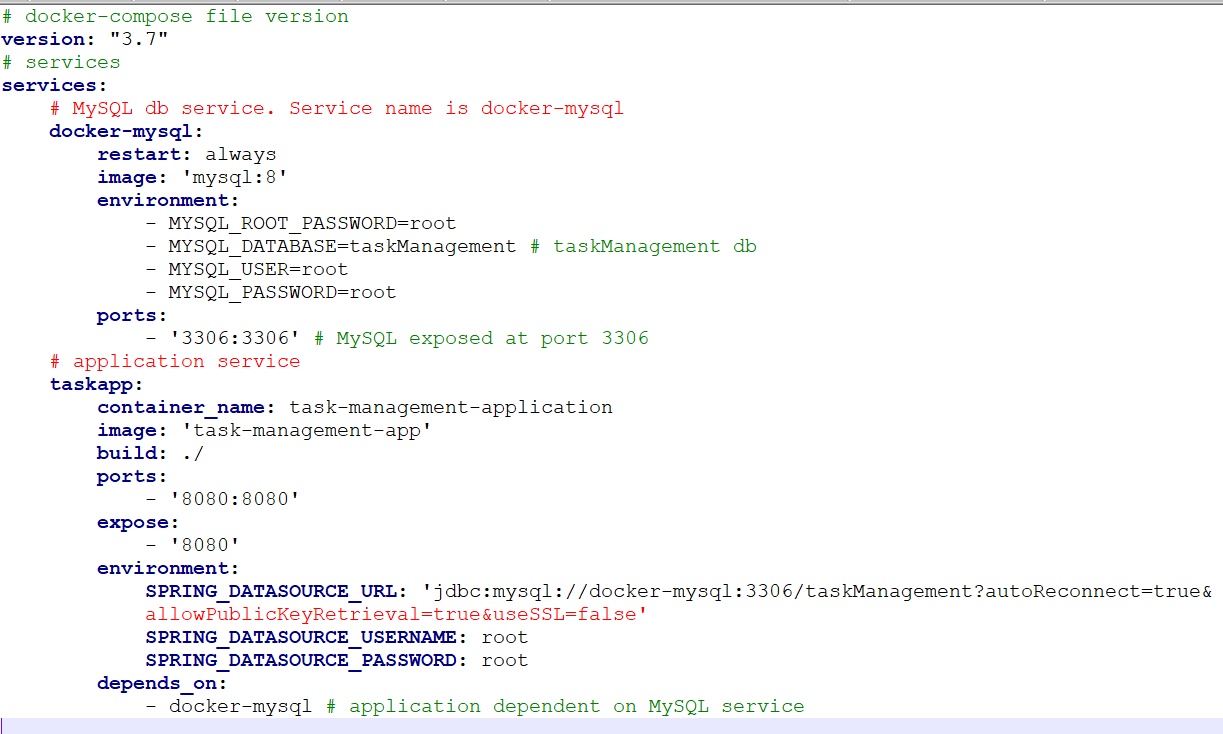
1. Now docker-compose file is created. We have to edit and add contents in docker-compose file to make application run at port 8080. To edit use below command.

vi docker-compose.yml





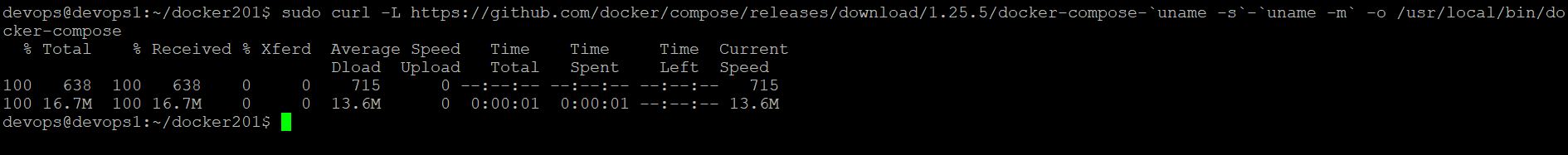




Here I have defined two services – one for spring boot application, one for MySQL database. For MySQL service, gave name as ‘docker-mysql’ which will be a docker container. Base image is “mysql with version 8”. Exposed port as “3306” and specified environment variables like username, password, db name. When we started the application using docker-compose up, a docker container will be created for MySQL with name “docker-mysql” which will be running at port 3306. For application, gave base image as “task-management-app” which we created earlier. Application is exposed at port 8080 and depends on “docker-mysql” container.

1. Now docker-compose file ready. In order to use docker-compose first we should install docker-compose. Use below command to install docker-compose.

**sudo curl –L** [**https://github.com/docker/compose/releases/download/1.25.5/docker-compose-‘uname**](https://github.com/docker/compose/releases/download/1.25.5/docker-compose-‘uname) **-s’-‘uname -m’ –o /usr/local/bin/docker-compose**

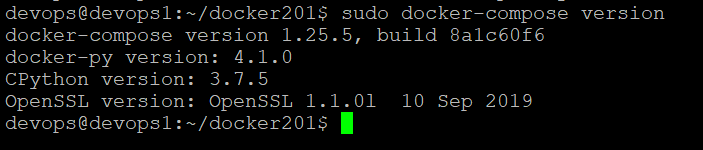


Now set permissions using below command.

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

Now verify version using below command

**sudo docker-compose –version**



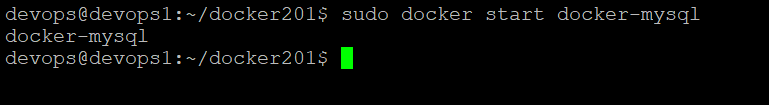
1. To start the application using docker-compose, we should ensure the dependent services are started. Here the application is dependent on docker-mysql container, before starting our application docker-mysql container should be up and running at port 3306. To verify docker-mysql is running or not, use below command.

**sudo docker container ps –a**



For “docker-mysql” container, status is showing as “Exited”. So start “docker-mysql” container using below command.

**sudo docker start docker-mysql**



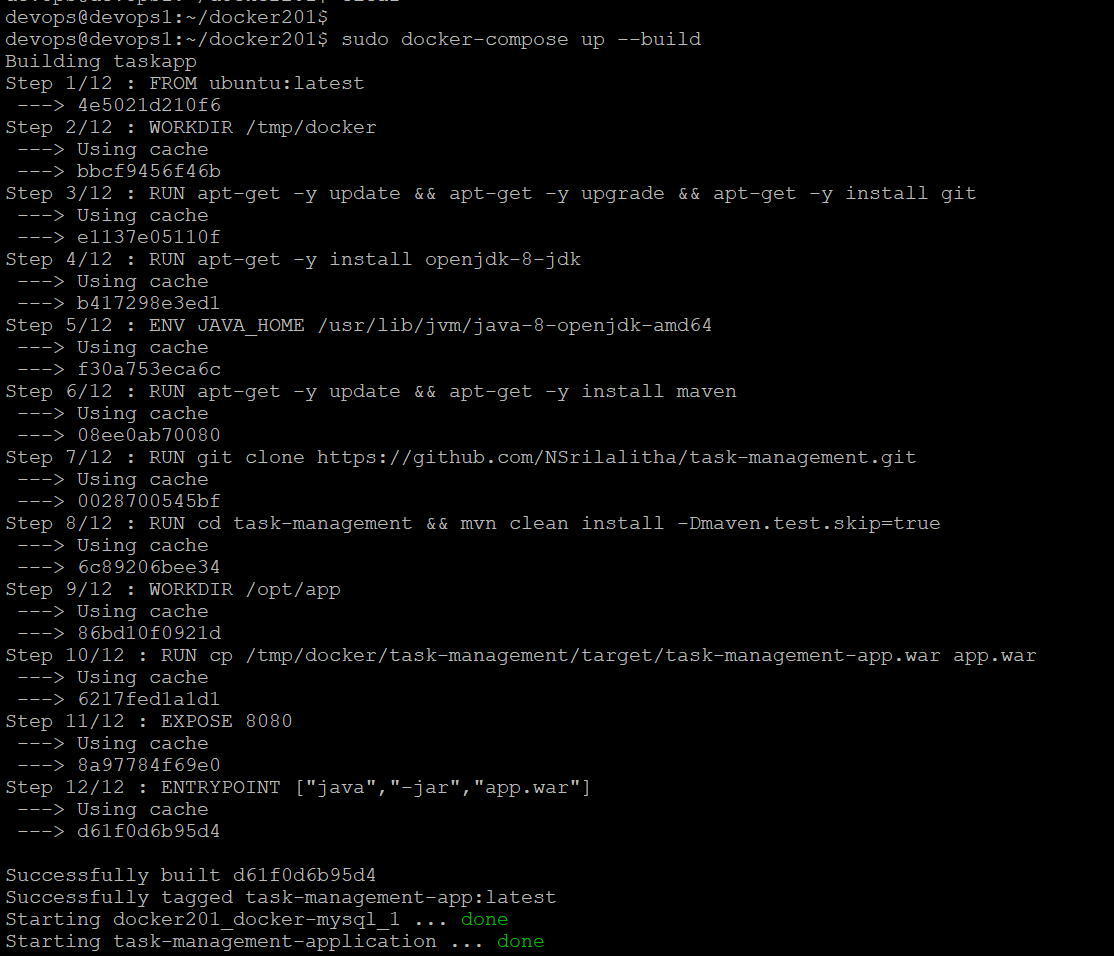
Now verify status of “docker-mysql” container, by again running sudo docker container ps –a command.



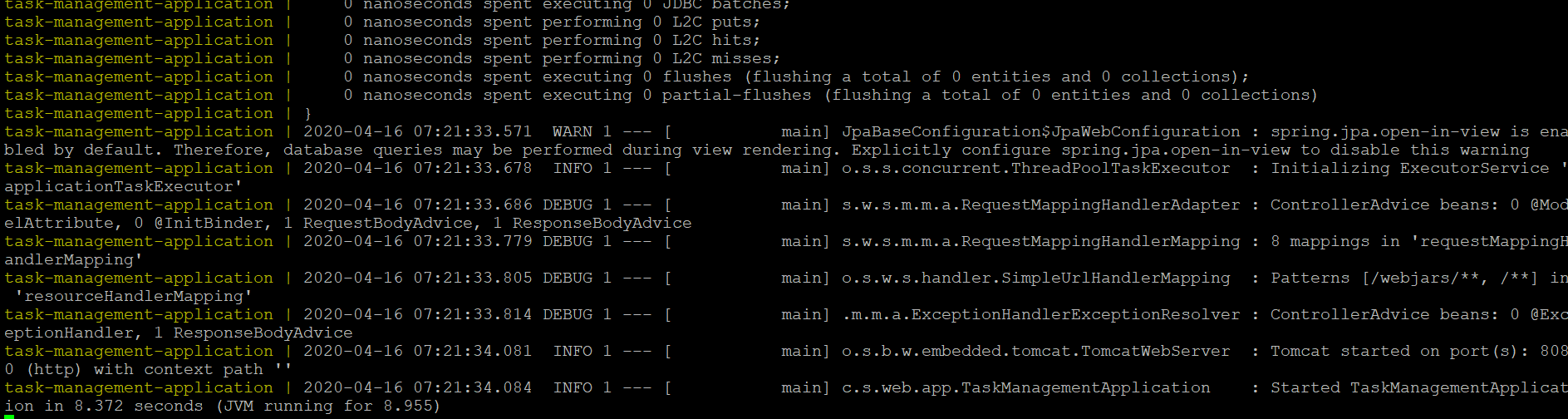
Now status of “docker-mysql” container is “Up”. Since dependencies are ready, Now we can run our application using docker-compose.

1. We can scale up and down the containers using docker compose. Now we will be scaling up the containers using below command. With this, application will be running in docker container. The network is established between dependent containers using docker-compose.

**sudo docker-compose up**





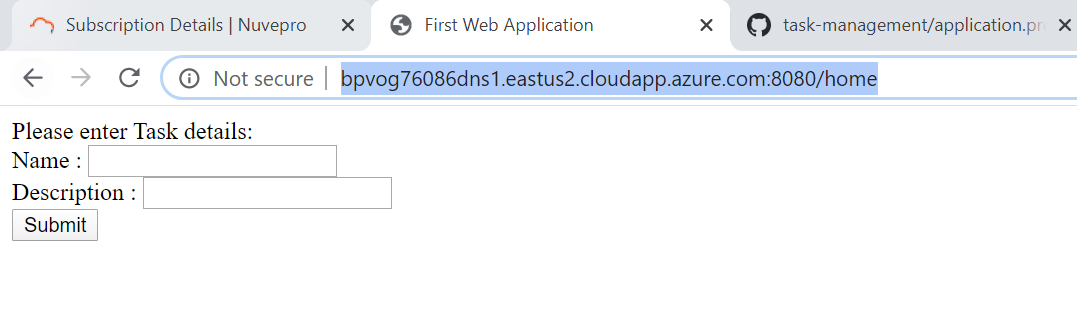


Now application is started at port 8080.

Cloud lab host is bpvog76086dns1.eastus2.cloudapp.azure.com

So we can access this application home page at below url

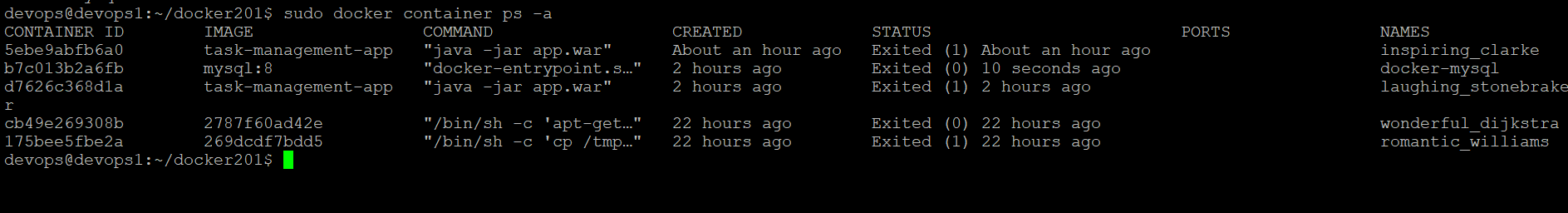
<http://bpvog76086dns1.eastus2.cloudapp.azure.com:8080/home>



We can stop the docker-compose using below command.

**sudo docker-compose stop**

Now verify containers status using “docker container ps –a”



For all the containers, now status is showing as “Exited” since docker-compose is stopped.

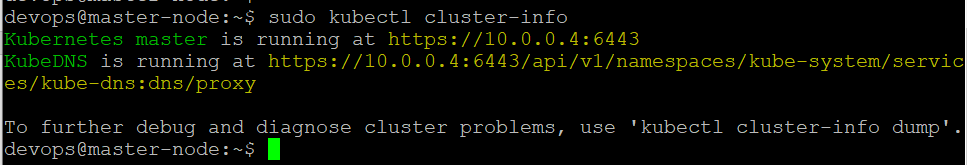
Here by using docker-compose file I am able to successfully run the application. Now I will make use of this docker-compose file and convert to deployment and service configuration files using **Kompose**. So that I can run my application and manage the containers using **Kubernetes**.

Either we can use Kompose to convert docker-compose to generate deployment and service configuration yaml files to run application using kubernetes or we can create our own deployment and service configuration files. Since I already have a docker-compose file, I will use Kompose to use this file to generate deployment and service configuration files.

I already setup kubernetes cluster in previous section. Here I have one master node and one worker node. Joined worker node to cluster. To start with Kompose cluster should be ready and Kubectl command line tool should be present. Since I have both I can start with yaml files.

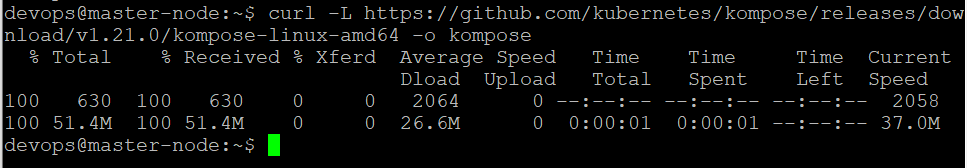
1. Verify cluster information using below command

**sudo kubectl cluster-info**



1. Install Kompose using below command.

**curl -L https://github.com/kubernetes/kompose/releases/download/v1.21.0/kompose-linux-amd64 -o kompose**

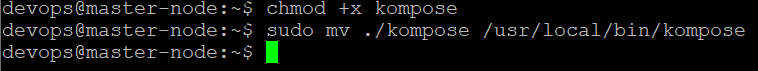


Make the binary executable using below command

**chmod +x compose**

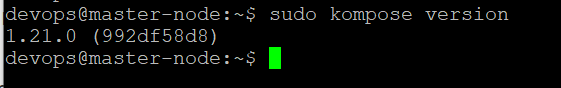
Move it to your PATH

**sudo mv ./kompose /usr/local/bin/compose**



Now Kompose is successfully installed. Verify installation using below command.

**sudo kompose version**

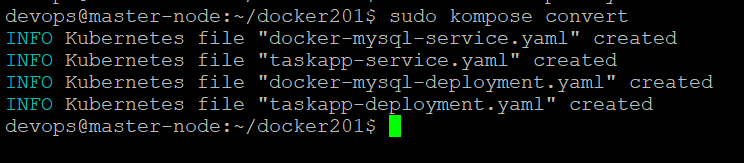


1. We can use Kompose up or Kompose convert and followed by Kubectl apply command.

Here I am using Kompose convert command. This command generate deployment and service configuration files for the services mentioned in the docker compose file. We can create deployments and services using Kubectl apply command.

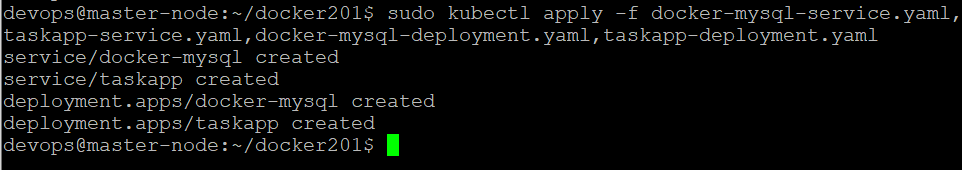
Since my docker-compose file is present in docker201 folder, switch to that folder and run below command

**sudo kompose convert**



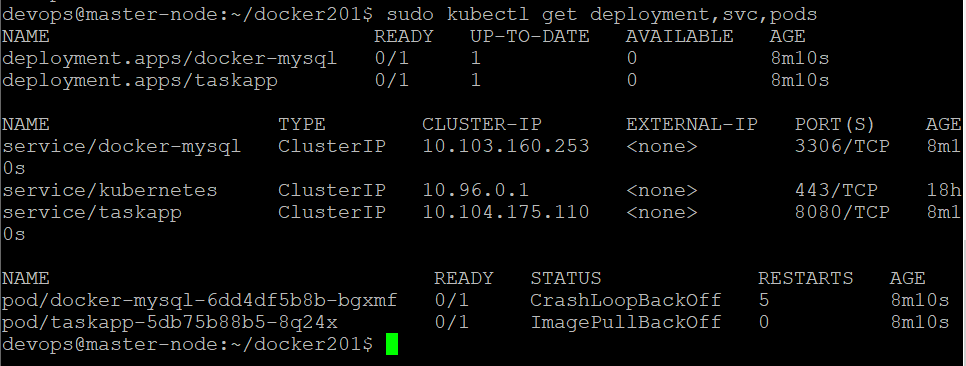
My docker-compose file has two services – task-app and docker-mysql. So Kompose created deployment and service configuration files for these two services. Now to create deployment and service I can use kubectl apply command as shown below.

**sudo kubectl apply –f taskapp-service.yaml, taskapp-deployment.yaml, docker-mysql-service.yaml, docker-mysql-deployment.yaml**



1. We can get deployments, services, replicasets information using below command

**sudo kubectl get deployment, svc, pods**



In this way we can use **Kompose** to convert docker-compose to generate Kubernetes deployment and service configuration files.

* Use Deployments / Replication sets

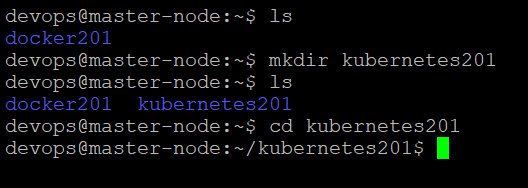
Next I am going to create these configuration yaml files manually. I will be creating two namespaces namely db-namespace and app-namespace. In Kubernetes “namespace” is used to group related resources. Deployments, services, pods all are kubernetes resources. So I will group mysql related resources means mysql-deployment, mysql- service and mysql-pods in db-namespace. And I will group application specific resources namely task-app-deployment, task-app-service and pods in app-namespace. To create all these configurations I am going to create new folder.

1. Create new folder to store these configuration files.

**mkdir kubernetes201**

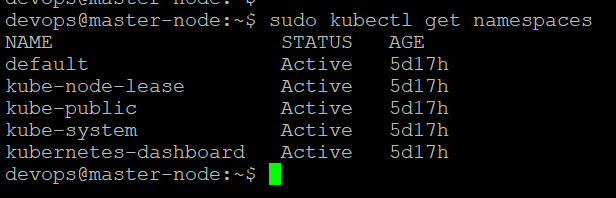
1. Switch to kubernetes201 folder

**cd kubernetes201**



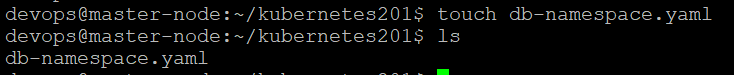
1. Before creating namespaces for this application lets verify default existing namespaces in kubernetes using below command

**sudo kubectl get namespaces**



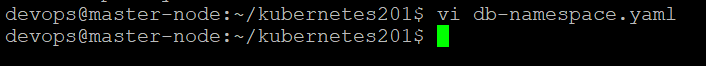
1. Now create namespace for database to group MySQL related resources

**touch db-namespace.yaml**



1. Edit db-namespace.yaml to provide configuration as shown below

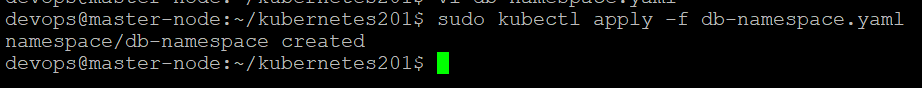
**vi db-namespace.yaml**





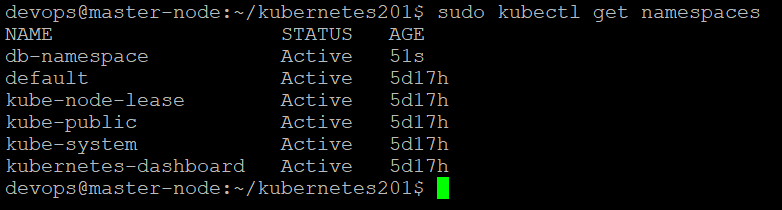
1. Now create db-namespace using kubectl apply command as shown below.

**sudo kubectl apply –f db-namespace.yaml**



db-namespace is created. Lets verify all the namespaces using below command

**sudo kubectl get namespaces**



* Ensure data persistence by mounting the data outside of the containers

1. I am going to create following yaml files to create dockerized MySQL database, running in a separate container.

mysql-persistent-volume.yaml

mysql-persistent-volume-claim.yaml

mysql-deployment.yaml

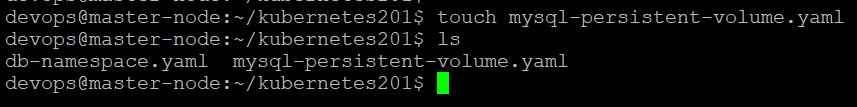
mysql-service.yaml

Here I am persisting data outside of the containers using persistent volume and persistent volume claim. I will give reference to this persistent volume and claim in persistent storage in mysql-deployment configuration file.

Now lets start creating these files one by one.

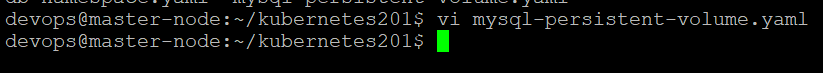
1. Create mysql-persistent-volume.yaml using below command

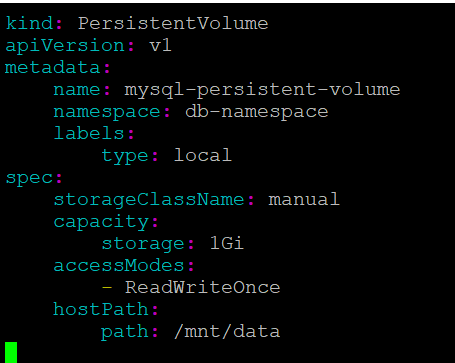
**touch mysql-persistent-volume.yaml**



Now edit this configuration file as shown below.

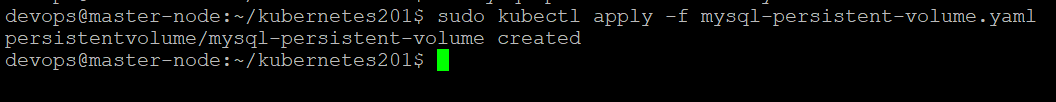
**vi mysql-persistent-volume.yaml**





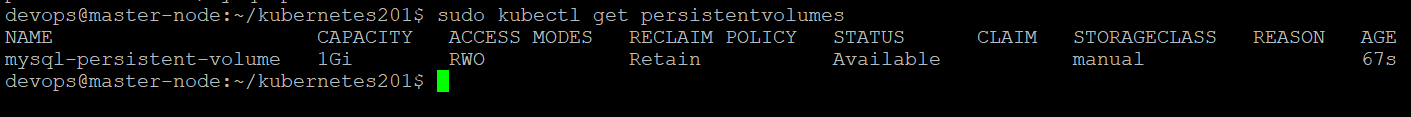
Now create persistent volume using below command

**sudo kubectl apply –f mysql-persistent-volume.yaml**



Now to view all persistent volumes, use below command

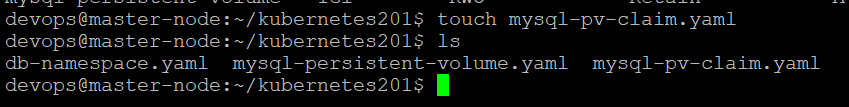
**sudo kubectl get persistentvolumes**



1. Now create claim for persistent volume

Create mysql-pv-claim.yaml file using below command

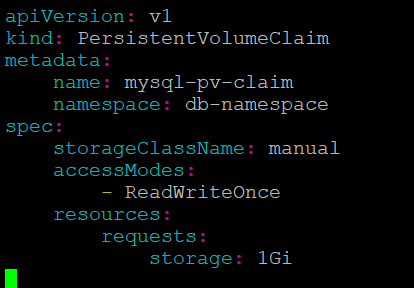
**touch mysql-pv-claim.yaml**



Now provide configuration as shown below

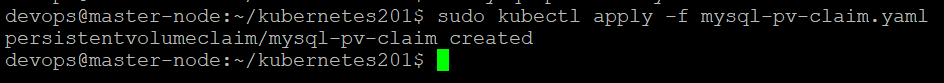
**vi mysql-pv-claim.yaml**





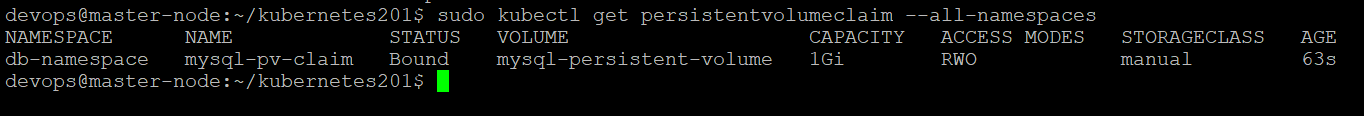
Now create persistent volume claim using below command

**sudo kubectl apply –f mysql-pv-claim.yaml**



To view all persistent volume claims use below command

**sudo kubectl get persistentvolumeclaim –all-namespaces**



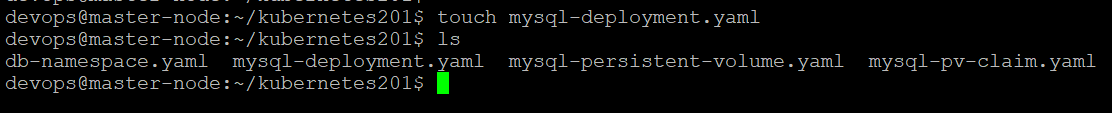
If we don’t specify –all-namespaces or –namespace=<namespace-name> then by default it looks for the mentioned resource in “default” namespace. Since this resource is created in db-namespace it will display “no resources found” message.

1. Now create deployment file for MySQL

Deployment is the abstraction over pods. In Kubernetes containers will reside inside pods of node. A node can contain one pod or multiple pods. Similarly a pod may contain one container or multiple containers. The specification of pod should be mentioned in deployment configuration file. Replicas needed for the application also specified in deployment file. For MySQL I need only one replica, one pod in which MySQL dockerized container will be available at port 3306.

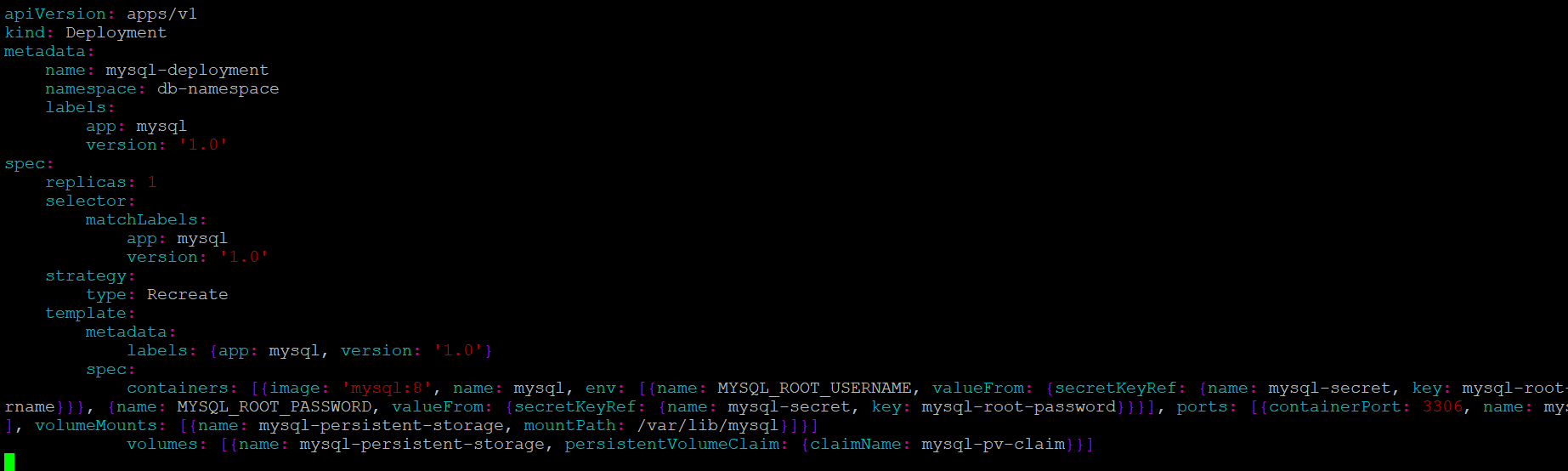
Create mysql-deployment.yaml file using below command.

**touch mysql-deployment.yaml**



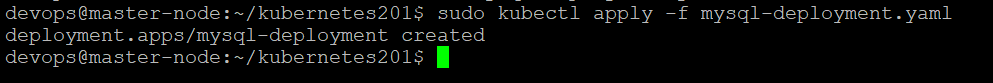
Edit configuration file as shown below

**vi mysql-deployment.yaml**



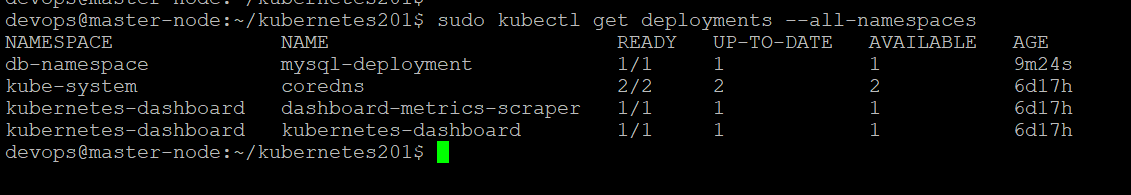
Now create MySQL deployment using below command

**sudo kubectl apply –f mysql-deployment.yaml**



Now get all deployments using below command

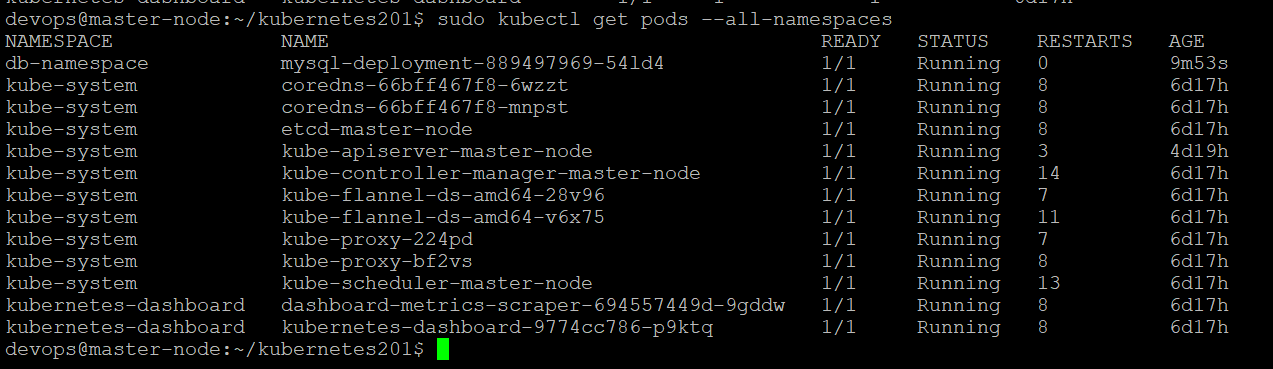
**sudo kubectl get deployments –all-namespaces**



Since deployment is abstraction over pods and replicas, when we create deployment it interneally creates pods and replicasets. So view all pods and replica sets.

Use below command to view pods

**sudo kubectl get pods –all-namespaces**



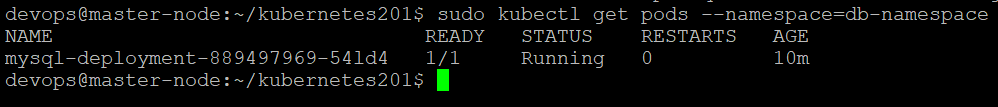
Here mysql pod is present in db-namespace and status is showing as running.

To verify pods specific to db-namespace use below command

**sudo kubectl get pods –namespace=db-namespace**

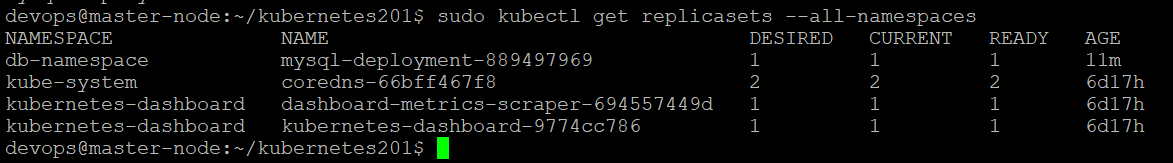
(or)

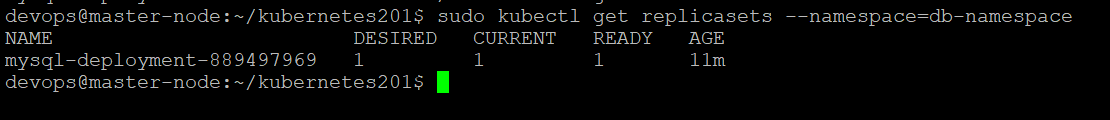
**sudo kubectl get pods –n db-namespace**



Verify replicasets using below command

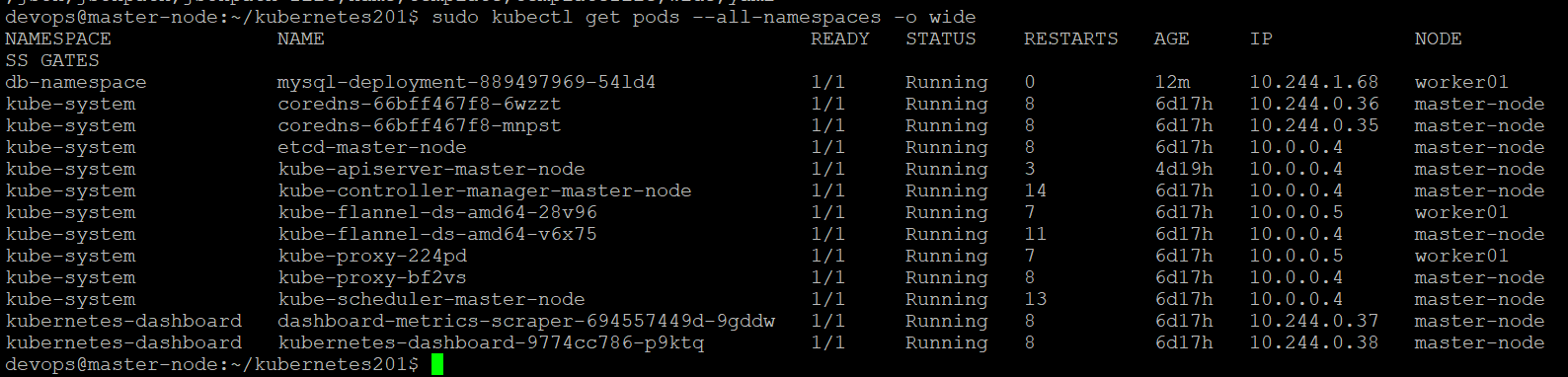
**sudo kubectl get replicasets –all-namespaces**



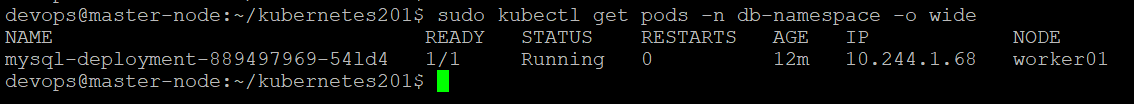


Since I have mentioned replica count as “1” in my mysql-deployment.yaml file, 1 replica is created. Verify desired and current count is same and it is “1”.

Now successfully my dockerized MySQL database is running inside pod of worker node “worker01”.



If we want resources specific to only db-namespace then mention namespace flag

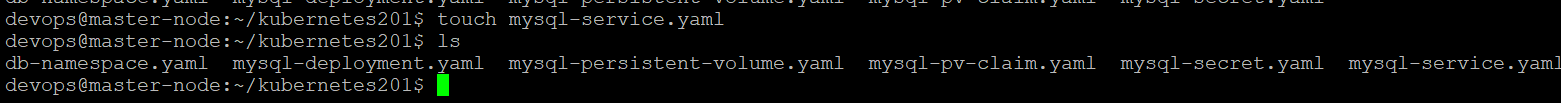


1. Create service for MySQL

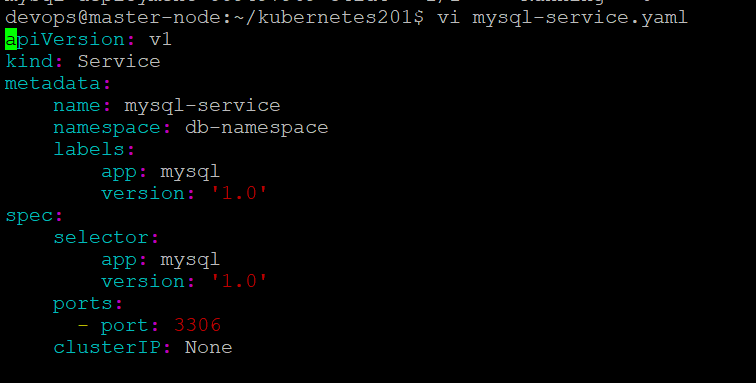
In kubernetes, in order to communicate with containers inside pod, we need services. Since in kubernetes it is very common that pods easily dies and recreate. When a pod dies, the newly created pod won’t get same ip address. So it is always recommended to communicate with containers inside pod through a service.

So lets create mysql-service.yaml file to create service for MySQL

**touch mysql-service.yaml**

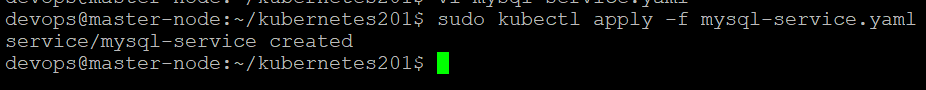


Edit mysql-service.yaml configuration file as shown below



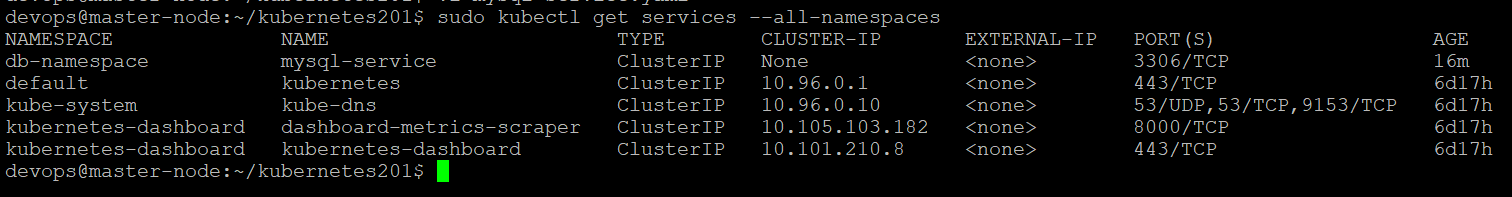
Now create service using kubectl apply command as shown below

**sudo kubectl apply –f mysql-service.yaml**



To view all services, use below command

**sudo kubectl get services –all-namespaces**

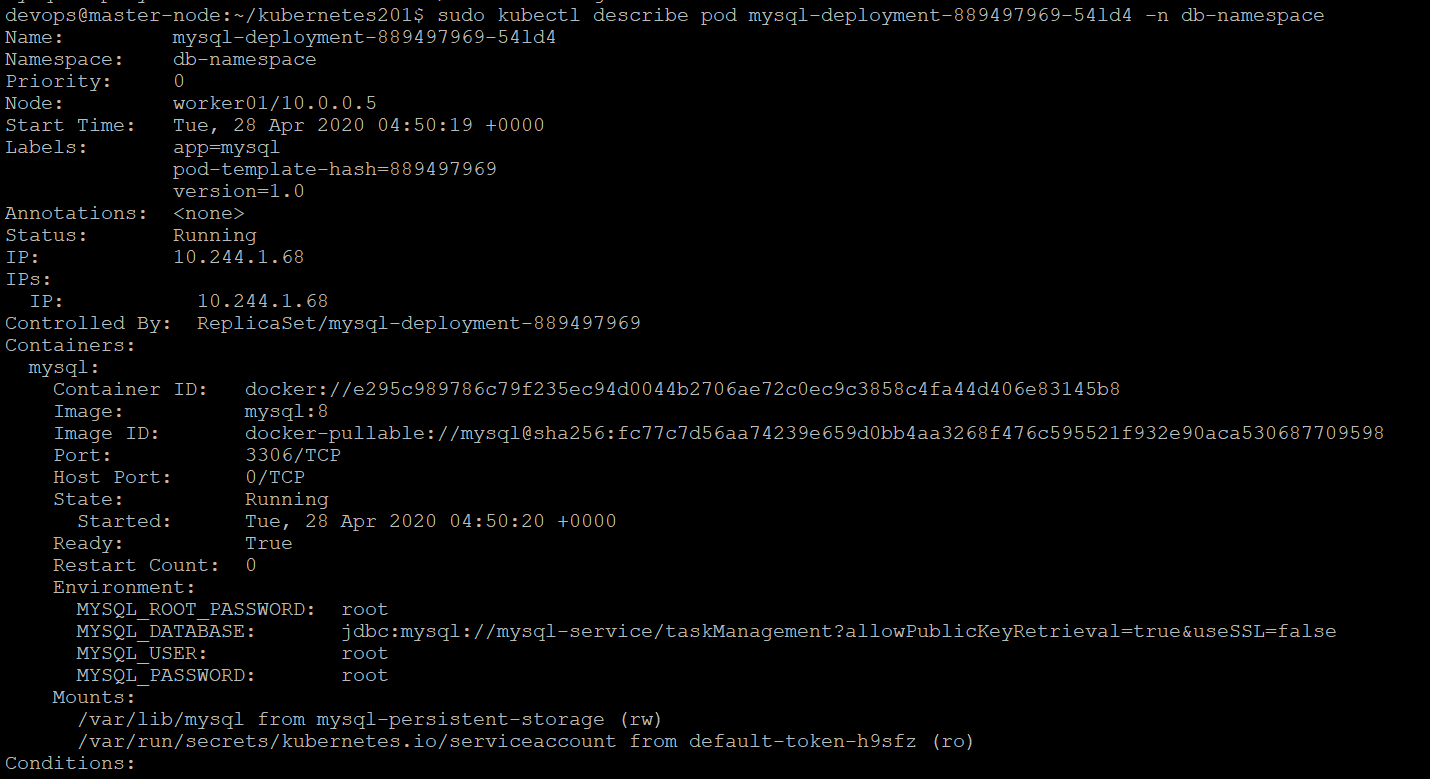


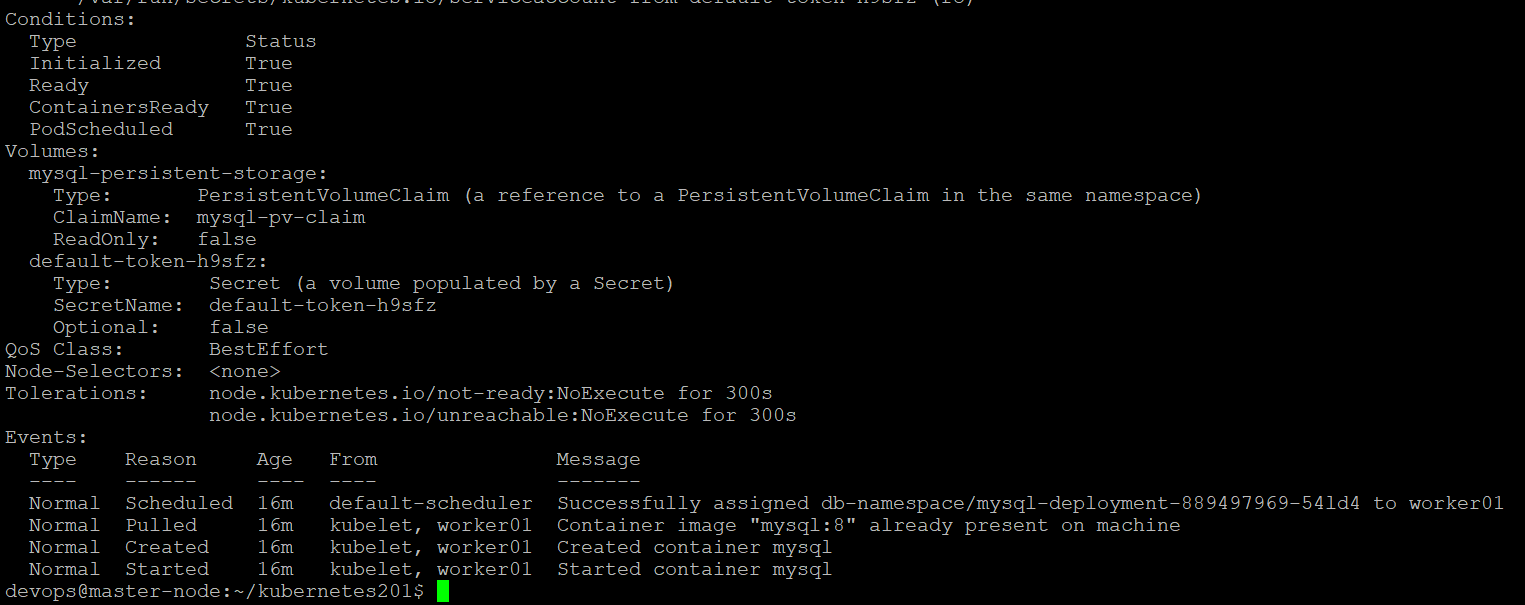
Now my MySQL is ready and available at port 3306.

1. To view information about pod, deployment, replicaset we can use kubectl describe command.

Lets verify information about mysql pod created as per mysql-deployment specification using below command.

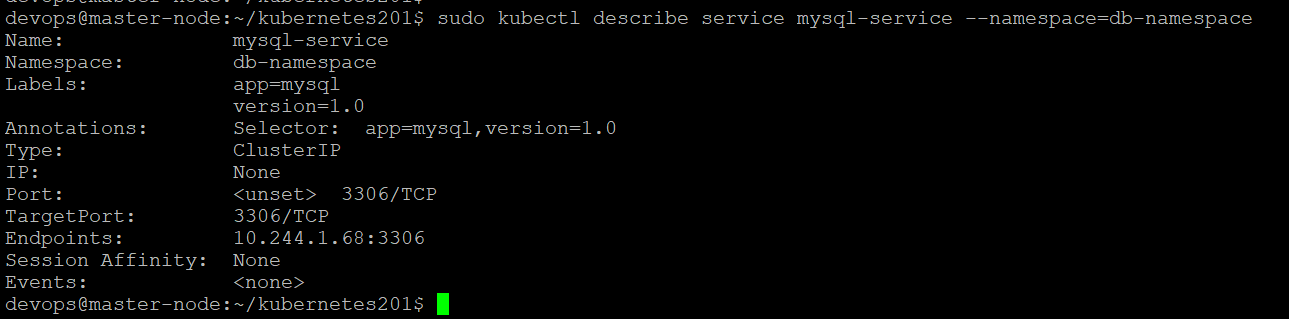
**sudo kubectl describe pod <pod-name> -n db-namespace**





To view information about service use below command

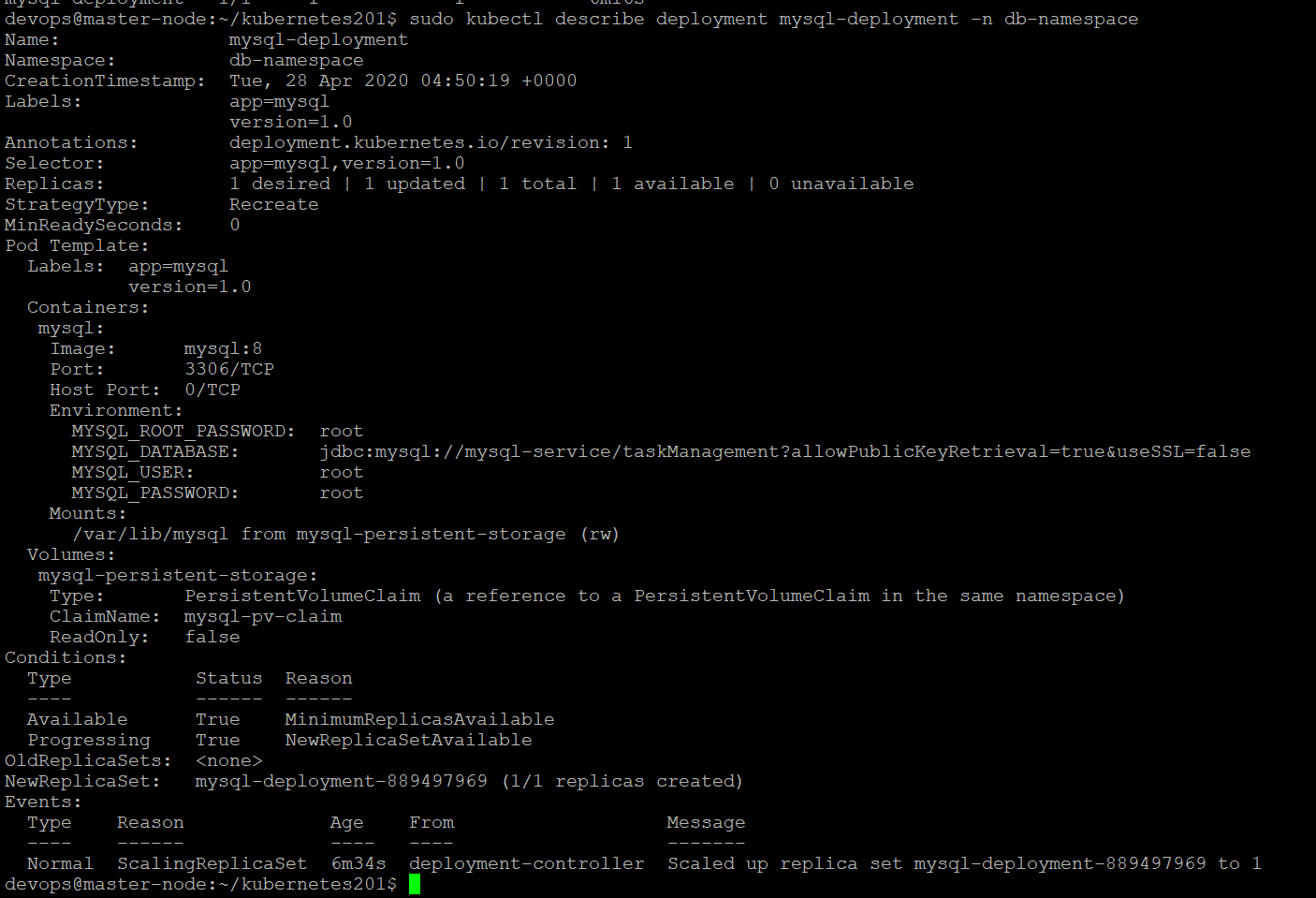
**sudo kubectl describe service mysql-service –n db-namespace**



Endpoints describes ip address of pod and port that is available to open. Here ip address of pod is 10.244.1.68

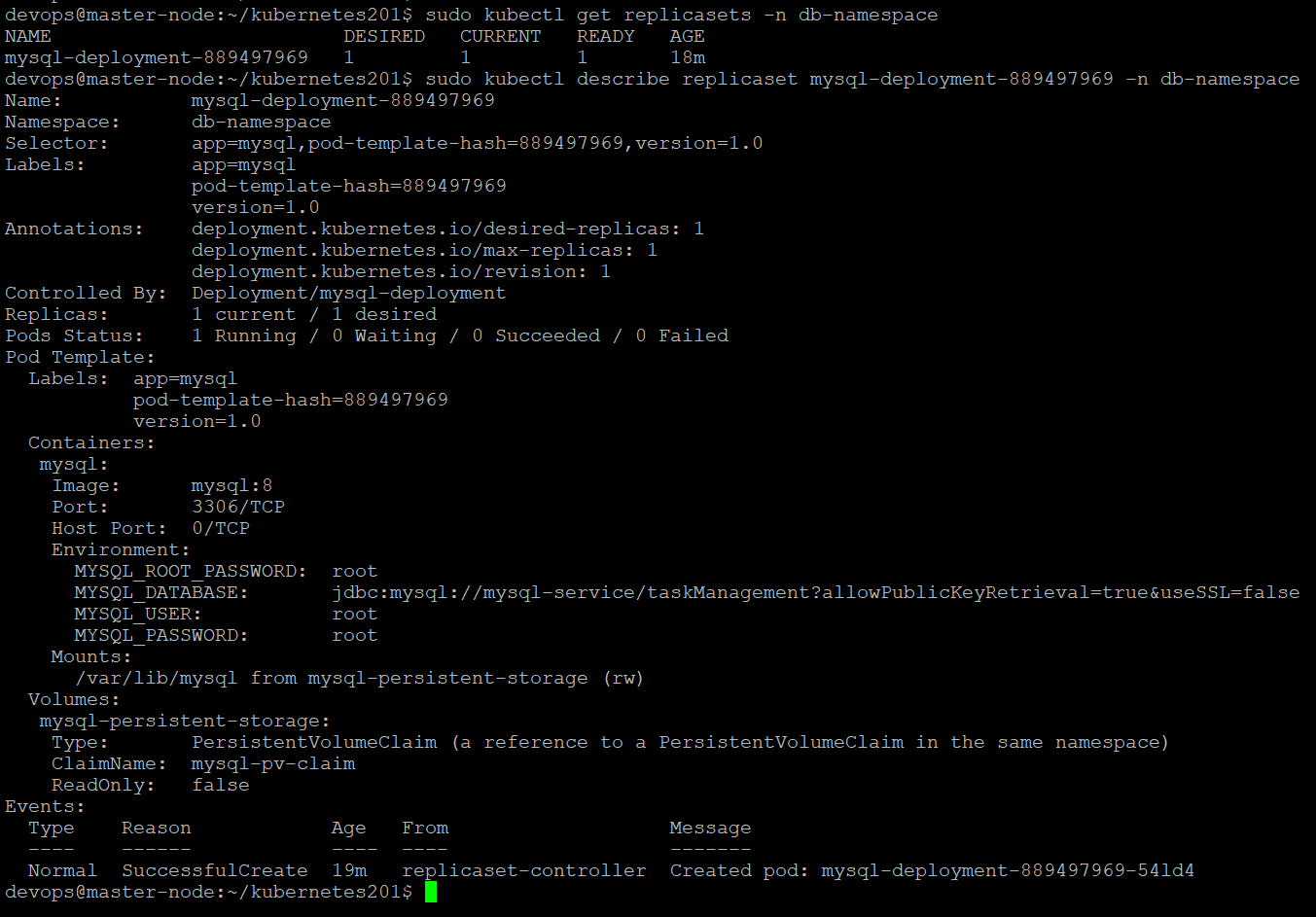
We can view information about deployment using describe command as shown below

**sudo kubectl describe deployment mysql-deployment –namespace=db-namespace**



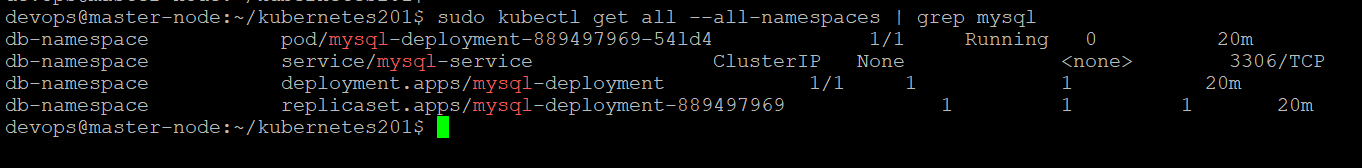
We can view information about replicaset using describe command as shown below.

**sudo kubectl describe replicaset <replicaset-name> --namespace=db-namespace**



To view all the resources with “mysql” name use below command

**sudo kubectl get all –all-namespaces | grep mysql**



Here it displays all the resources pod,deployment,service,replicaset with name “mysql”.

Now I can conclude below details for MySQL database:

* Cluster IP address of mysql service is : None
* Internla endpoints of this services is

mysql-service.db-namespace:3306 TCP

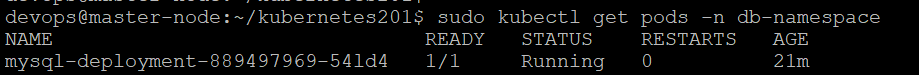
* The endpoint of pod/container is:

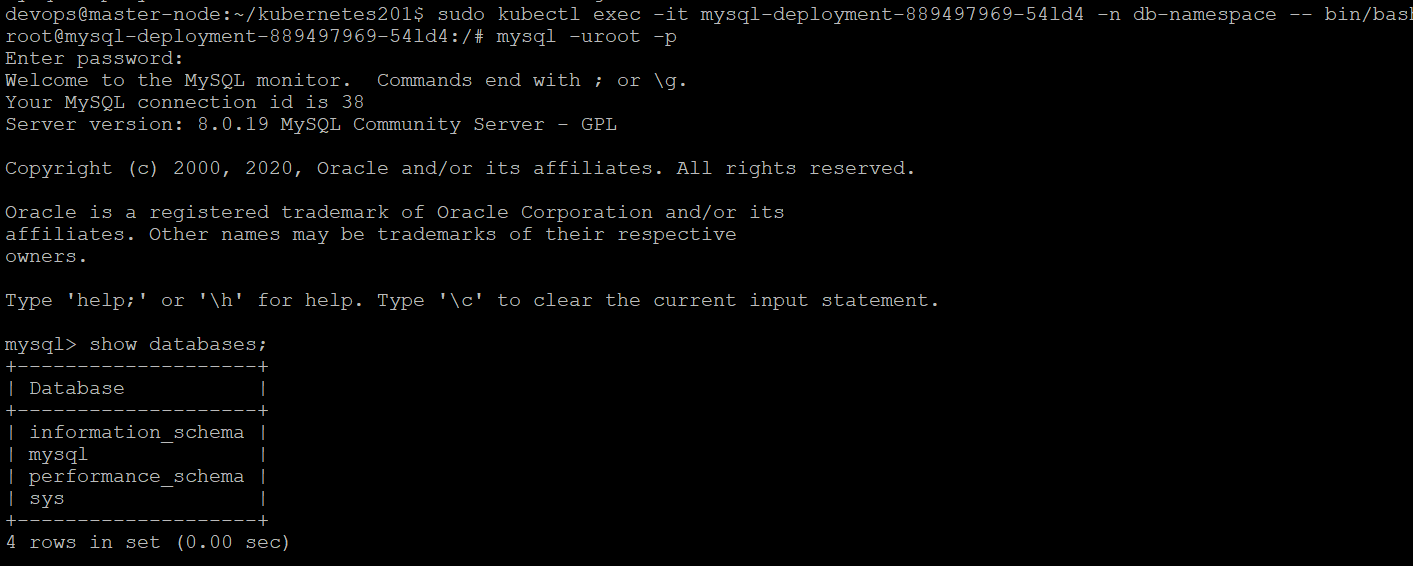
10.244.1.68:3306 TCP

1. To execute mysql commands we can enter into MySQL container which is running inside pod. To enter into the container we can use kubectl exec command.

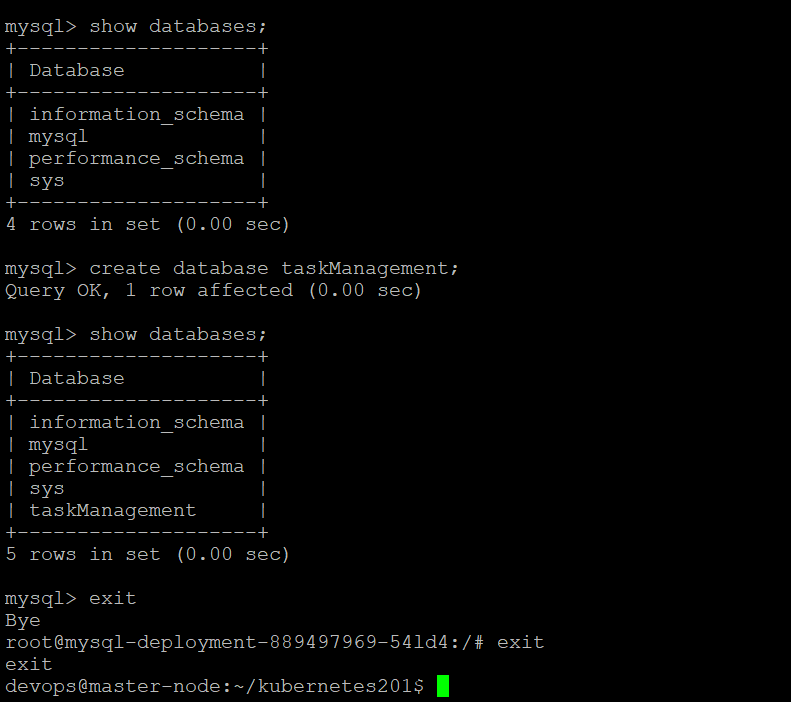
Now to open MySQL terminal, we can use exec command

**sudo kubectl exec -it <pod-name> --namespace=db-namespace -- bin/bash**





Now create a database called “taskManagement” since I am using this database name in my spring boot application.



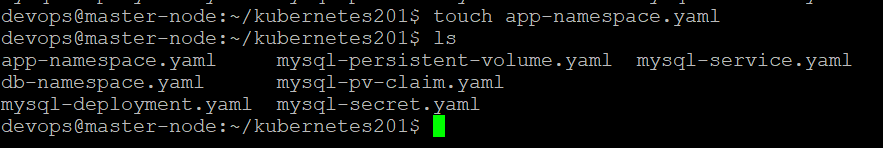
Here we entered into the MySQL container which is reside inside pod using exec command. Created taskManagement database. This database name I mentioned in the environment variables of spring boot application.

Now MySQL container is ready and running inside a pod of worker node worker01.

1. Now create namespace for spring boot application to group resources related to this application.

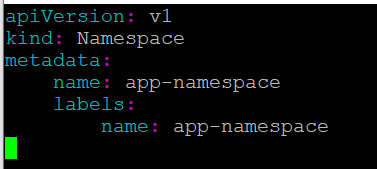
Create app-namespace.yaml file

**touch app-namespace.yaml**



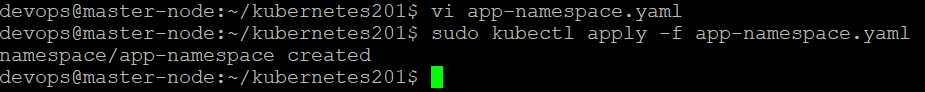
1. Edit app-namespace.yaml file as shown below.

**vi app-namespace.yaml**



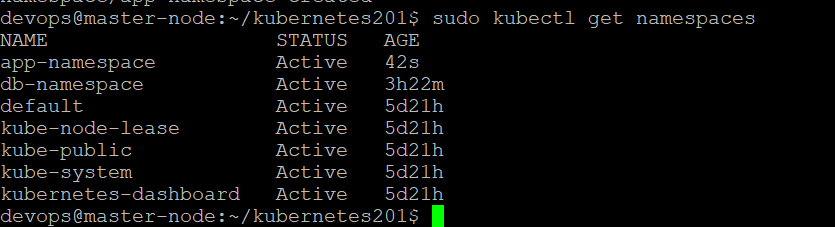
1. Now create namespace using kubectl apply command.

**sudo kubectl apply –f app-namespace.yaml**



1. Now to view all namespaces use below command

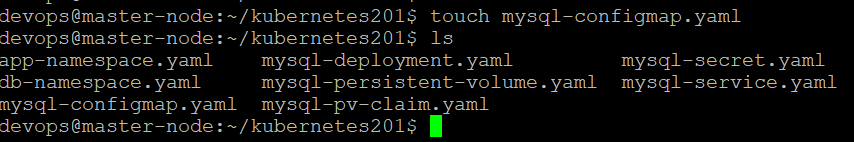
**sudo kubectl get namespaces**



1. Lets create deployment for our spring boot application i.e., task-management application. In the application deployment file, we need to specify mysql db properties in environment variable to connect to MySQL database. So inorder to access mysql-service from another namespace, we need kubernetes ConfigMap.

Lets create configMap first.

**touch mysql-configmap.yaml**



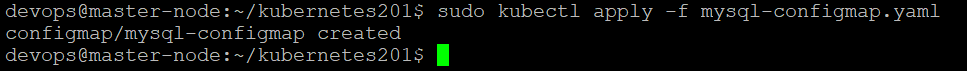
Edit mysql-configmap.yaml as shown below

**vi mysql-configmap.yaml**



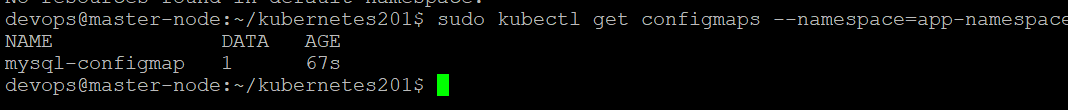
To create configmap use below command

**sudo kubectl apply –f mysql-configmap.yaml**



View configmaps using below command

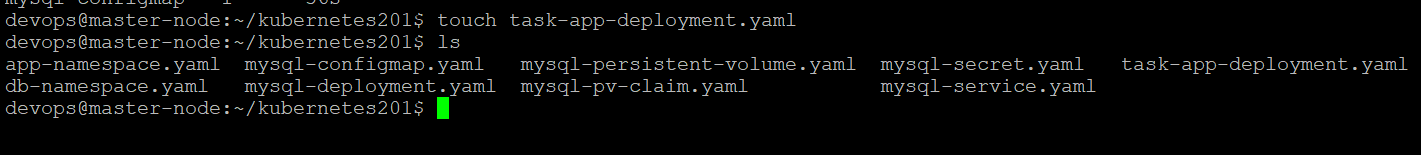
**sudo kubectl get configmaps –n app-namespace**



1. Now create deployment file for task management application

Create task-app-deployment.yaml file

**touch task-app-deployment.yaml**



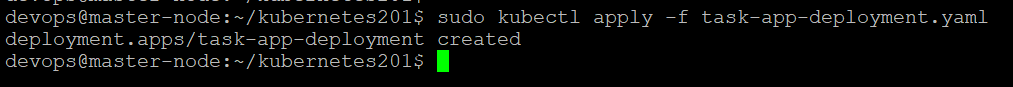
Edit task-app-deployment.yaml as shown below

**vi task-app-deployment.yaml**



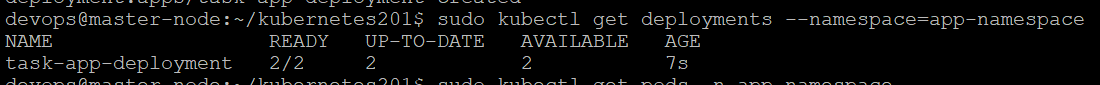
Create deployment using below command

**sudo kubectl apply –f task-app-deployment.yaml**



To view deployments in app-namespace use below command

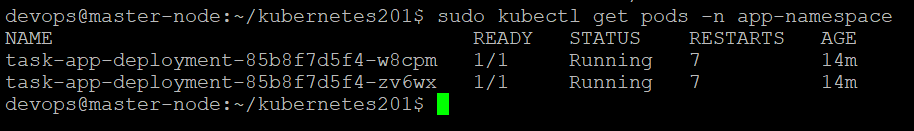
**sudo kubectl get deployments –n app-namespace**



Since I mentioned replicas count as “2”, Here for the task-app-deployment two replicasets and two pods created. Deployment is abstraction over pods. Whenver deployment is created internally pods and replicasets will be created. Lets verify pods and replicasets.

1. View task-app pods created as per task-app-deployment specification

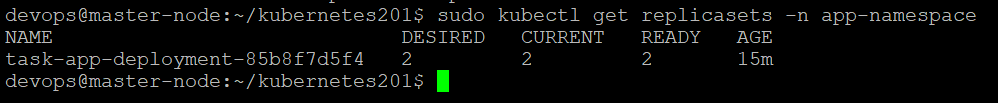
**sudo kubectl get pods –n app-namespace**



Two pods are created and running since replica count is “2”

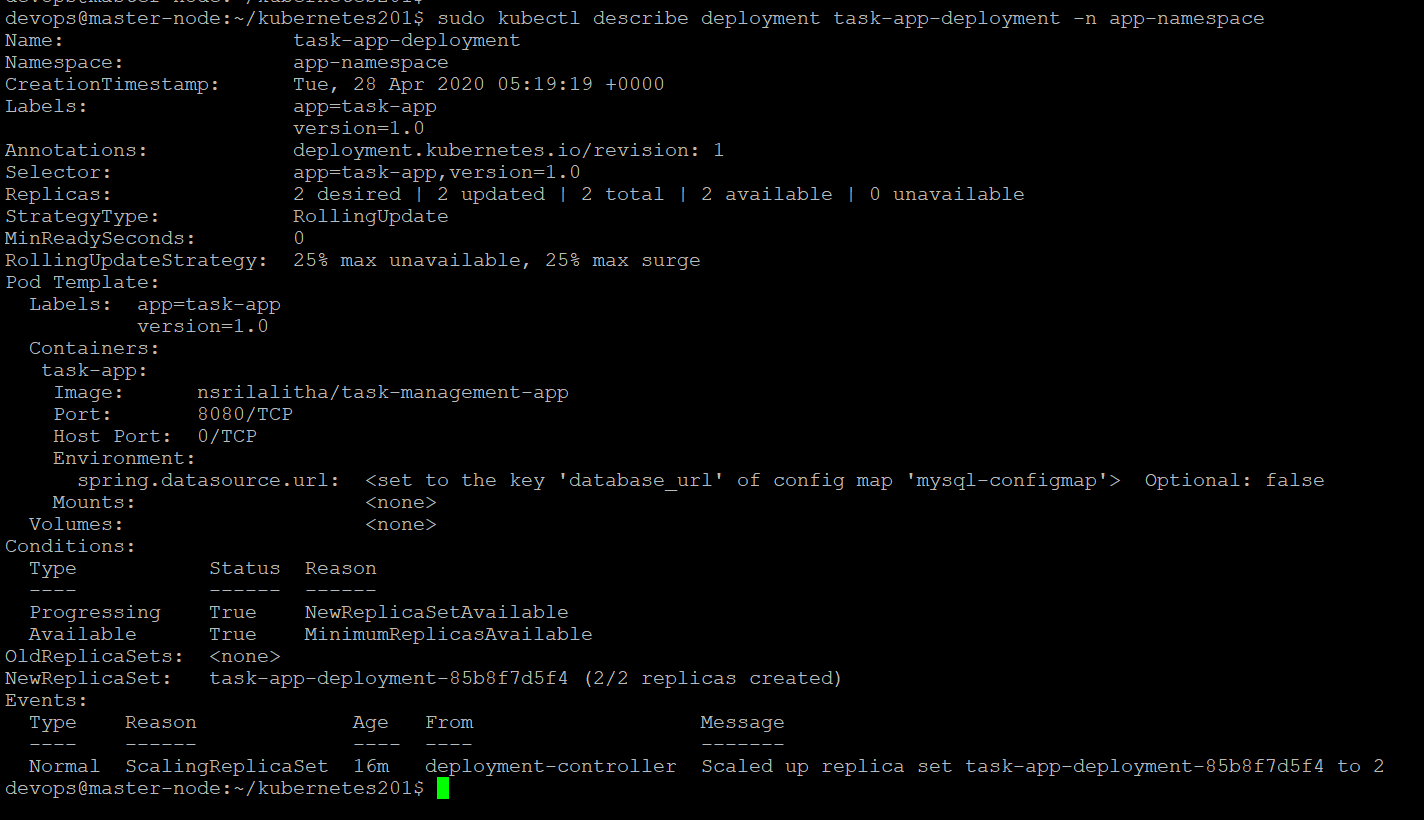
1. Now verify replicasets using below command

**sudo kubectl get replicasets –n app-namespace**



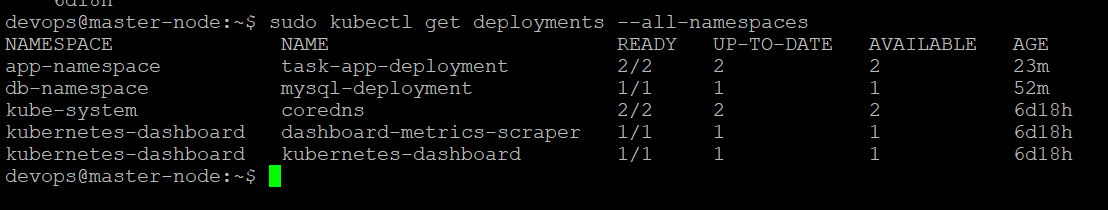
1. To get information about deployment/pod/replicaset or any resource we can use describe command. Lets get information about task-app-deployment using below command.

**sudo kubectl describe deployment task-app-deployment –n app-namespace**



Since I created deployments in two namespaces, to view all deployments avaialble in all namespaces use below command

**sudo kubectl get deployments –all-namespaces**

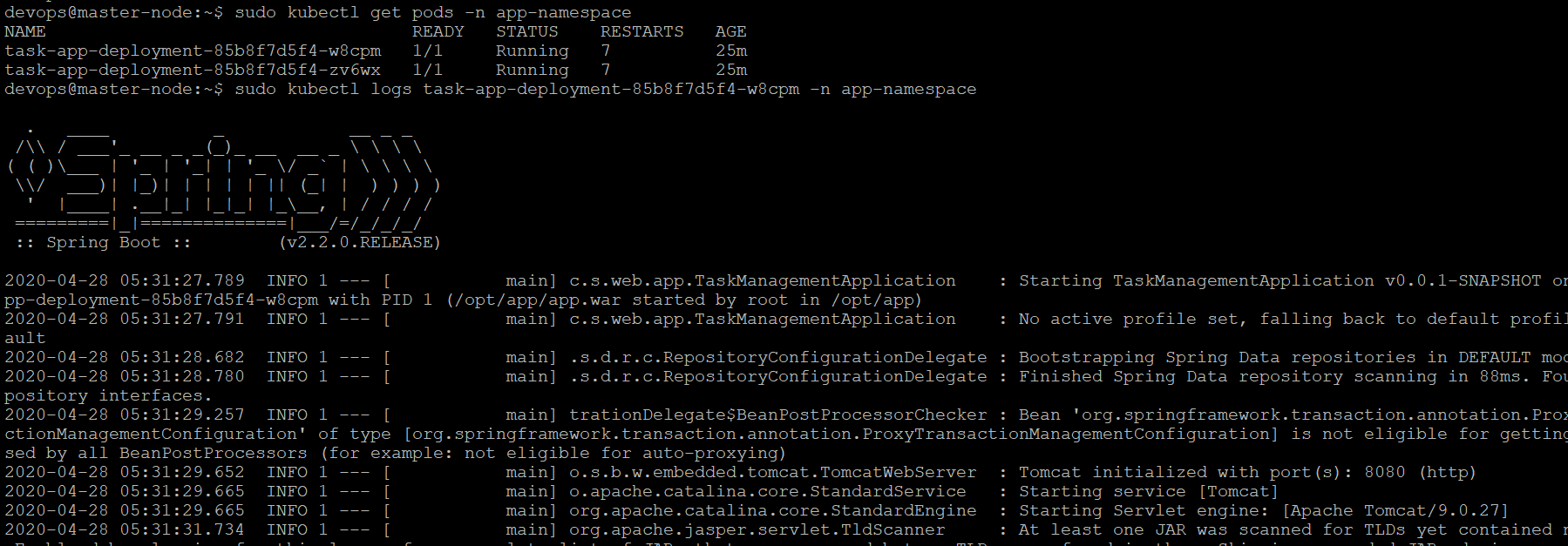


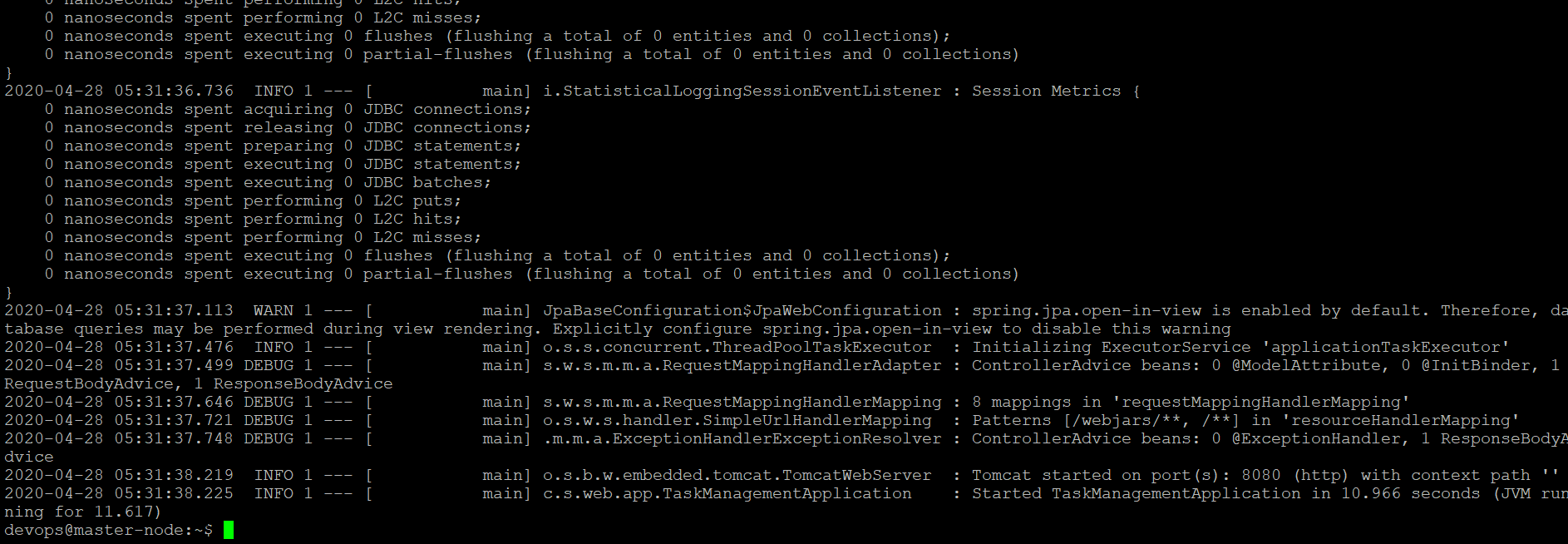
Here verify, task-app-deployment is avaialble in app-namesapce and replica count is 2 and

Mysql-deployment is avaialble in db-namespace with 1 replica.

1. Since task-app pod status as “running”, now verify task-management application is running inside pod or not by using logs command.

**sudo kubectl logs <pod-name> -n <namespace-name>**





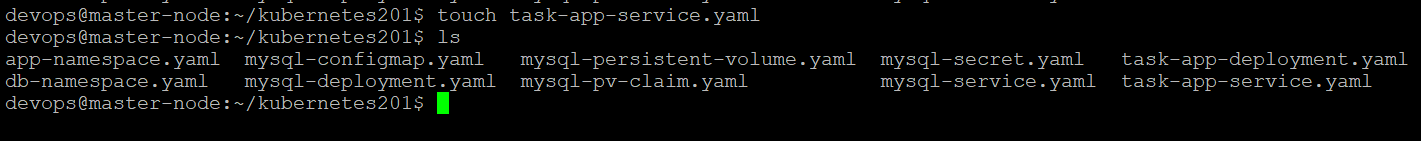
Application is successfully started at port 8080.

So inorder to access the application which is running inside a pod, we need service. So we need service for task-management application.

1. Create service for task management application

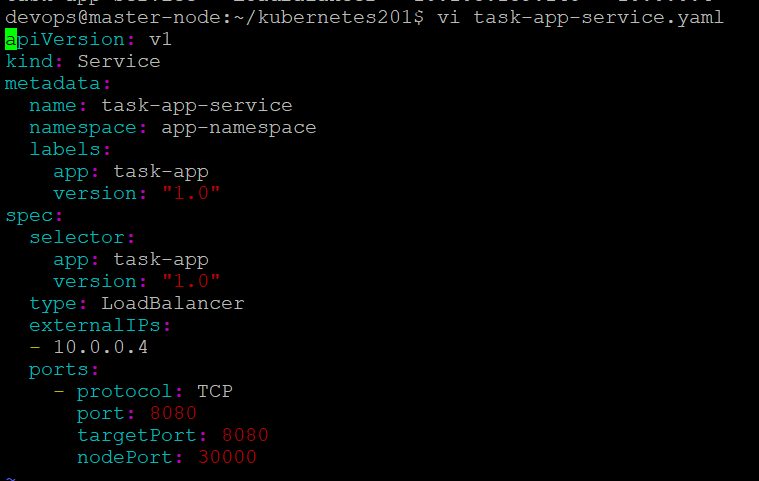
Lets create task-app-service.yaml file

**touch task-app-service.yaml**



Edit task-app-service.yaml as shown below

**vi task-app-service.yaml**



Here service is going to be created as external service since we have to access this application from browser not through internal endpoint/ip address. So in the service configuration, mentioned type as “LoadBalanacer”. We need external ip address to accept external requests. The range of external ip port is between 30000 to 32767. “nodePort” in configuration specifies external ip port. Here I gave external ip port as 30000. And “targetPort” should match with “containerPort” in deployment configuration. So giving targetPort as 8080.

Here external Ip ’10.0.0.4’ is the IP address of host machine which we can get using ifconfig command.

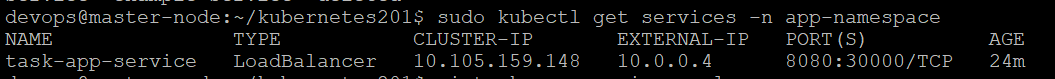
Now create task-app-service using apply command as shown below

**sudo kubectl apply –f task-app-service.yaml**



Now verify services available in app-namespace using below command

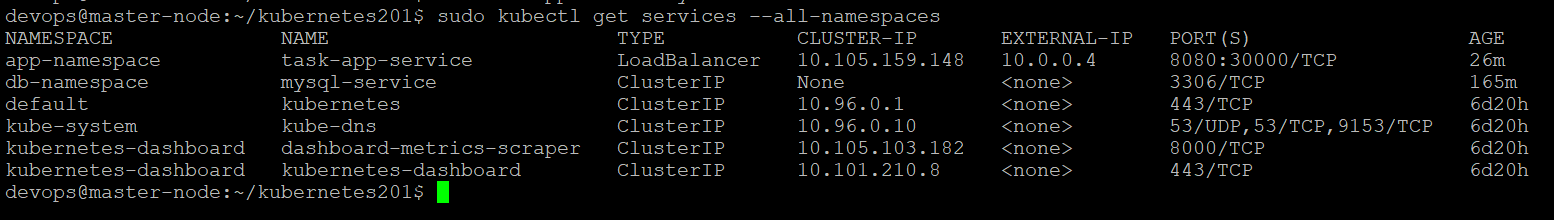
**sudo kubectl get services –n app-namespace**



Here Type is “LoadBalancer” and Ports “8080:30000/TCP” that means, 8080 is internal ip port and 30000 is external Ip port.

Verify all the services in all namespaces using below command

**sudo kubectl get services –all-namespaces**



1. Now access the application which is running using kubernetes

Now I can access the application using curl command using external ip with external port or Cluster IP with internal port address.

**Curl http://<external-ip-address>:<external-port>**

I can access the application in kubernetes cluster using below urls

<http://10.0.0.4:3000> -> external ip with external port

<http://10.105.159.148:8080> -> Cluster IP with internal port

Here my application has following urls:

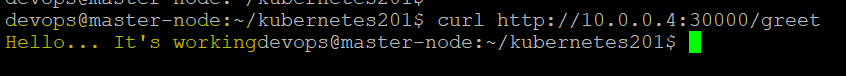
<http://10.0.0.4:30000/greet> which prints “Hello...its working message”

<http://10.0.0.4:30000/home> which gives home screen where task name and description need to be submitted.

<http://10.0.0.4:30000/tasks> which displays previously added tasks details

Now will hit greet url using below command

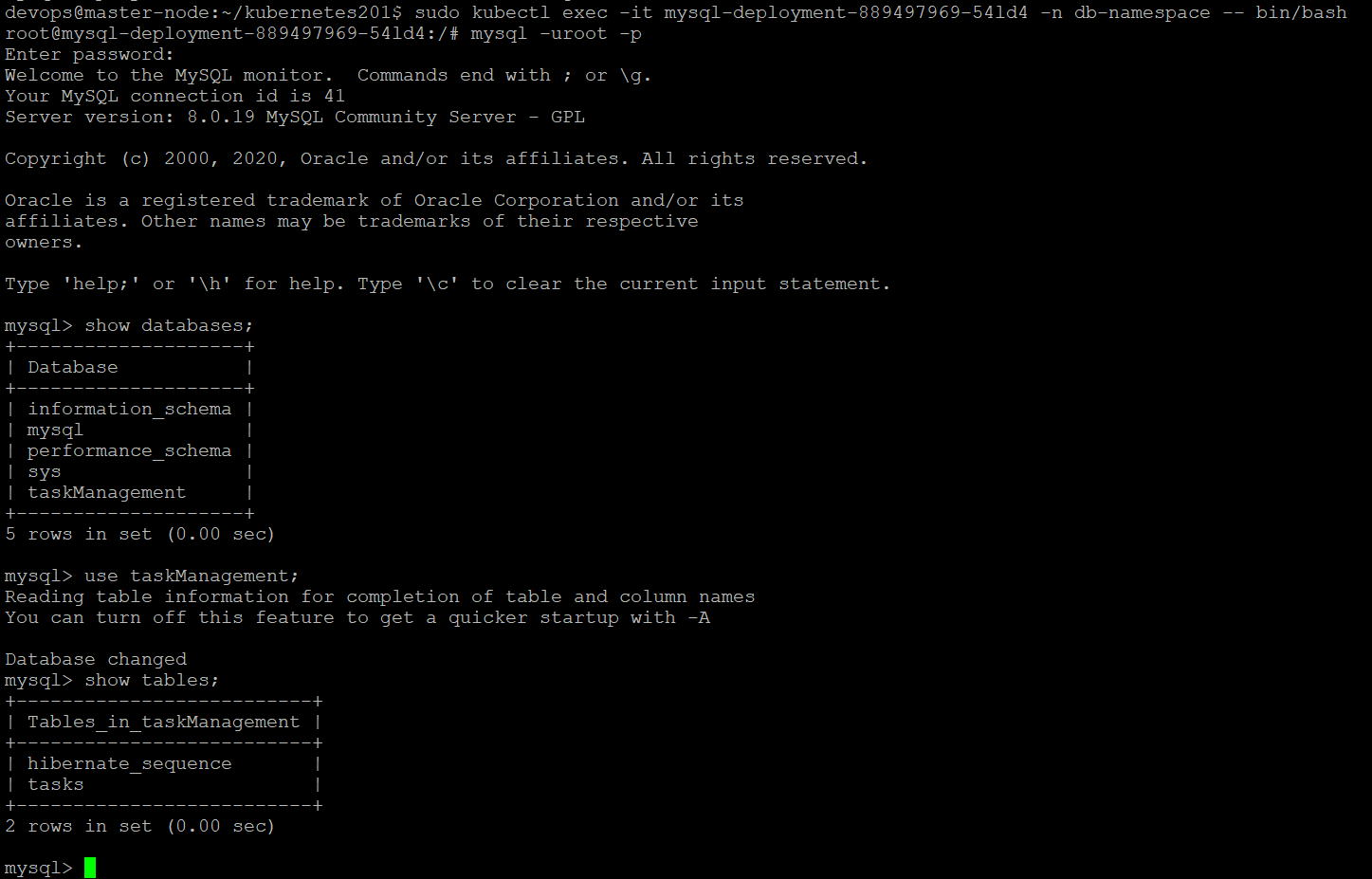
**curl** [**http://10.0.0.4:30000/greet**](http://10.0.0.4:30000/greet) (GET request)



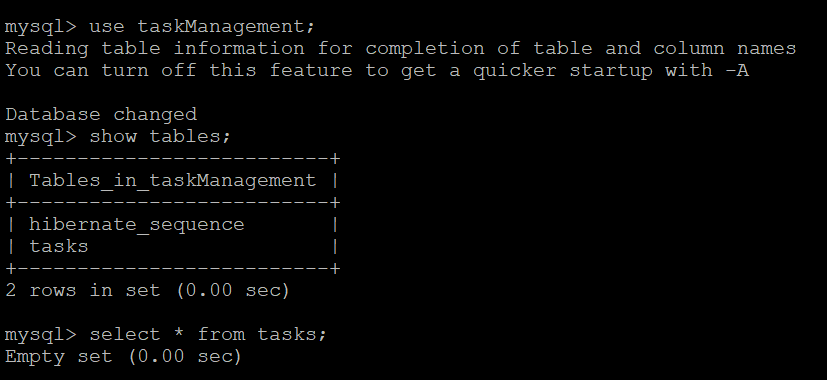
Whenver application is started, inside taskManagement database, tables will be created by spring data JPA. Lets verify tables in taskManagement database which is availabale in mysql-deployment pod. Lets get into mysql pod using exec command and verify tables.

Now will verify db tables in mysql pod using exec command

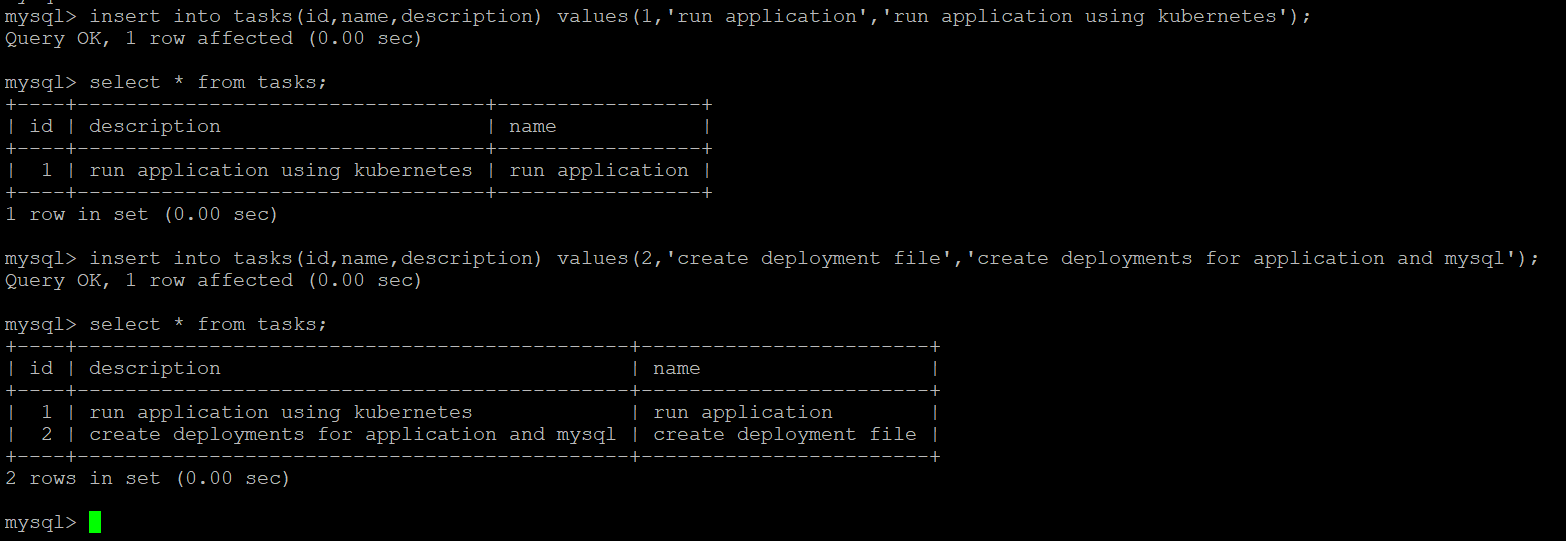
**Sudo kubectl exec –it <mysql-pod-name> -n db-namespace – bin/bash**



Since application is started, tasks table is created. It is empty since as of now I didn’t add any tasks. Lets add one task in db.



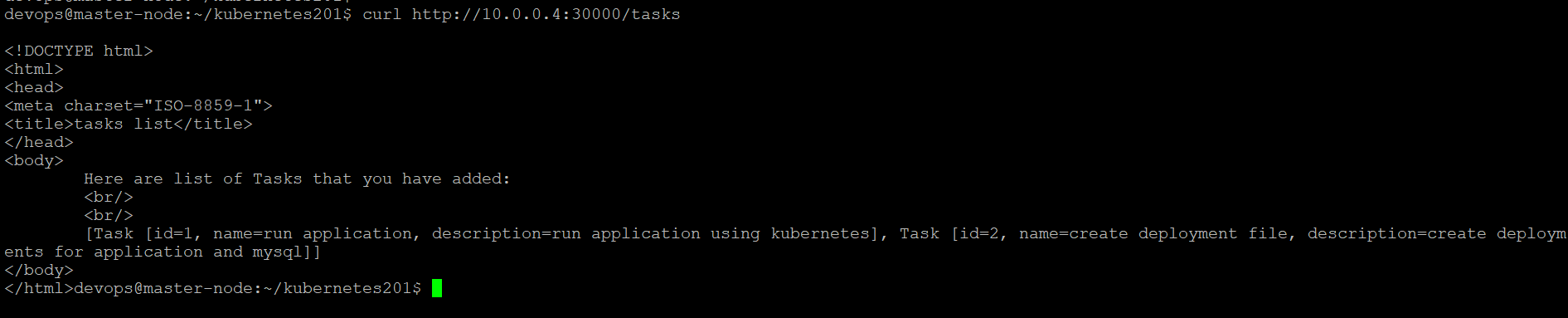
Lets add few tasks here, which we can view them using <http://10.0.0.4:30000/tasks> url.



Now will hit http://10.0.0.4:30000/tasks url (GET request) using below command

**curl** [**http://10.0.0.4:30000/tasks**](http://10.0.0.4:30000/tasks)

The tasks added within mysql pod will reflected here. Since output of this /tasks url is html page, we get html tags and inside html body tasks will be displayed.

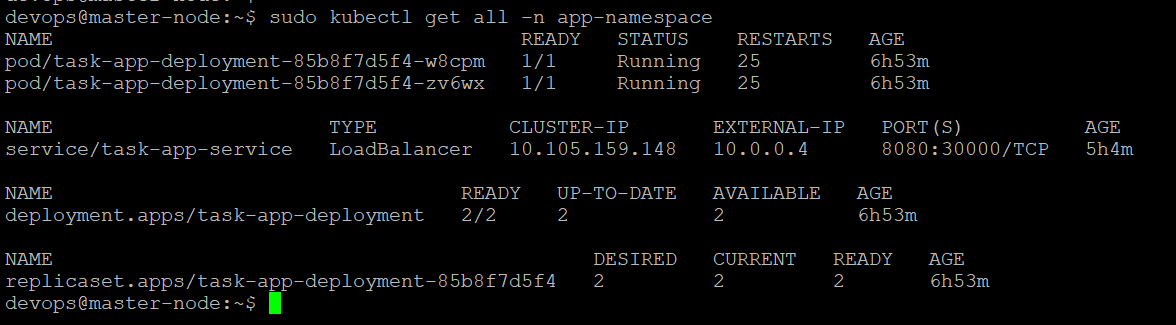


Here I am successfully able to access my application which is running using kubernetes. I am able fetch tasks details that are added into tasks table to MySQL container which is running inside another pod. The application pod is communicating with MySQL pod through mysql service. And the application is accessible through application service i.e., task-app-service with external ip and cluster ip as well.

* Bring in Auto Healing & Auto Scaling capabilities to the app.

Lets verify deployments, services, pods, replicasets available for this application using below command

**sudo kubectl get all –n app-namespace**



**Scaling up:**

Suppose if I need 5 replicas, I have to mention replica count as “5” in deployment file. Based on deployment, pods, replicasets will be created.

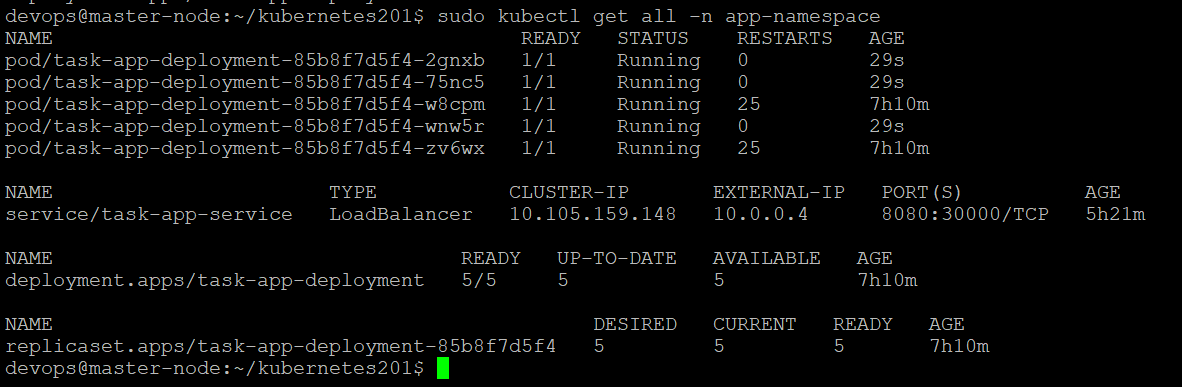
Lets scale up the application, using kubectl scale command

**sudo kubectl scale –replicas=5 deployment/task-app-deployment**



Now verify, Pods and replicasets should be 5

**sudo kubectl get all –n app-namespace**

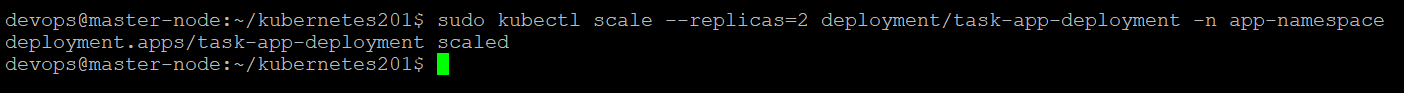


My desired and current replica sets showing as “5”. And 5 pods are running as expected.

**Scaling down:**

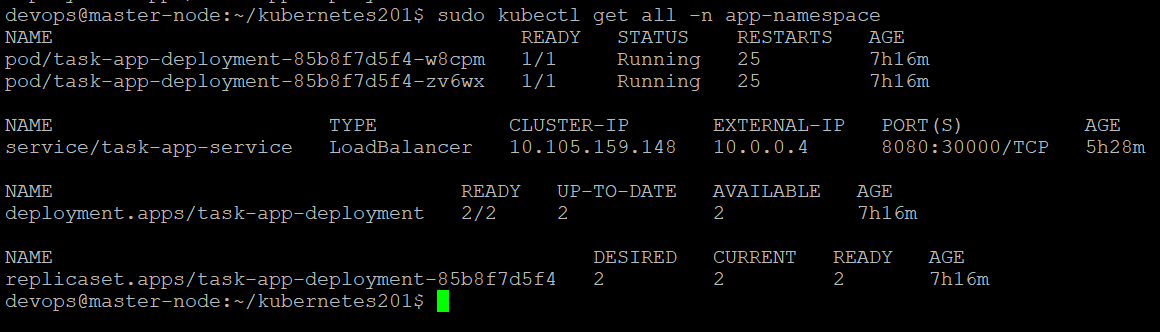
Suppose If I want to reduce replicas and I want to go back to original state i.e., two replicas. Then I can run the kubectl scale command

**sudo kubectl scale –replicas=2 deployment/task-app-deployment**



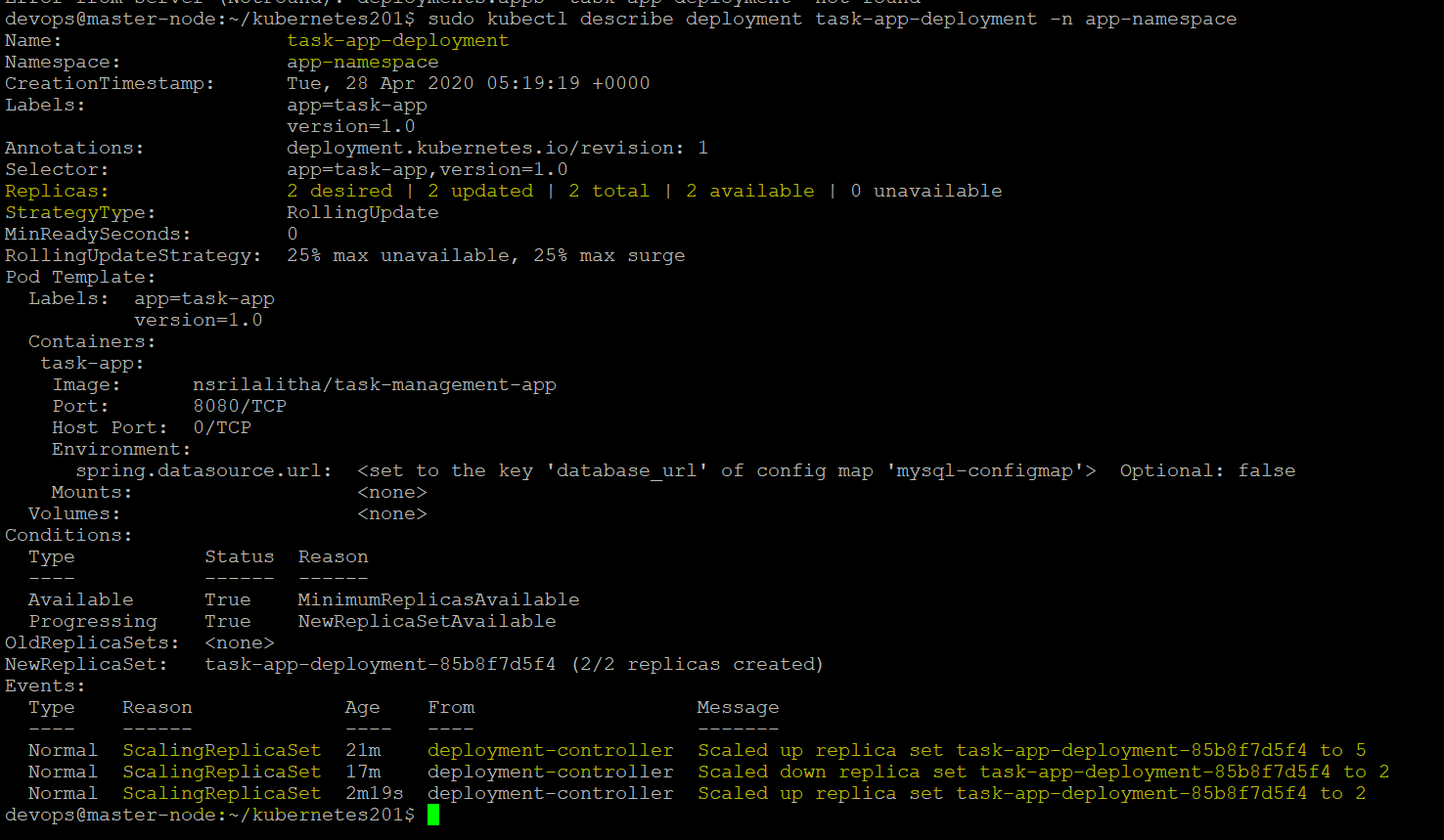
Now verify, it should display 2 pods and 2 replica sets

**sudo kubectl get all –n app-namespace**



We can view scaling up and down information using kubectl describe deployment command. Deployment-controller will monitor scaling up and down.

**sudo kubectl describe deployment task-app-deployment –n app-namespace**



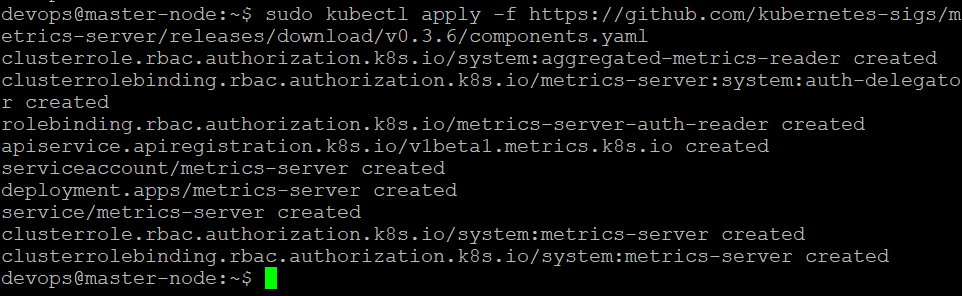
**Auto Scaling:**

**Horizontal Pod Autoscaler (HPA)** automatically scales the number of pods in a replication controller, deployment, replica set or stateful set based on observed CPU utilization, memory or other metrics.

**metrics-server** monitoring needs to be deployed in the cluster to provide metrics via the resource metrics API, as HPA uses this API to collect metrics.

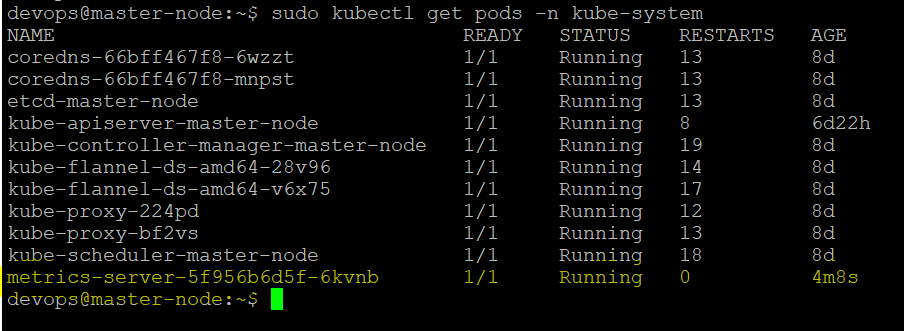
So first install metrics-server. It can be installable with below command

**sudo kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.6/components.yaml**



This will install metrics-server in our K8s cluster. Verify with below command

**sudo kubectl get pods –n kube-system**

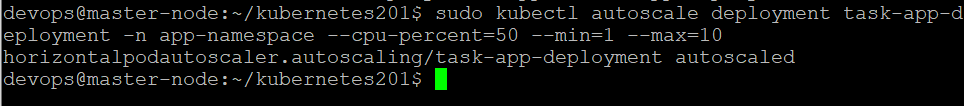


We can create Horizontal Pod Autoscaler either declaratively using kubectl autoscale command or imperatively using hpa configuration file.

**Delcarative approach for autoscaling:**

Now I am going to implement autoscaling for task management application decalaratively using kubectl autoscale command.

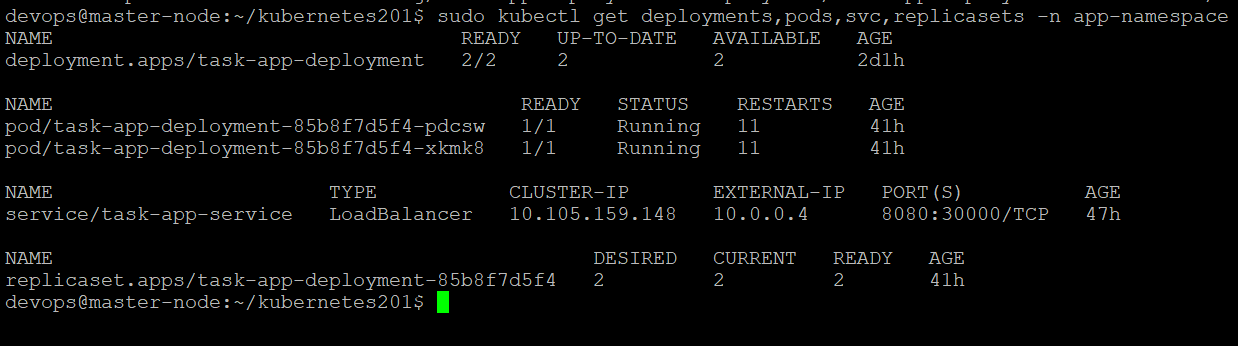
**sudo kubectl autoscale deployment <deployment-name> --cpu-percent=50 –min=1 –max=10**



The above command will create a Horizontal Pod Autoscaler that maintains between 1 and 10 replicas of the Pods controlled by the task-app deployment we created in the earlier sections. It will increase and decrease the number of replicas (via the deployment) to maintain an average CPU utilization across all Pods of 50% i.e., when CPU utilization is more than 50% it autoscales pods. At max it can reach 10 pods serving the request.

Now verify how many pods running

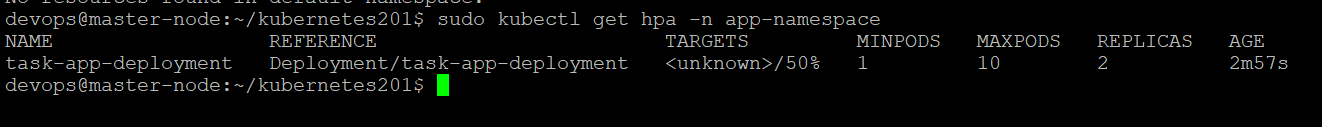
**sudo kubectl get deployments,pods,svc,replicasets –n app-namespace**



As per task-app-deployment two pods are running. Two replicasets are available as per desired state.

check autoscaler status using below command

**sudo kubectl get hpa –n app-namespace**



It is showing min pods 1 and max pods 10. It will autoscale as per CPU utilization.

Please note that the current CPU consumption is UNKNOWN as we are not sending any requests to the server (the TARGET column shows the average across all the pods controlled by the corresponding deployment).

**Increase load:**

Now will increase load on task-app pod using busybox. Autoscaler responds to increased load.

I will run below command to increase load on task-app. This command acts as a load generator.

**sudo kubectl run --generator=run-pod/v1 -i --tty load-generator --image=busybox /bin/sh**

will run shell script command to generate load on task-app pod by hitting application

**while true; do wget -q -O- http://10.0.0.4:30000/greet; done**



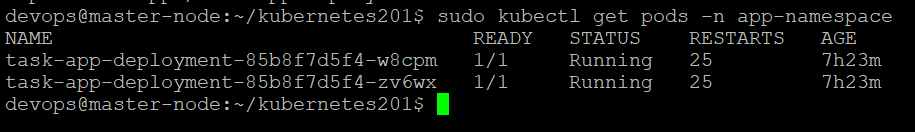
**AutoHealing:**

For this task-app-deployment now replicas are 2 and pods are 2 as per configuration. So this is my desired state. Suppose if I delete one pod by mistake, kubernetes will check cluster status and it tries to match current state with expected desired state. It automatically creates another pod which is running at another ip address.

Lets verify auto healing scenario.

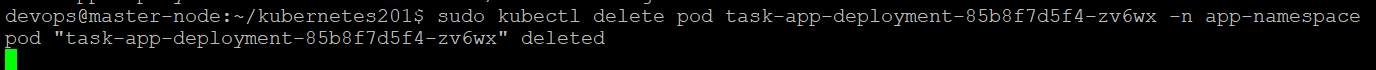
Lets get pods using kubectl get pods command

**sudo kubectl get pods –n app-namespace**



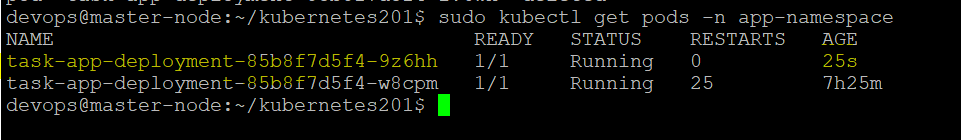
Now I will delete one pod using kubectl delete command

**sudo kubectl delete pod task-app-deployment-85b8f7d5f4-zv6wx -n app-namespace**



Now verify pods using kubectl get command

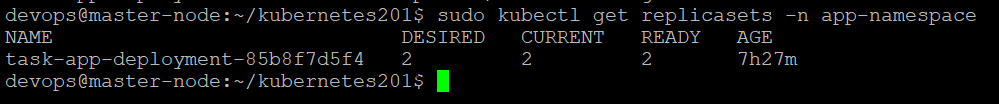
**sudo kubectl get pods –n app-namespace**



Here another new pod is created and running as per desired state.

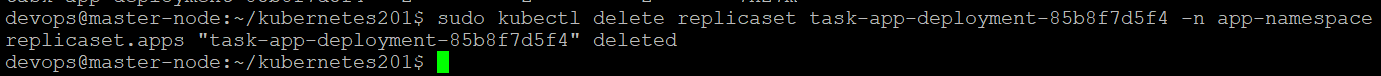
Now lets verify replica sets

**sudo kubectl get replicasets –n app-namespace**

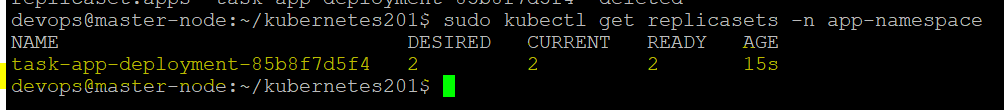


Now lets delete this replicaset

**sudo kubectl delete replicaset task-app-deployment-85b8f7d5f4 -n app-namespace**



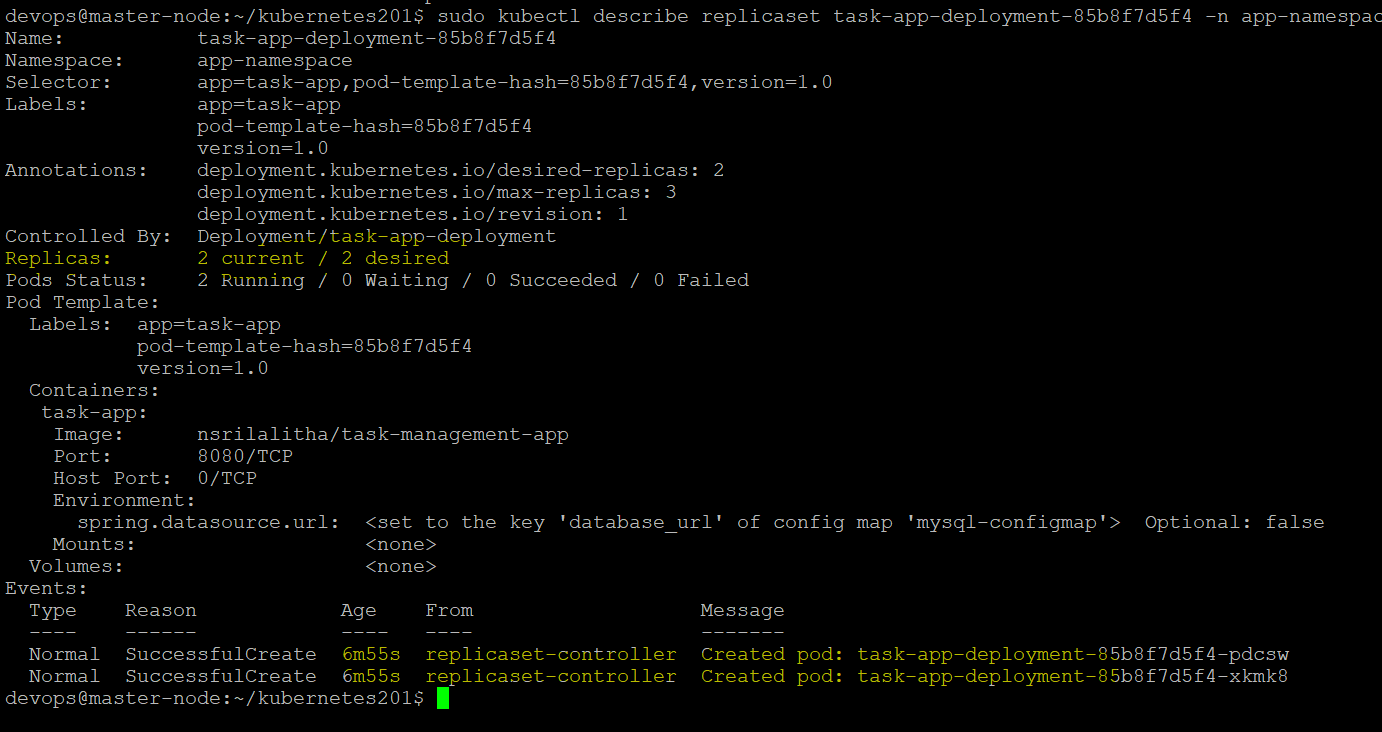
Now verify replicasets using kubectl get command



New Replicasets are automatically created by Kubernetes as per desired state. The age of replicaset is showing as 15sec.

We can view information about replicaset using describe command

**sudo kubectl describe replicaset <replicaset-name> -n app-namespace**



New pods created by kubernetes as per desired state.

