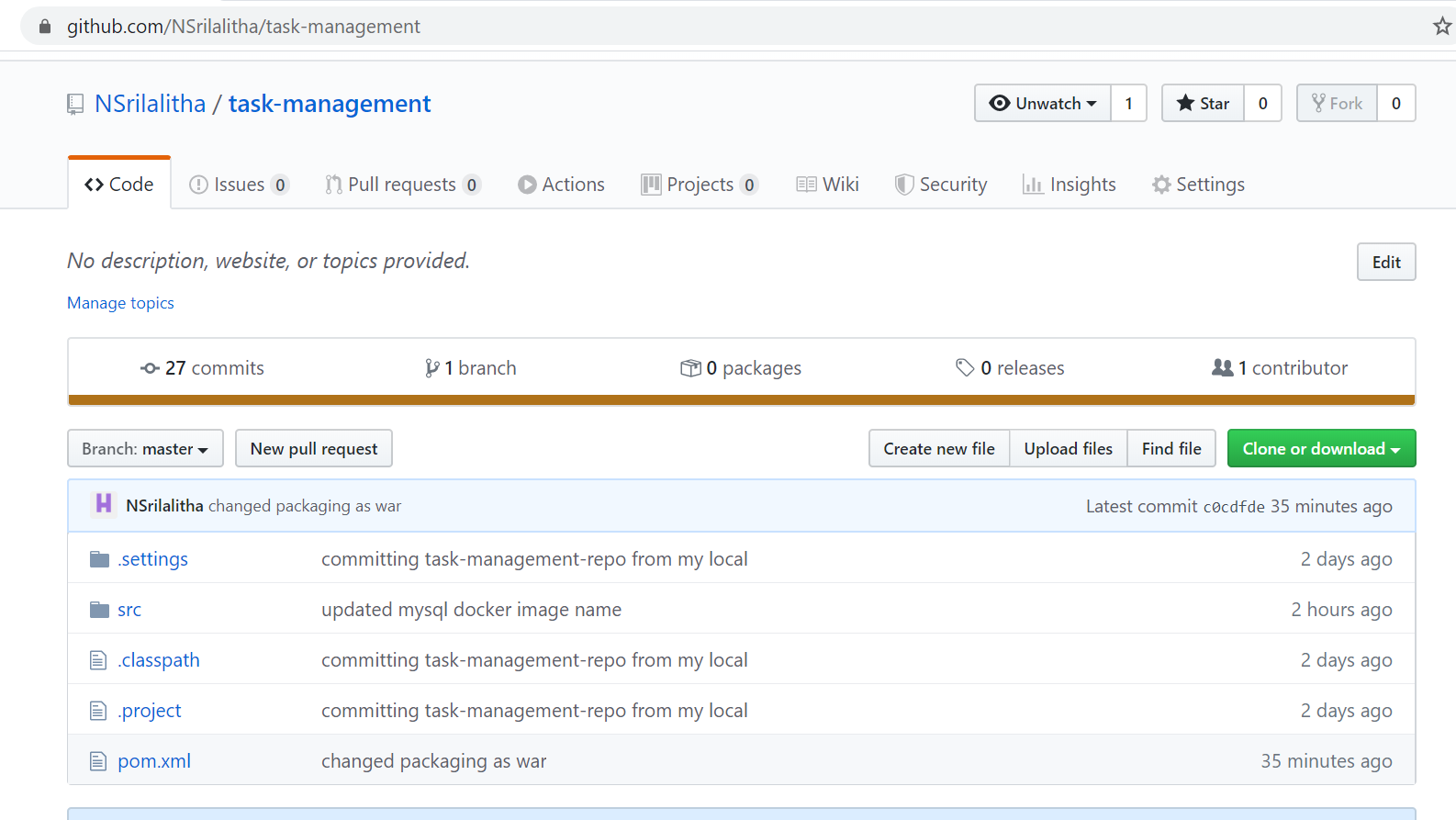
**Managing Docker images - Case Study**

* Write a simple Java / Node application or download a sample from Github.

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>

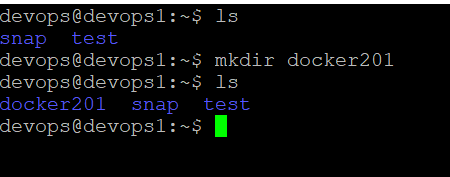


In order to dockerize this application, we need java 8, maven and git dependencies. We install these dependencies and make application to run in any environment using Dockerfile.

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

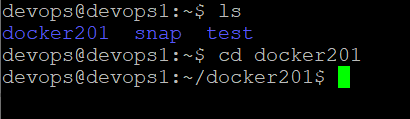
1. Now create a folder called docker201 to work on docker.

**mkdir docker201**



1. Move to docker201 folder

**cd docker201**

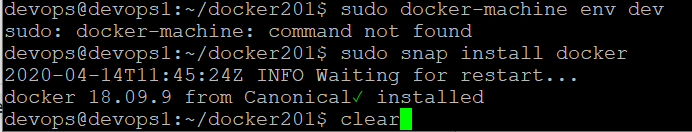


1. Install docker machine with below commands

**sudo apt update**

**sudo apt install snapd**

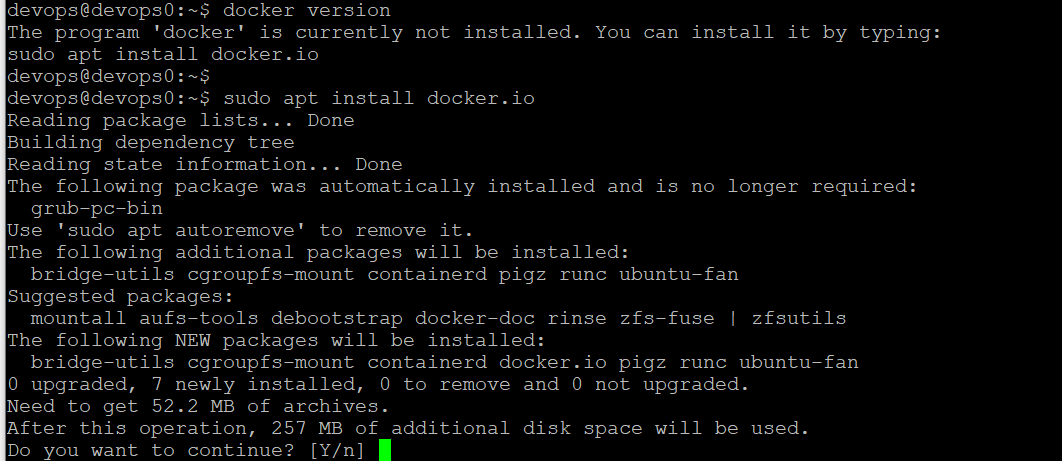
**sudo snap install docker**



(Or)

Install docker using below command

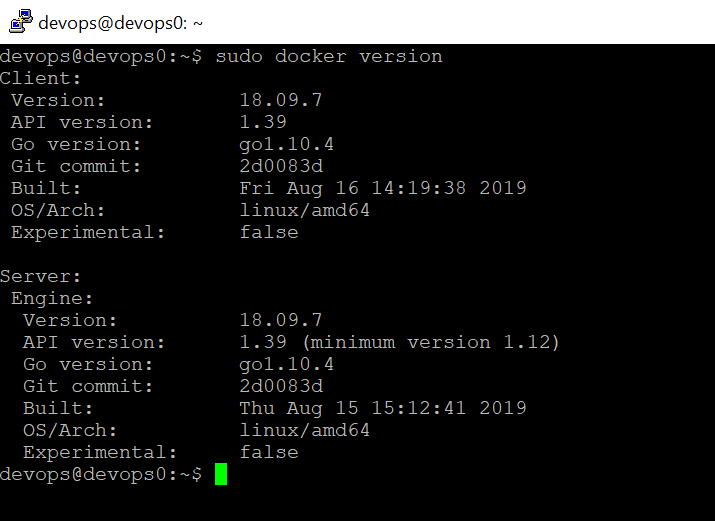
**sudo apt install docker.io**



Now docker is successfully installed. Now verify docker version

1. Verify docker version using below command

**sudo docker version**

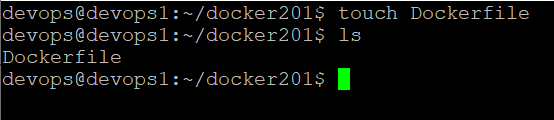


1. Now create Dockerfile.

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.

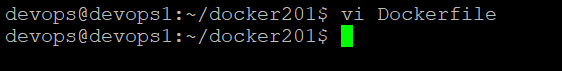
Use below command to create Dockerfile

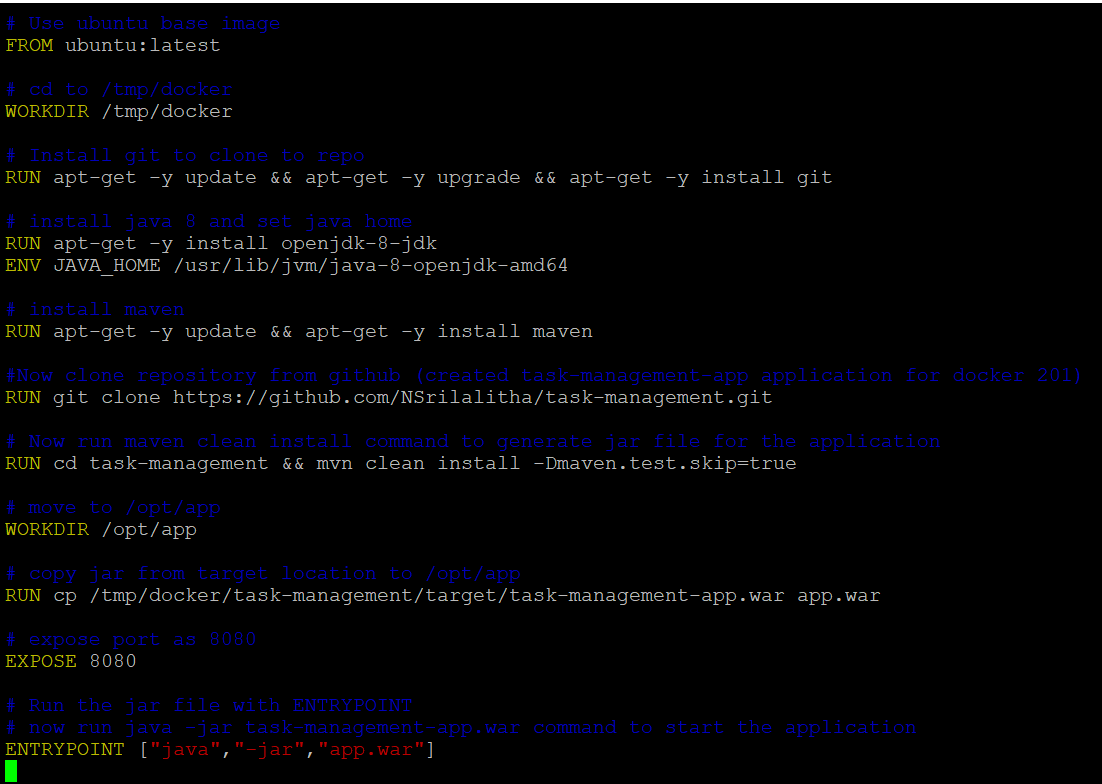
**touch Dockerfile**



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

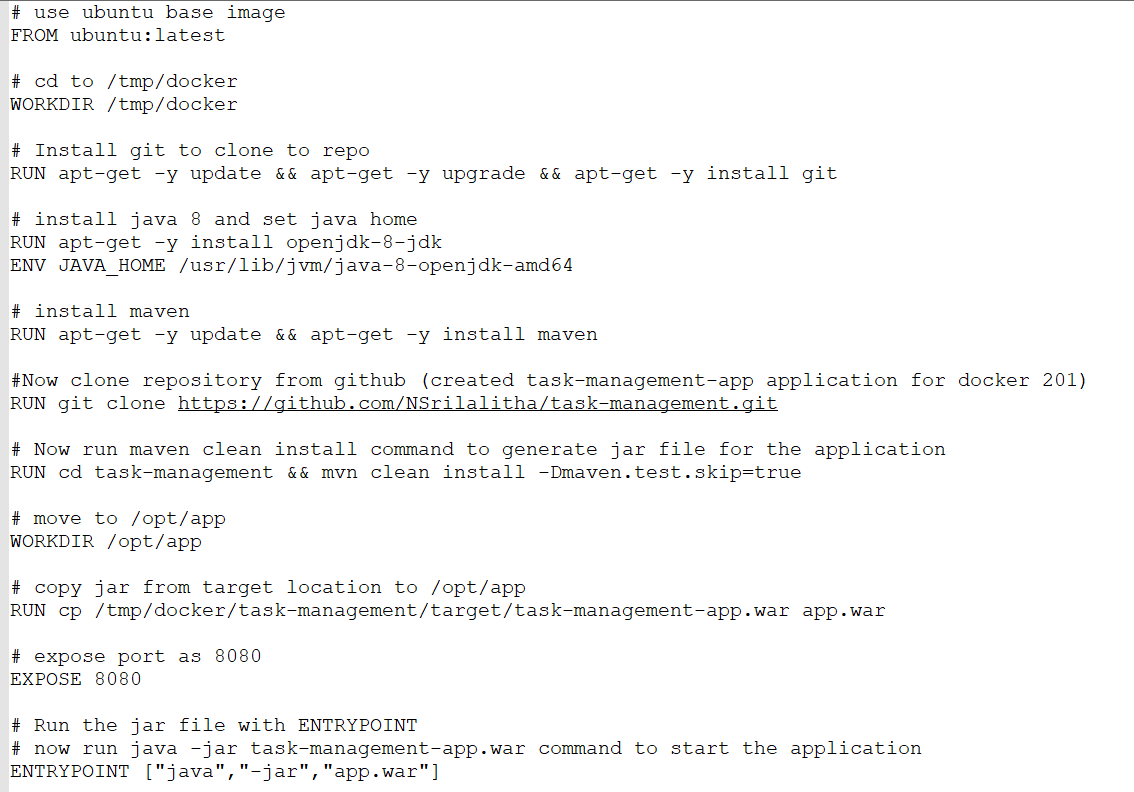
**vi Dockerfile**







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.



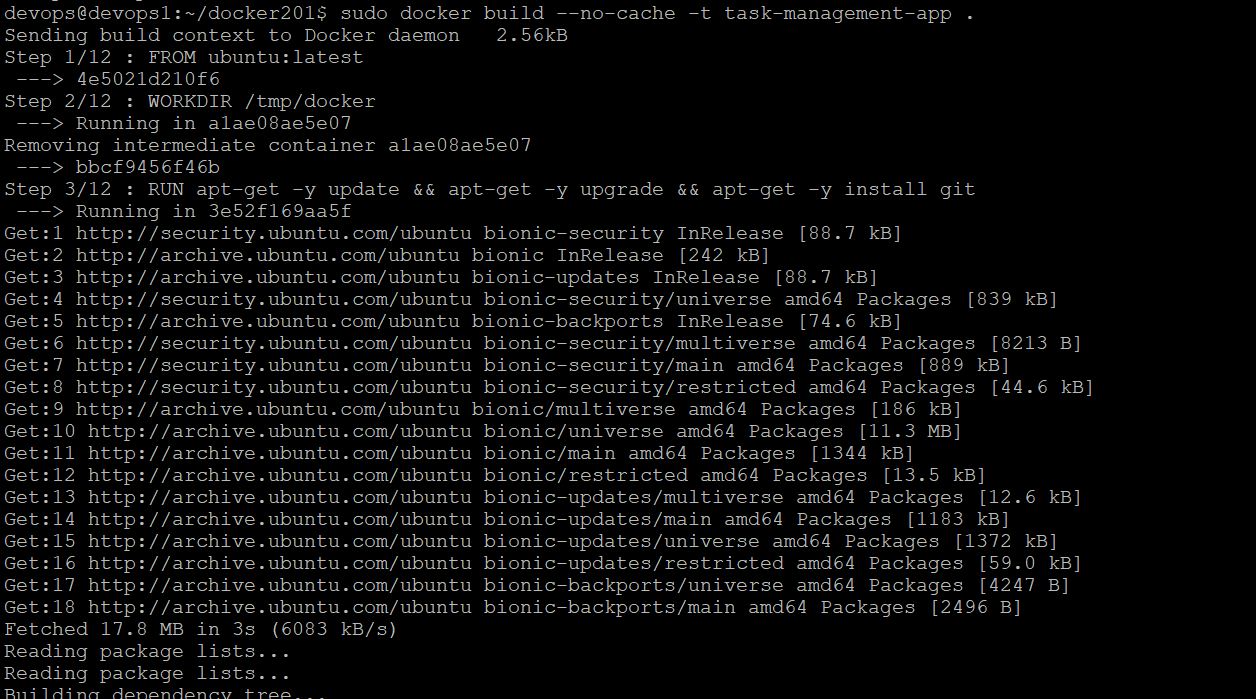
Let’s get into the docker file and understand each and every piece of it.

* **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.

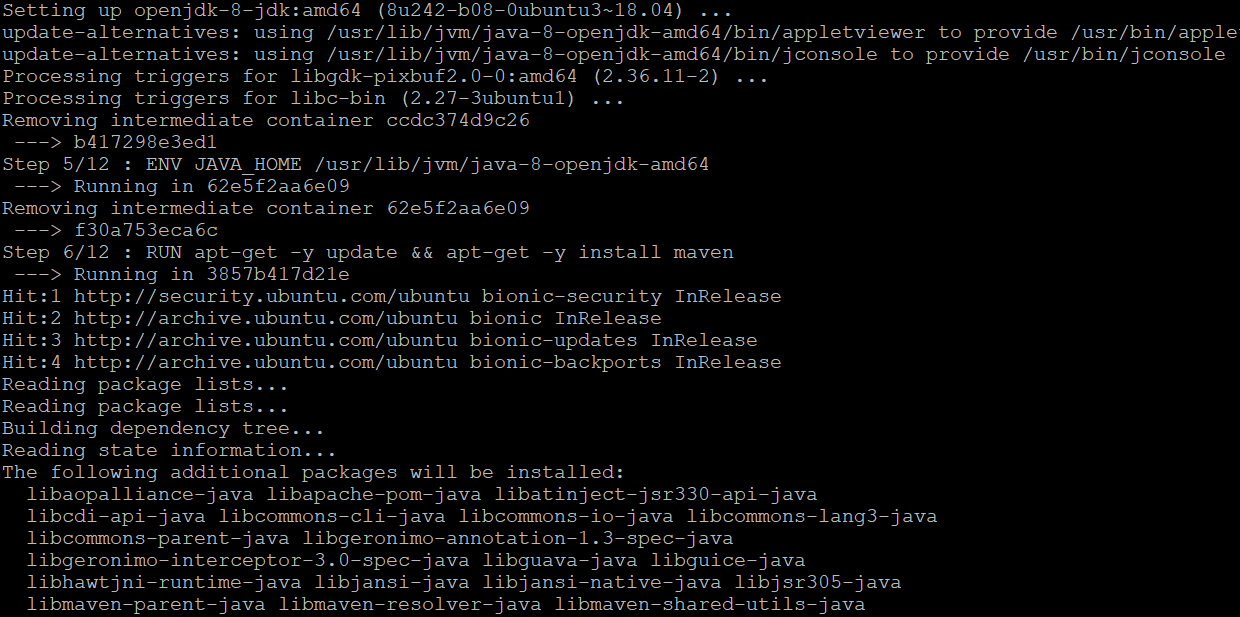
1. Now build docker image using below command

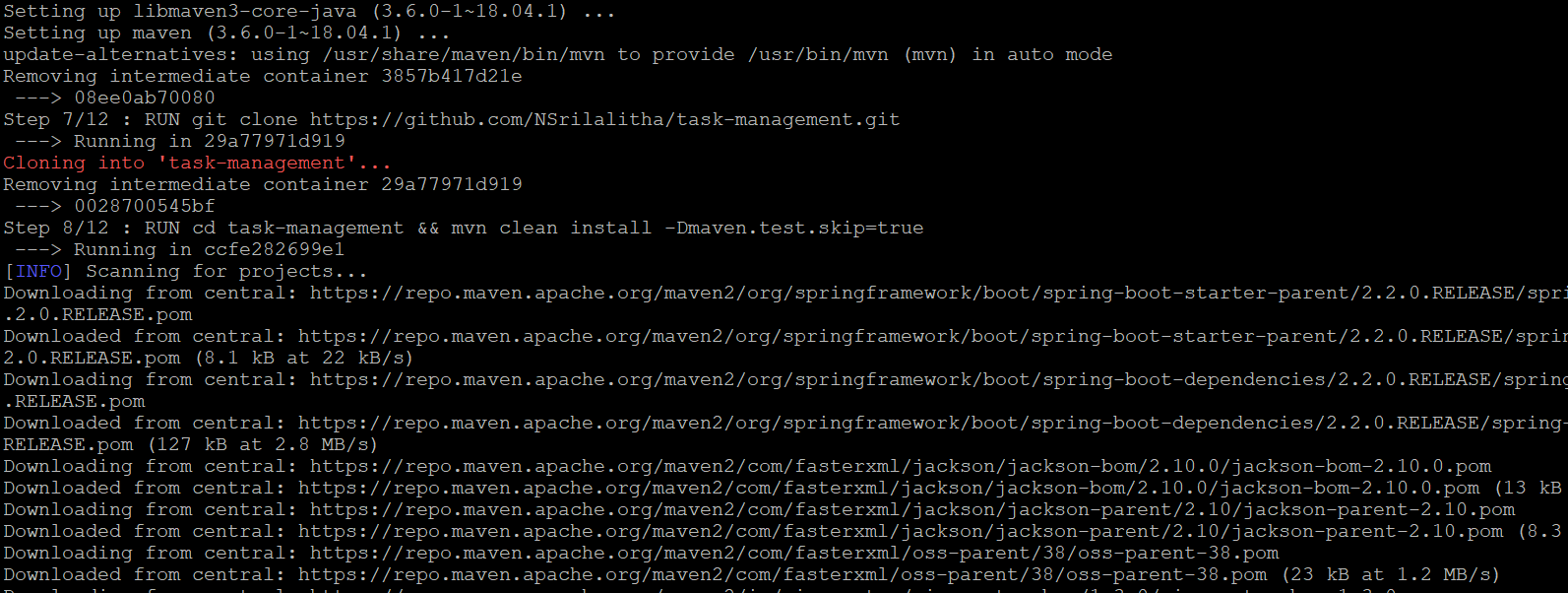
**sudo docker build –t task-management-app .**

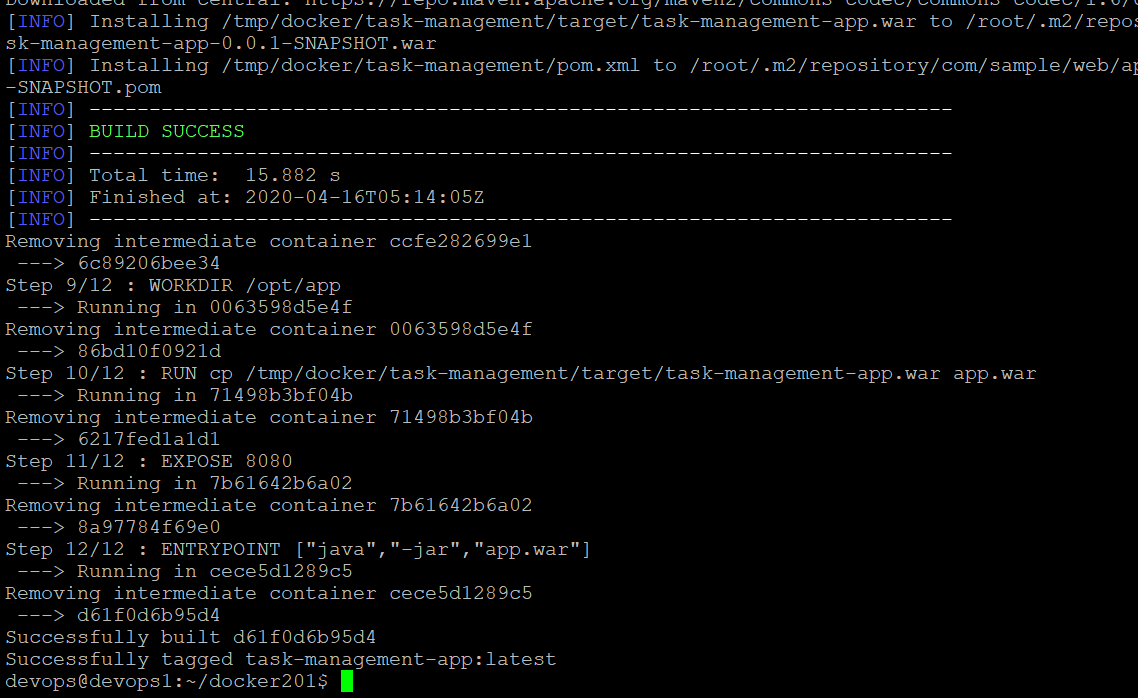
Here “task-management-app” is docker image name given for this application.





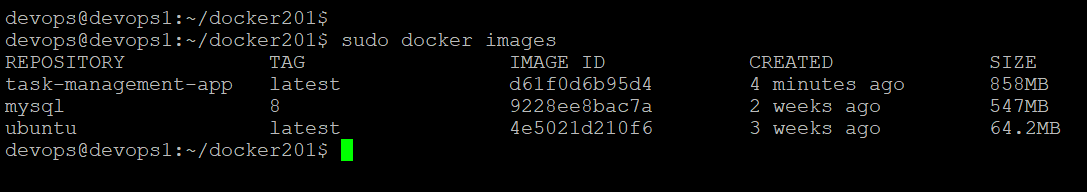






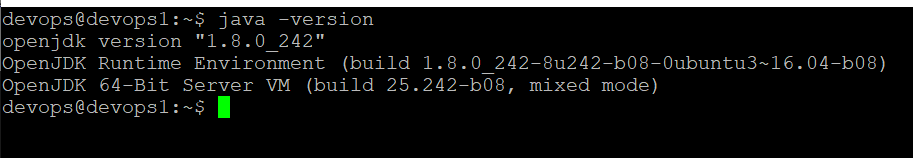
1. Now docker image is successfully build. To see all docker images created use below command.

**sudo docker images**



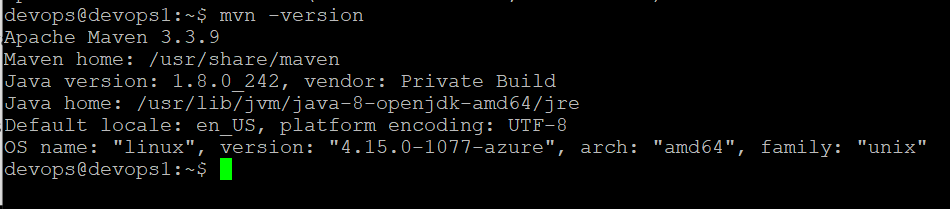
1. All the dependencies (java, maven, git) required to run this application is downloaded through Dockerfile. Now verify all these dependencies installed or not.
   1. Verify java version using below command

**java –version**



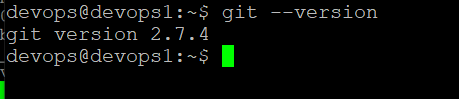
* 1. Verify maven version using below command

**mvn –version**



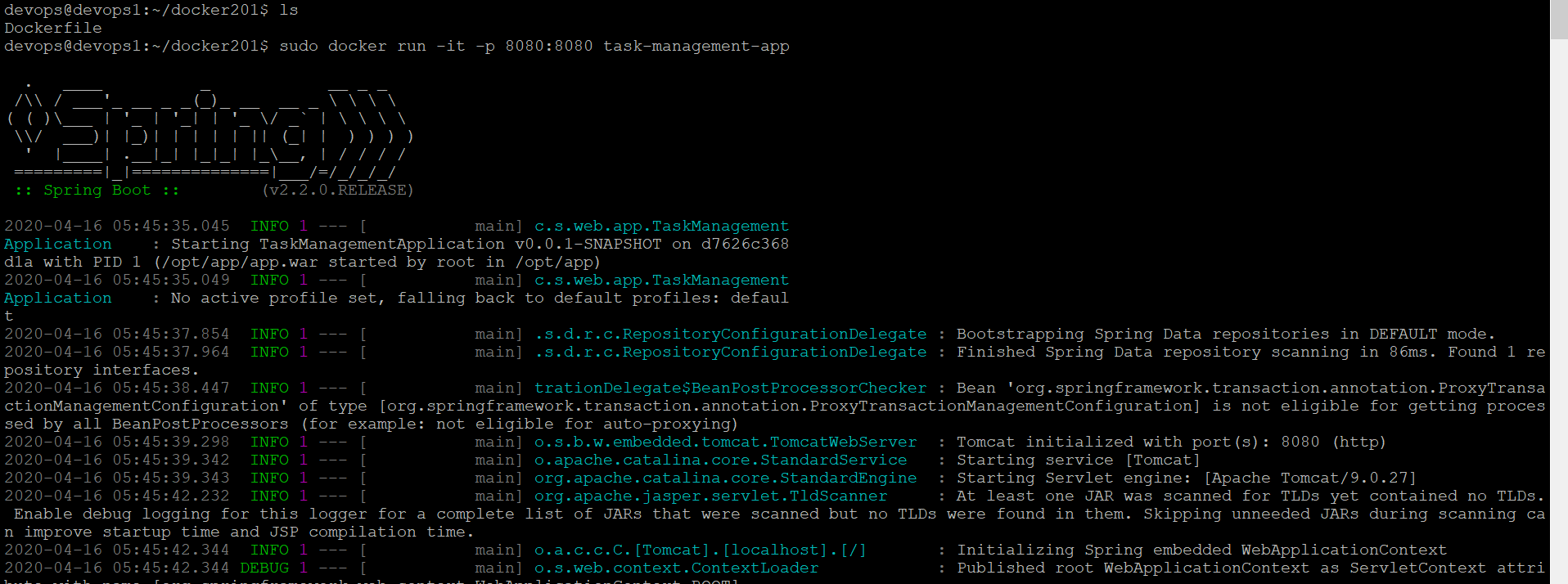
* 1. Verify git version

**git –version**



1. Now run the docker image using docker run command

**sudo docker run –it –p 8080:8080 task-management-app**

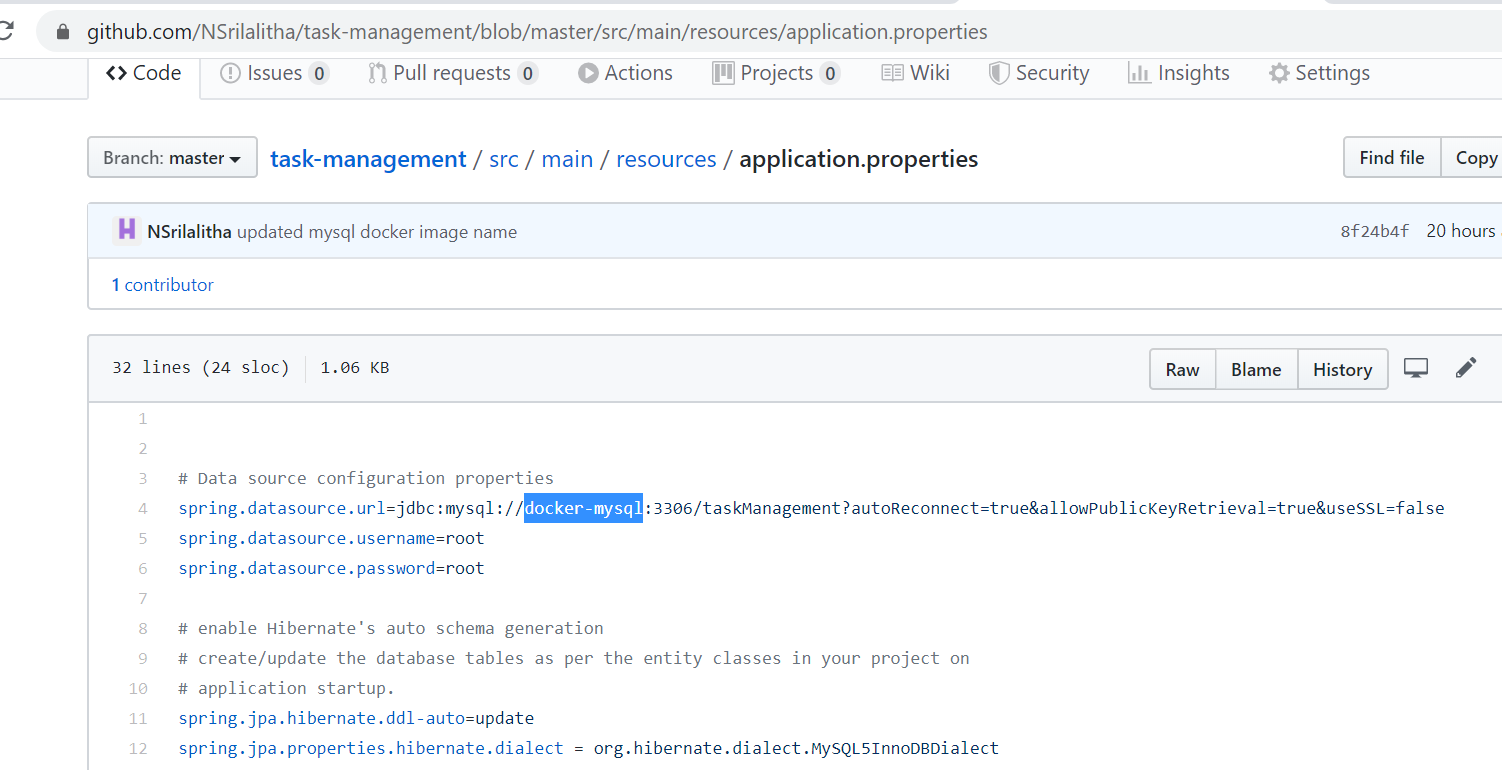


Here “docker run” is the command to create and run a docker container. “-it” is the interactive mode of display, “-p 8080:8080” is used to expose the container port to the host port. In this case, exported 8080 port of container to 8080 port of host machine.

In the application, we are using MySQL database for data persistence. In order to run the application using docker, we have to create docker container for MySQL as well. That MySQL docker container name should be specified in spring datasource property in application.properties file.

* Run a container based version of DB (e.g. MySql)  Get all these parts to work together.

To containerize the task-management application we need MySQL container as well. Now we will containerize MySQL using mysql base image. This container name should be specified in spring datasource property instead of localhost in properties file like below.



Here “docker-mysql” is mysql docker container, which will run on port 3306.

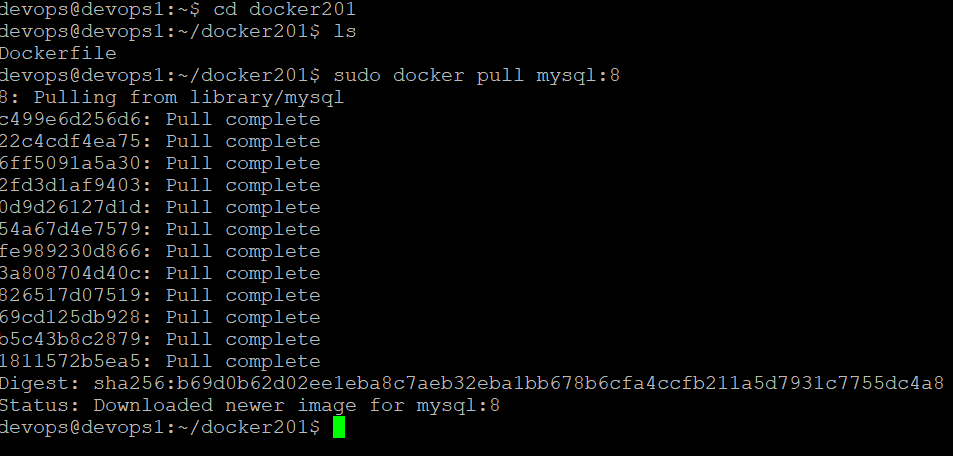
Now we will see how to create mysql container to run our application.

1. Move to docker201 folder

**cd docker201**

1. Now pull mysql base image with version 8.

**sudo docker pull mysql:8**



Run a container from this image. ‘--name’ gives a name to the container. ‘-e’ specifies run time variables you need to set. ‘-d’ tells the docker to run the container in the background. Set the below runtime variables for the MySQL container :

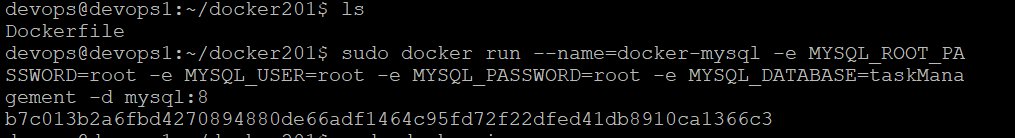
MYSQL\_ROOT\_PASSWORD=root

MYSQL\_DATABASE=taskManagement (this my database name for task management application)

MYSQL\_USER=root

MYSQL\_PASSWORD=root

**sudo docker run --name=docker-mysql –e MYSQL\_ROOT\_PASSWORD=root –e MYSQL\_USER=root –e MYSQL\_PASSWORD=root –e MYSQL\_DATABASE=taskManagement –d mysql:8**



1. Now check the status of this container using below command

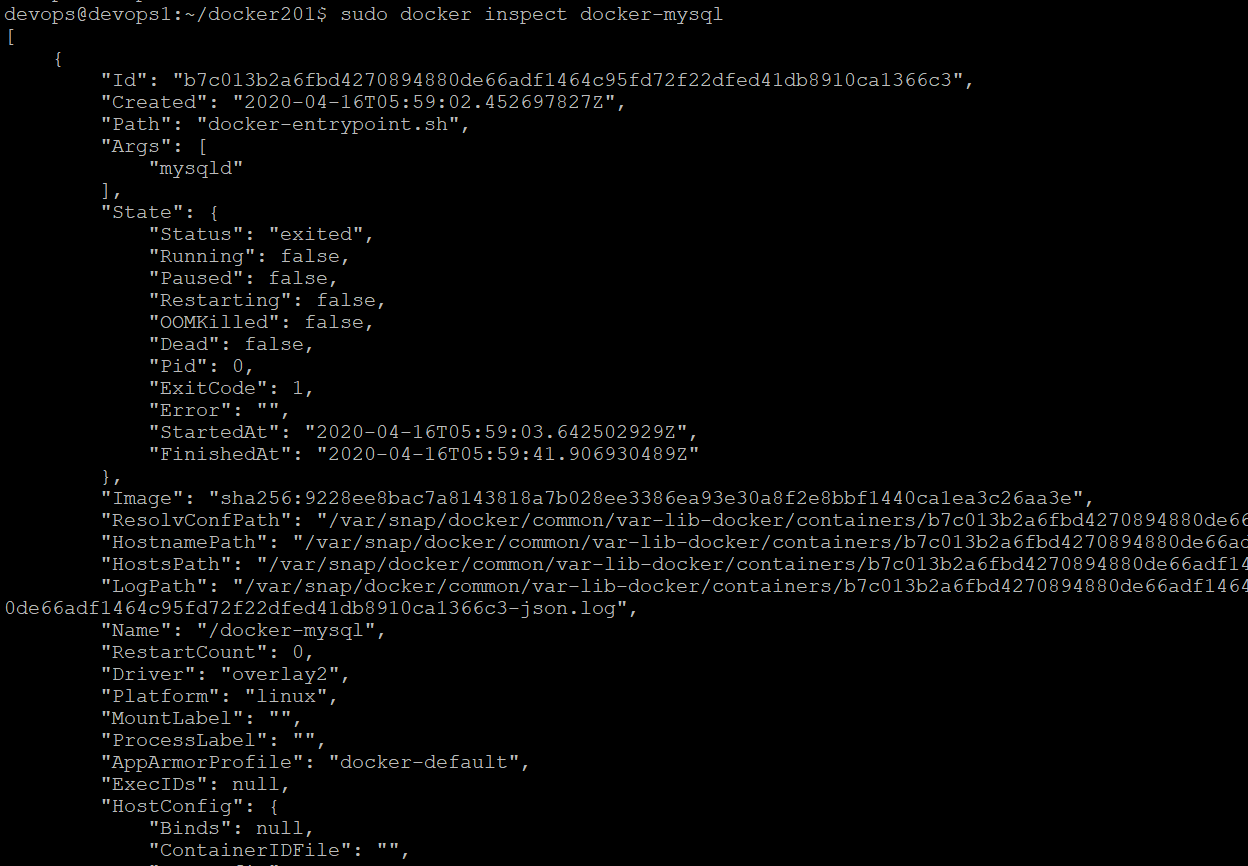
**sudo docker ps –a**



Now MySQL is running at port 3306.

1. To view the details about the container inspect it using below command.

**sudo docker inspect mysql-docker**





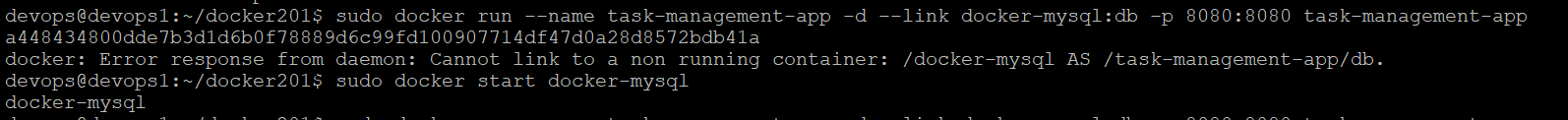
1. **Get all parts working together: Run the application by linking task-management-app with MySQL container**

Now we have task-management-app docker image for the task management application and docker-mysql container also created for MySQL database. Now we will run the application by linking task-management-app image with docker-mysql container since the application is dependent on MySQL database.

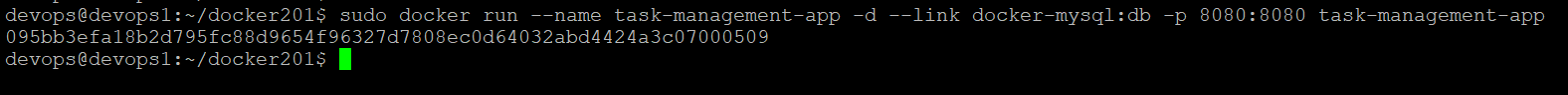
Below is the command to link both containers and run the application together.

**sudo docker run --name task-management-app –d --link docker-mysql:db –p 8080:8080 task-management-app**

‘-d’ indicates run the application in the background

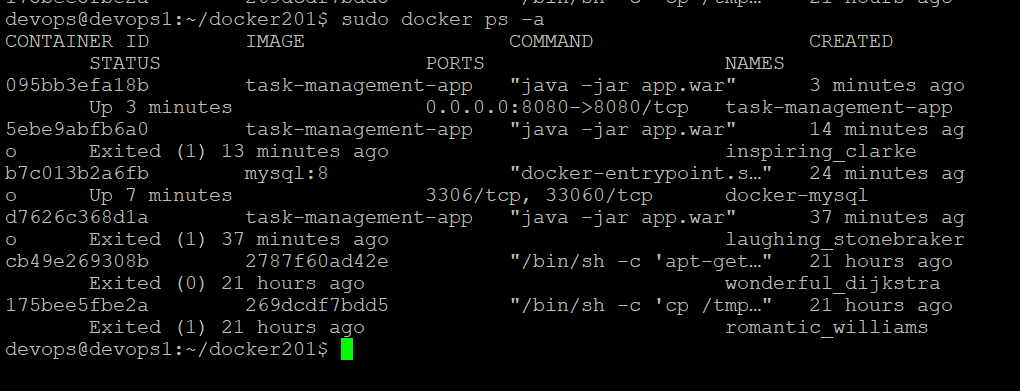


Got error saying “docker-mysql” container is not started. So started “docker-mysql” container as shown in above screenshot using “**sudo docker start docker-mysql**” command. Now again run the container.



Now task-management-app container is started at port 8080. Both task-management-app and docker-mysql containers are up. To verify running containers status use below command.

**sudo docker ps –a**



Application is started at port 8080, verify it in browser.

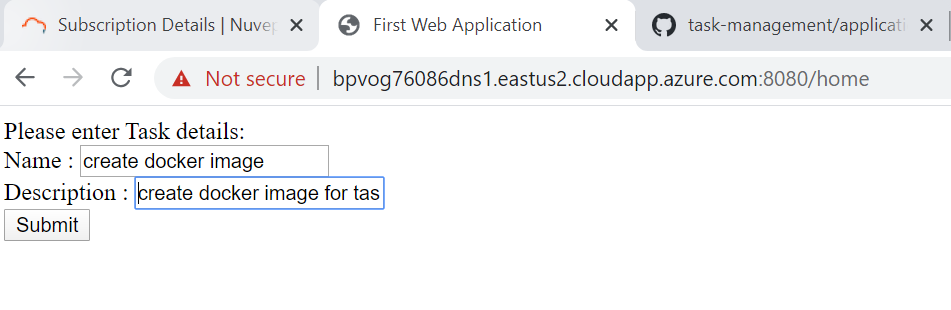
<http://bpvog76086dns1.eastus2.cloudapp.azure.com:8080/home>

Application is loaded. It is showing home page where we can add task details and submit.

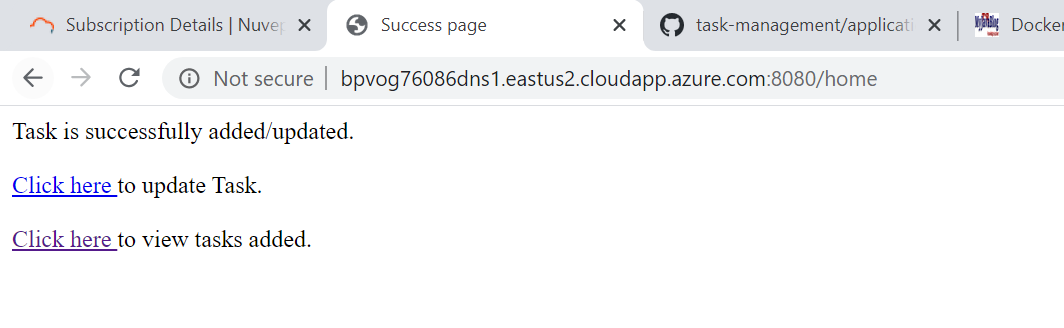


Now we will add tasks and verify the tasks are added or not. We can view the tasks that got added to MySQL database.

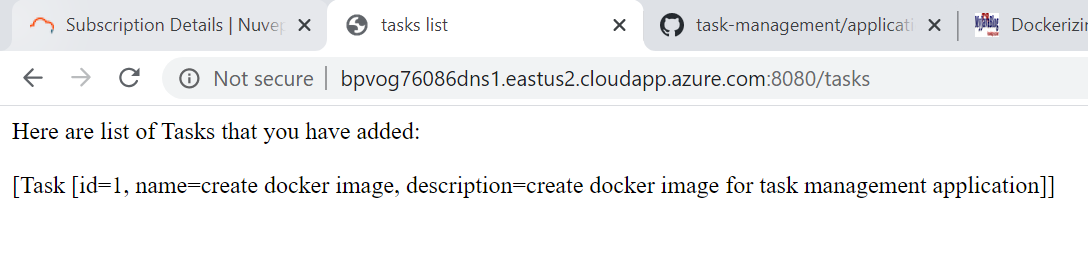
1. Add task details and click on submit



1. When click on submit below success page is displayed.

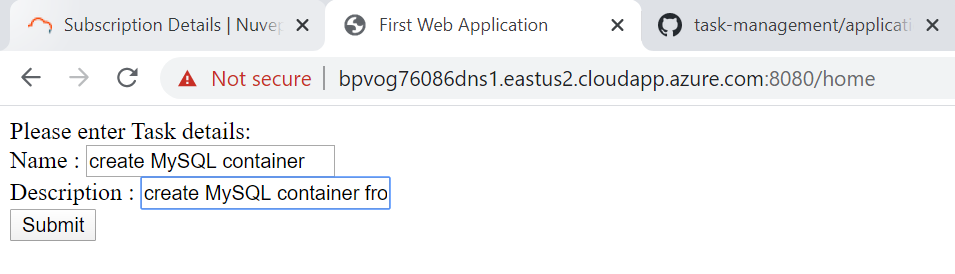


1. Now click on view tasks related link to view the tasks that got added. When click on view tasks link, tasks page is displayed.

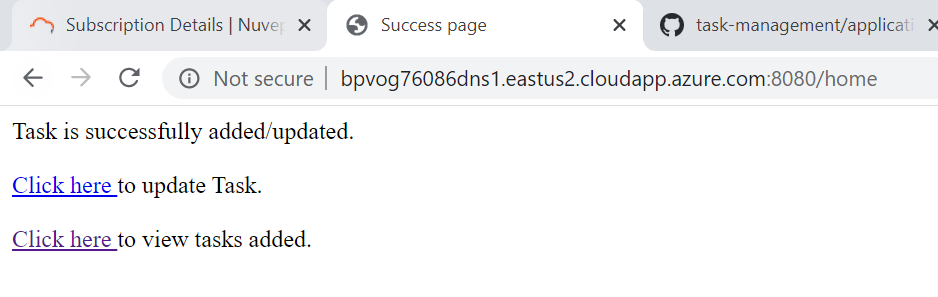


Here task is successfully added.

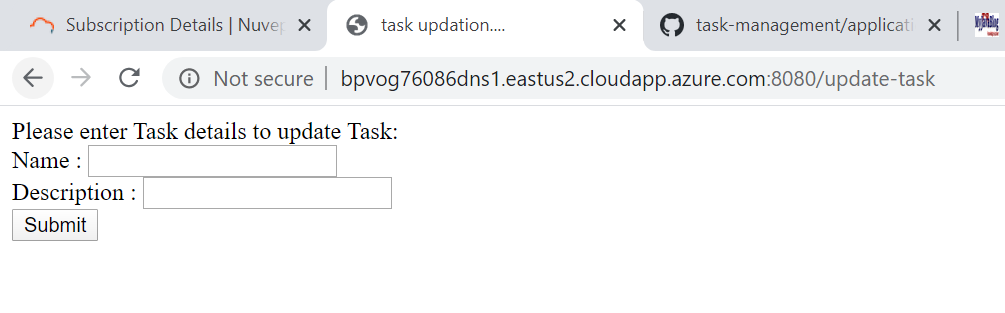
1. Now we will add one more task to the application. Go to home page again and add another task.



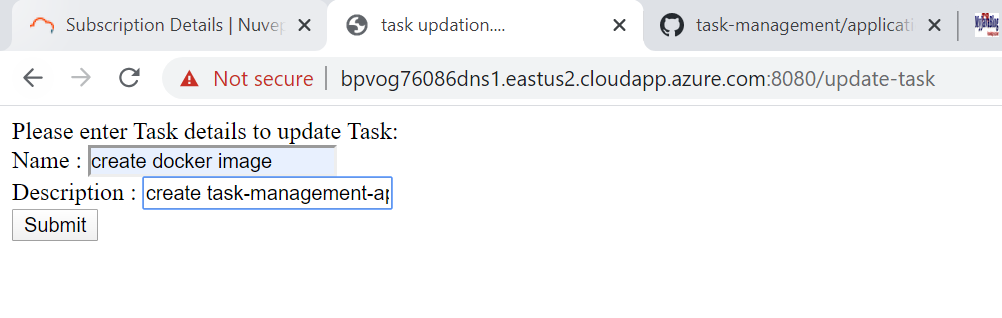
1. When click on submit success page is displayed as shown below.



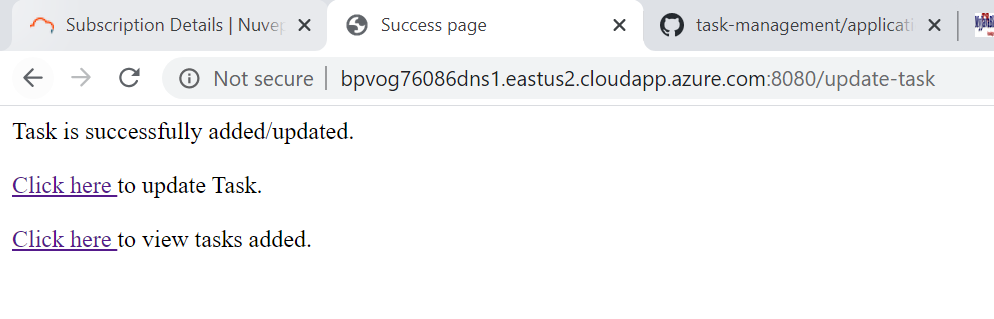
1. Now we will update the task by clicking on update task link. When click on update task link, update-task page is loaded as shown below.



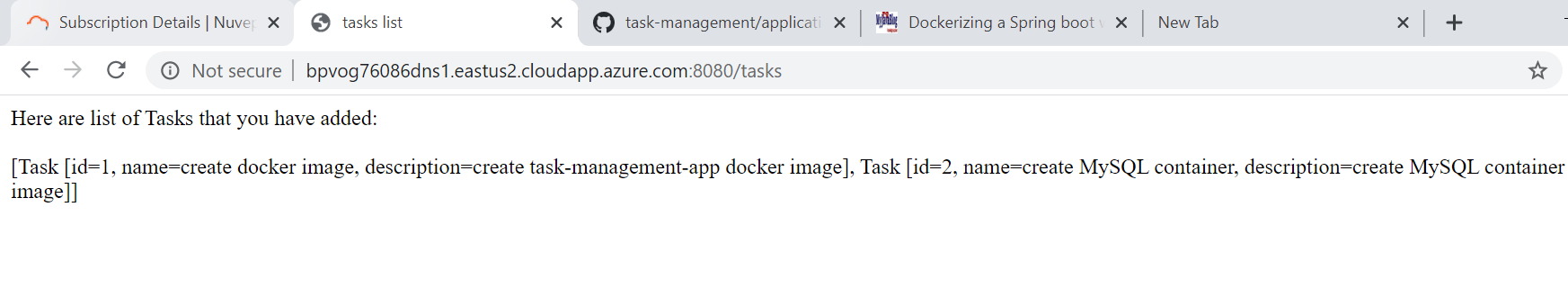
Now I will update the first task that got added to the application.



Click on submit. On click on submit, below page is displayed.



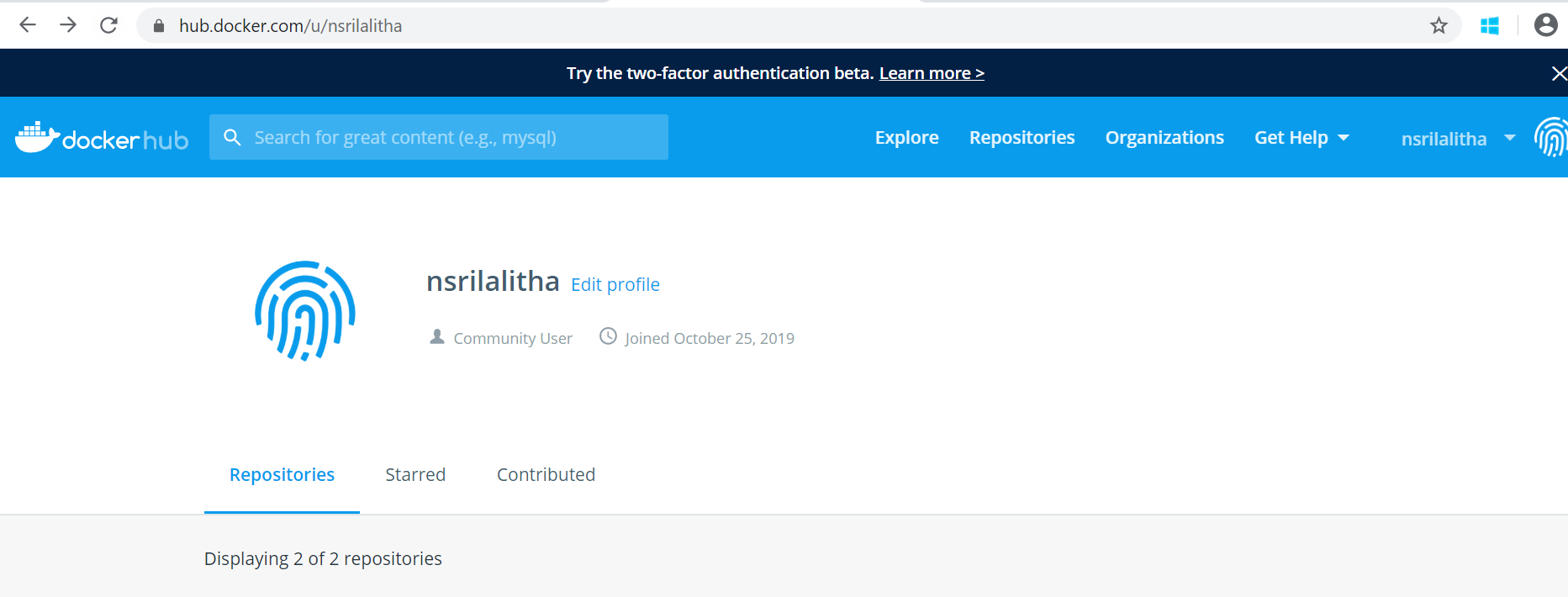
1. Now click on view task link, to verify whether task got updated or not.



Both tasks are successfully added. First task is updated as mentioned in step 6.

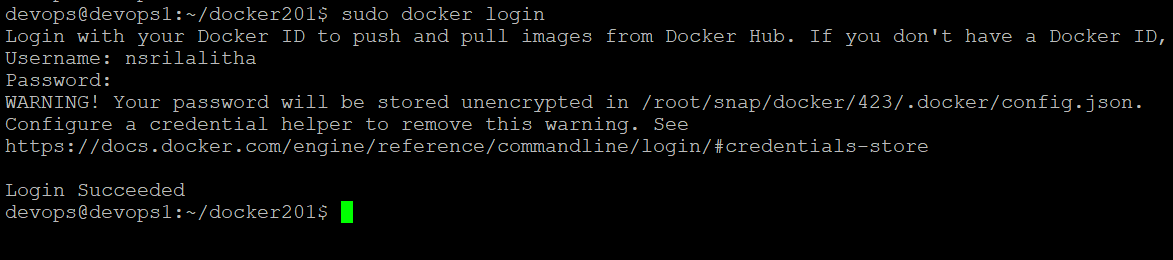
* Push the image to the Docker Hub

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>

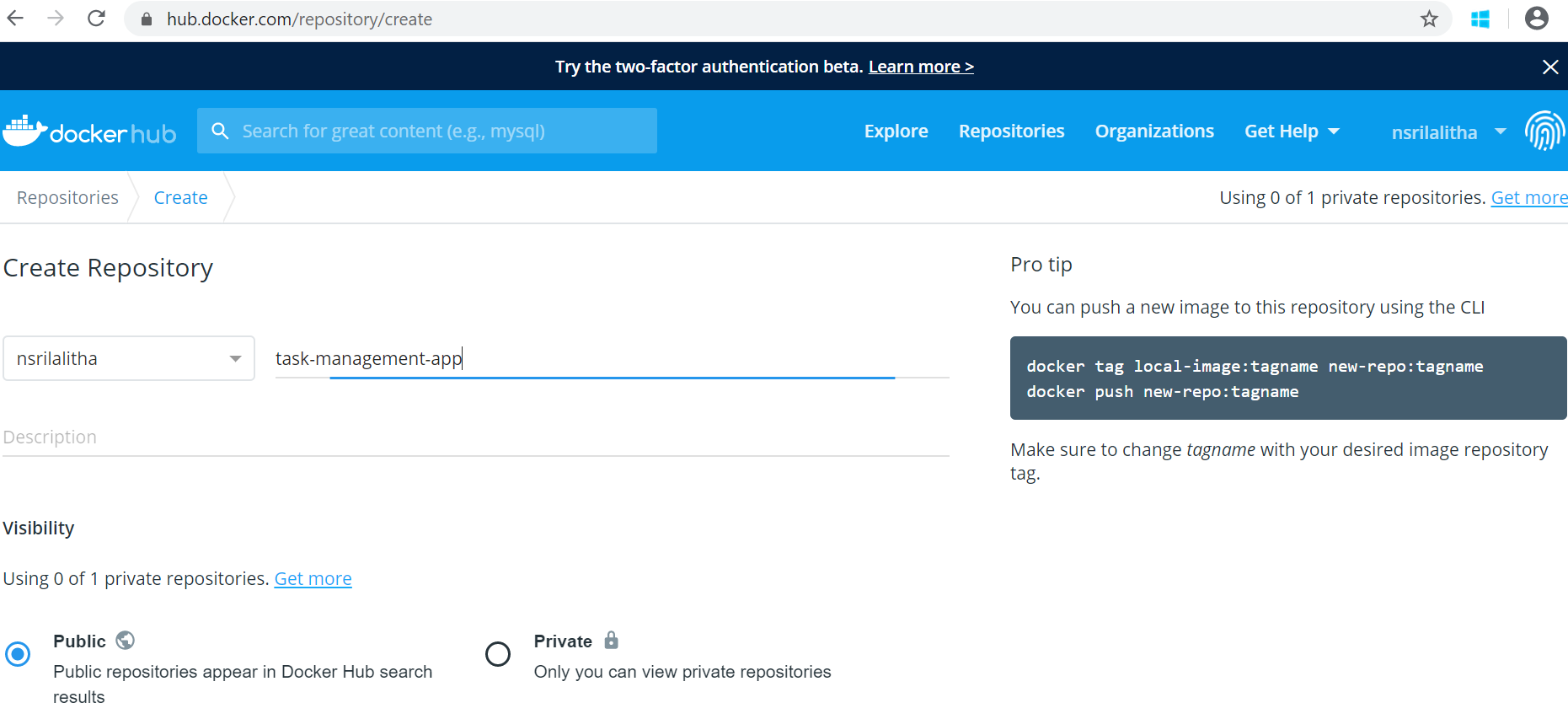


1. Login to docker hub from command prompt using below command

**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”.



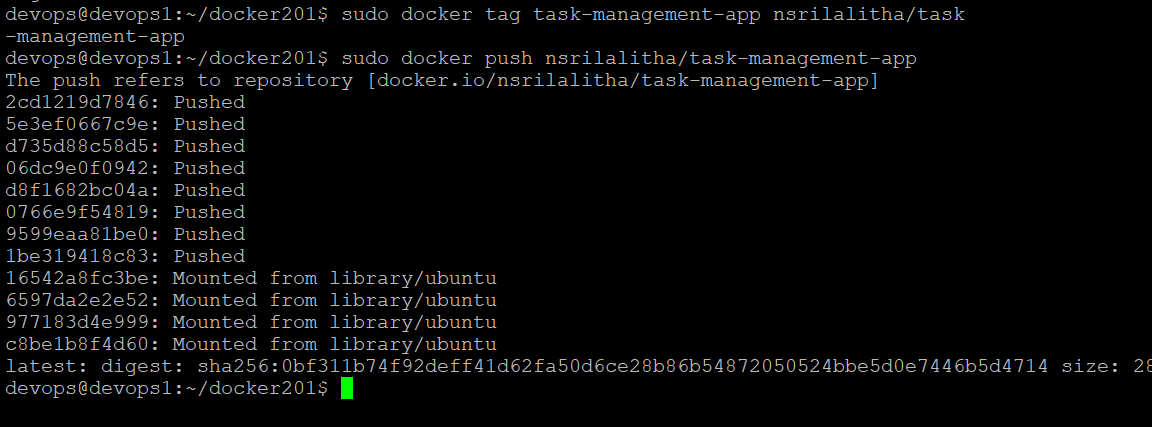


1. Now create a tag to link our image to docker repository. Use below command to create tag for this image

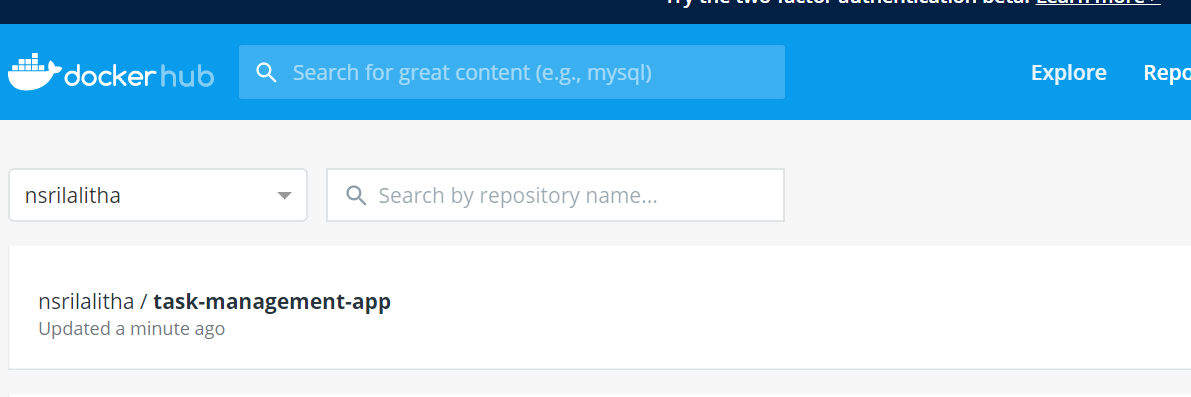
**sudo docker tag task-management-app nsrilalitha/task-management-app**

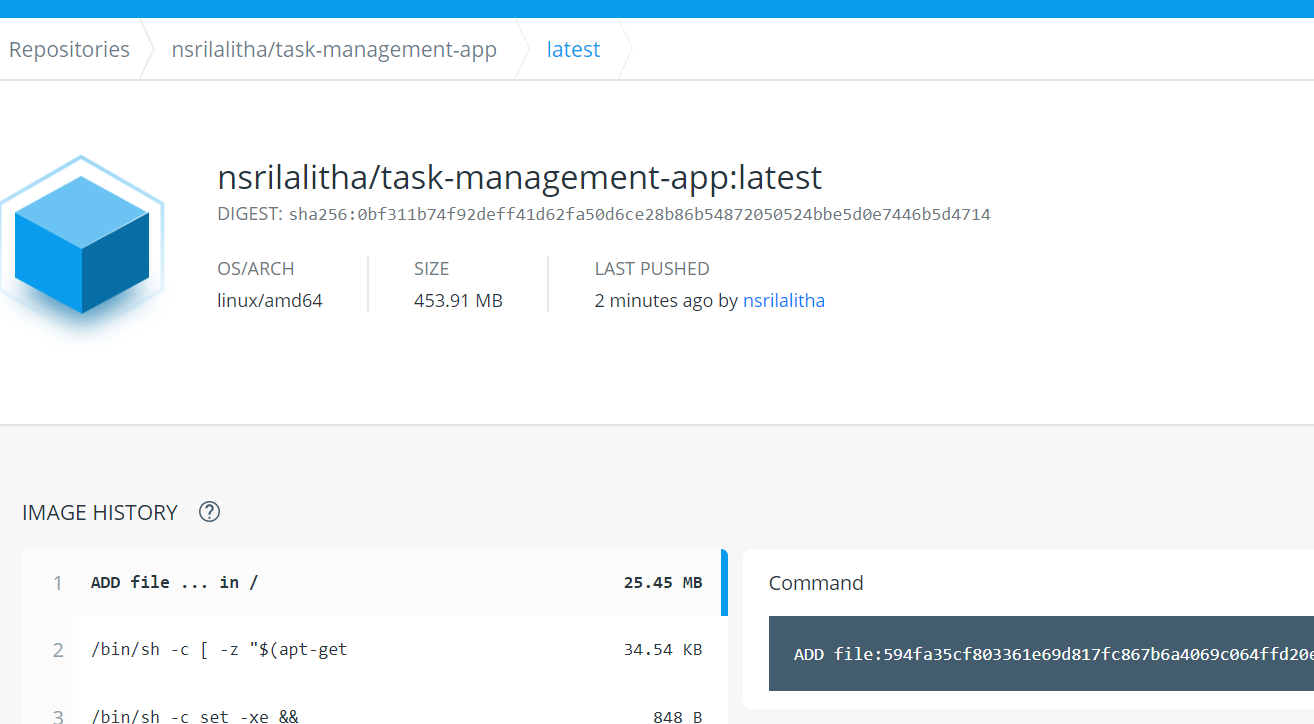
1. Use below command to push the image to created repo in docker hub

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



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

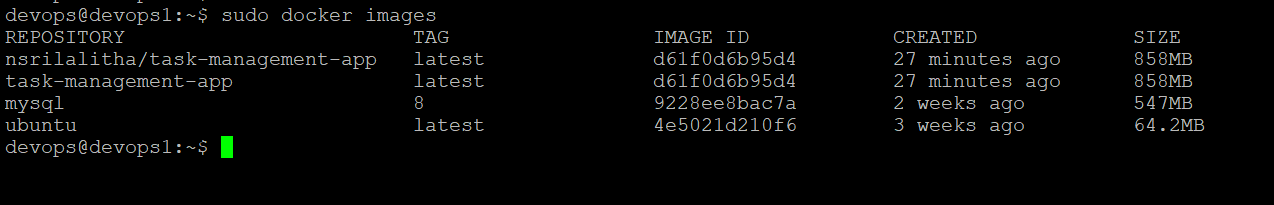




Now any one can access the docker image created for this application in below link

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

Now verify docker images.



* Script the spinning up of the containers using Docker Compose

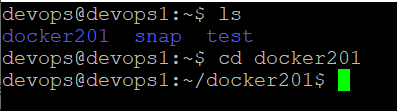
In the above section, we saw how to link task-management-app with docker-mysql container to run the application using docker run command with --link flag. Instead of manually creating link like this, we can use docker-compose to run this task management application. In the docker-compose file we have 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.

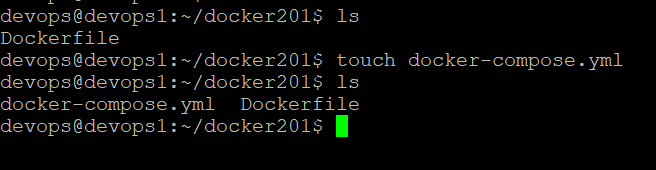
1. Now go to docker201 folder

**cd docker201**



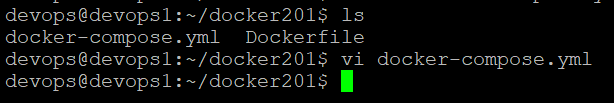
1. Create docker-compose file with name docker-compose.yml

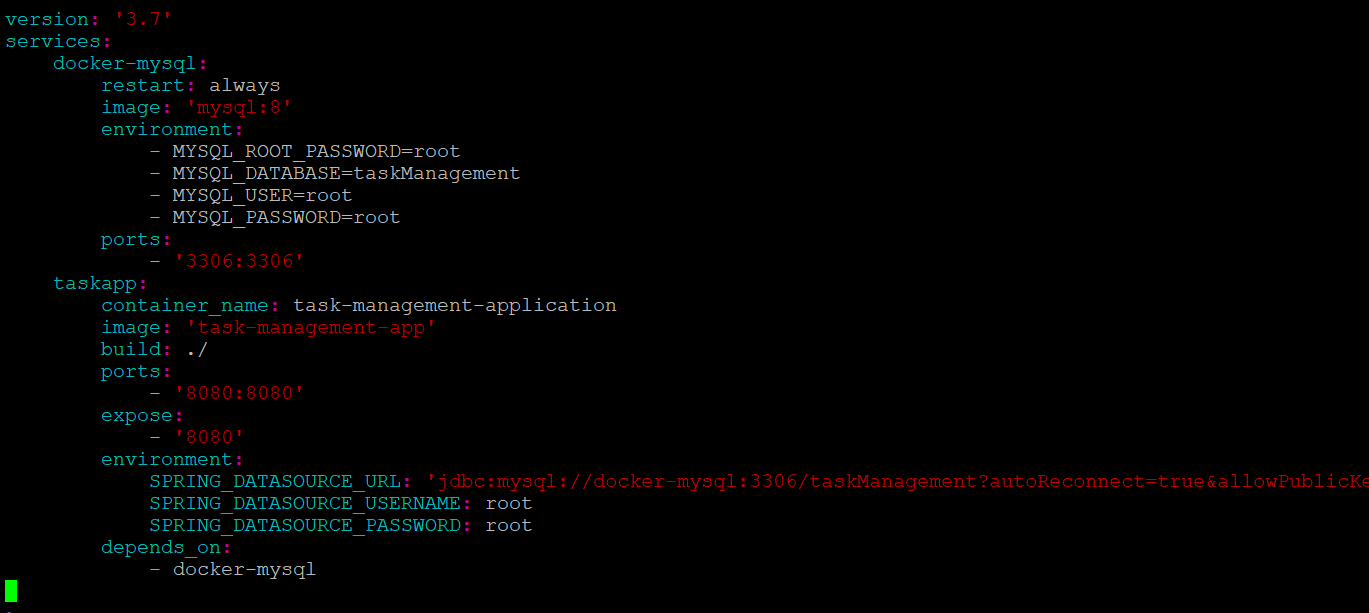
**touch docker-compose.yml**

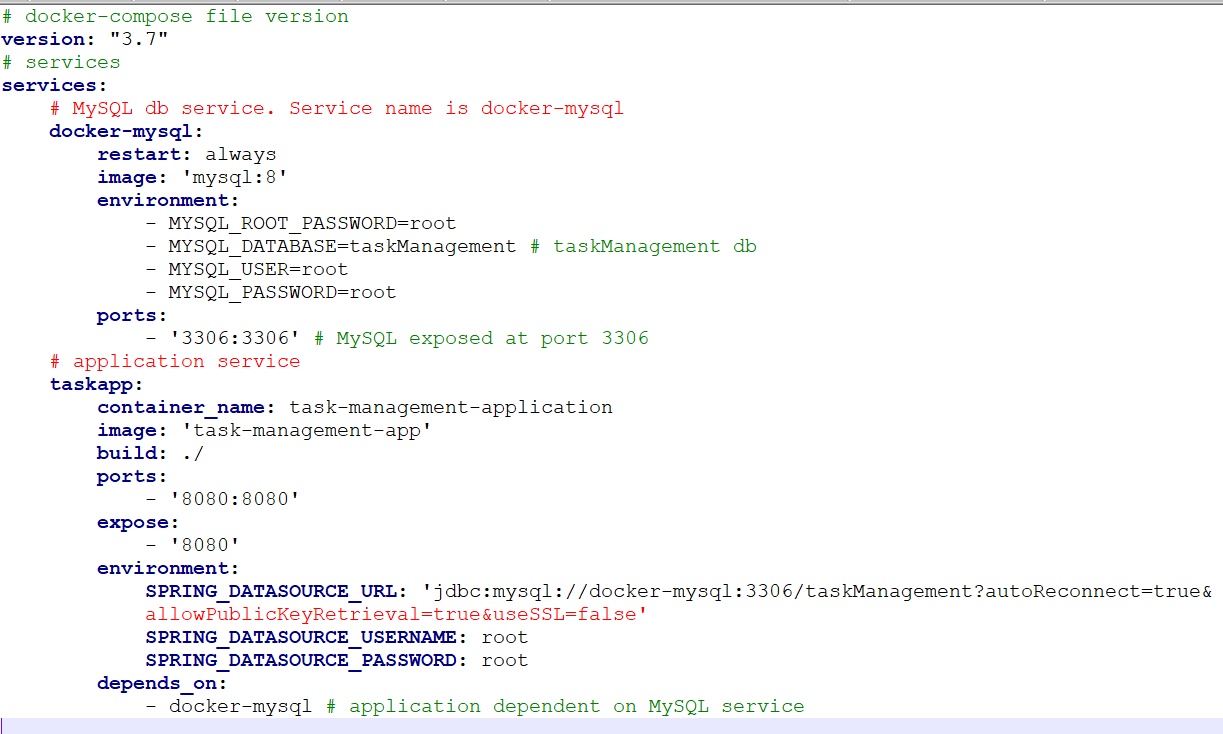


1. Now docker-compose file is created. We have to edit and add contents in docker-compose file to make application run at port 8080. To edit use below command.

vi docker-compose.yml





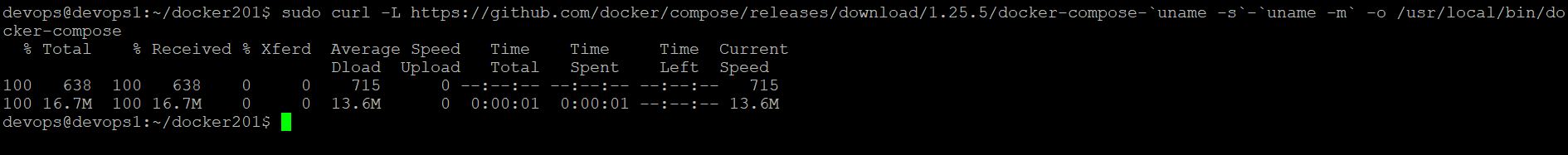




Here we defined two services – one for spring boot application, one for MySQL database. For MySQL service, gave name as ‘docker-mysql’ which will be a docker container. Base image is “mysql with version 8”. Exposed port as “3306” and specified environment variables like username, password, db name. When 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**

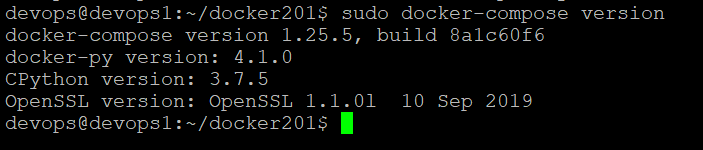


Now set permissions using below command.

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

Now verify version using below command

**sudo docker-compose –version**



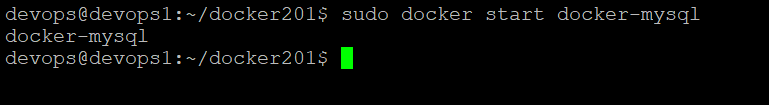
1. To start the application using docker-compose, we should ensure the dependent services are started. Here the application is dependent on docker-mysql container, before starting our application docker-mysql container should be up and running at port 3306. To verify docker-mysql is running or not, use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



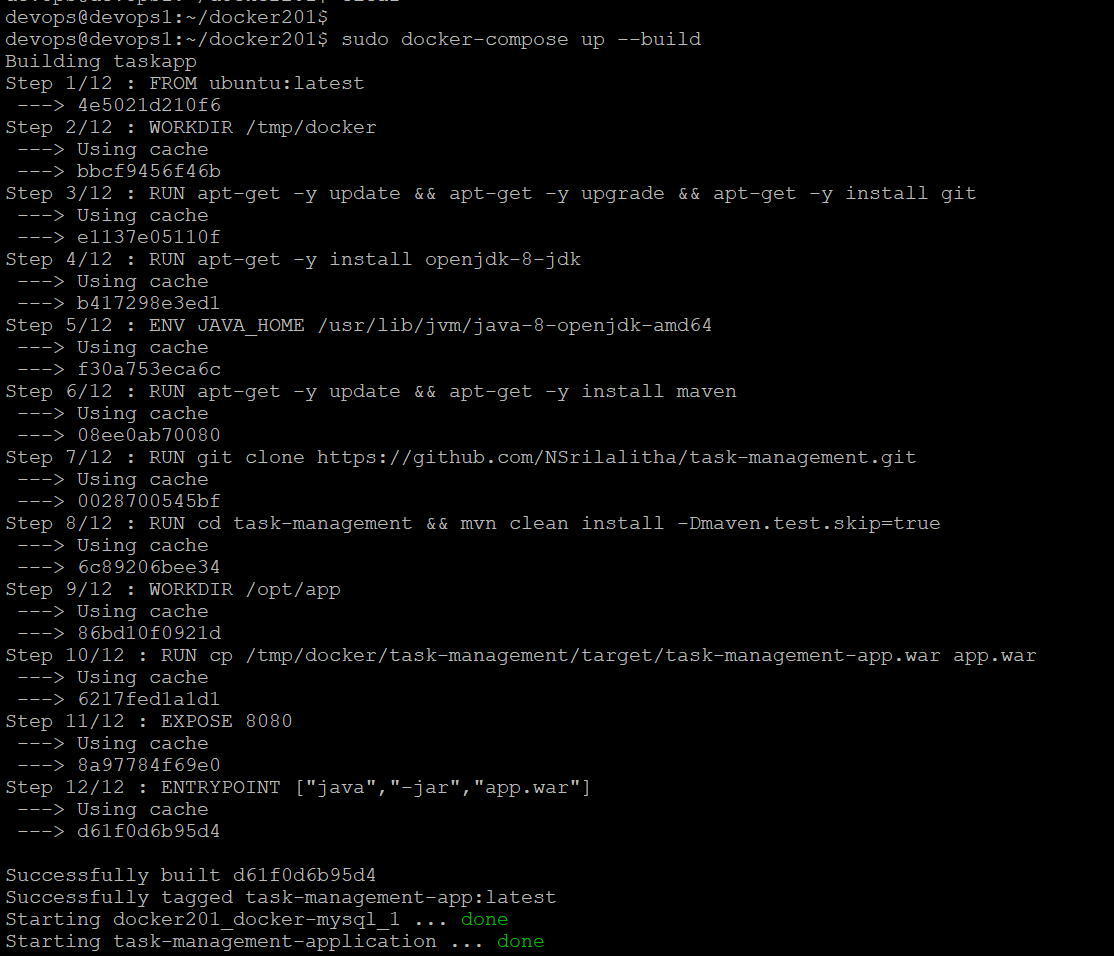
Now verify status of “docker-mysql” container, by again running sudo docker container ps –a command.



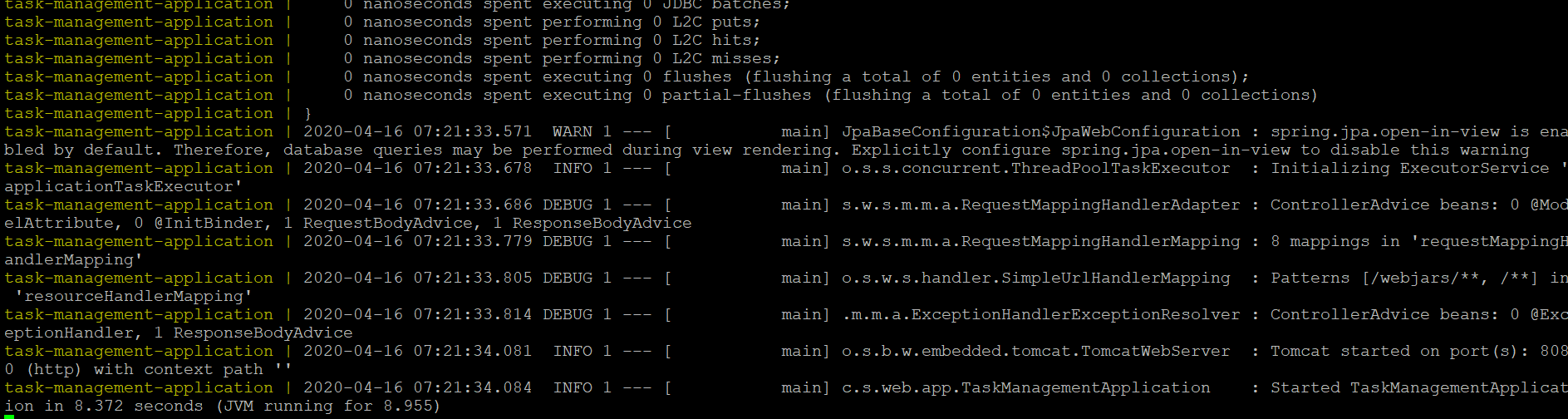
Now status of “docker-mysql” container is “Up”. Since dependencies are ready, Now we can run our application using docker-compose.

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**





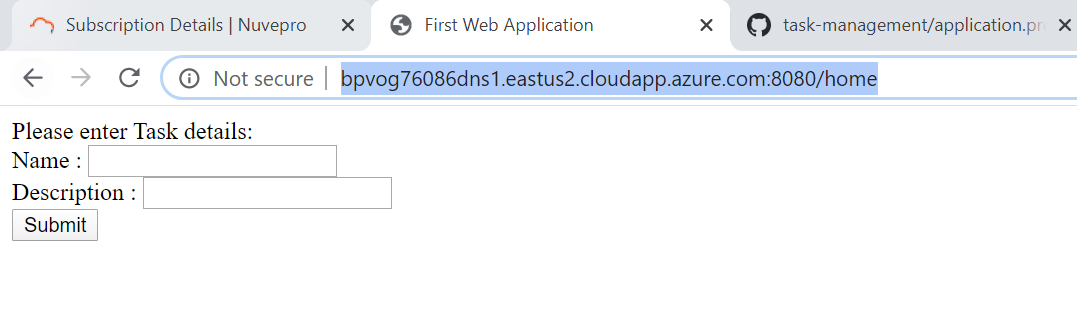


Now application is started at port 8080.

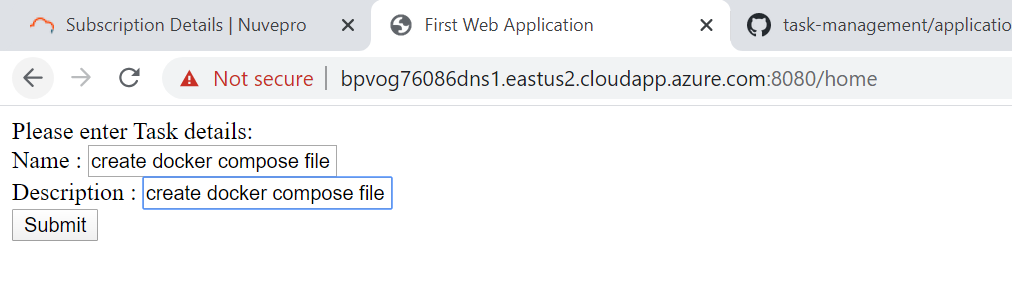
Cloud lab host is bpvog76086dns1.eastus2.cloudapp.azure.com

So we can access this application home page at below url

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

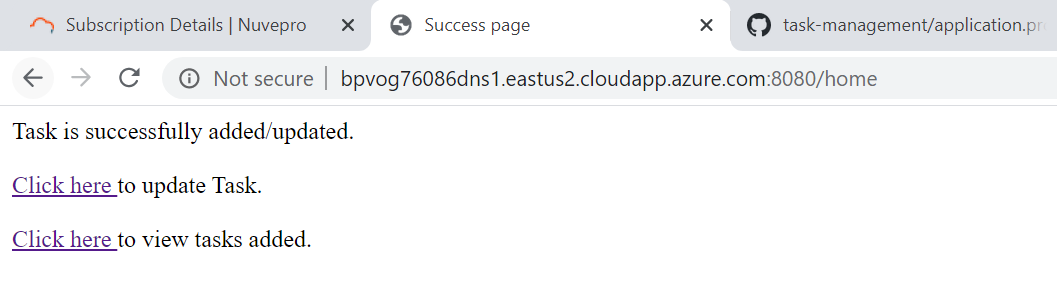


1. Now enter task name and description which will persist these details in taskManagement db.

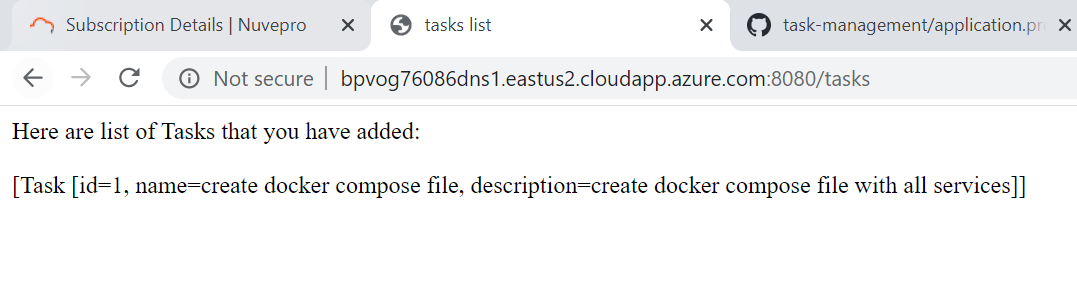


After entering details and when click on submit, task details successfully saved to db.

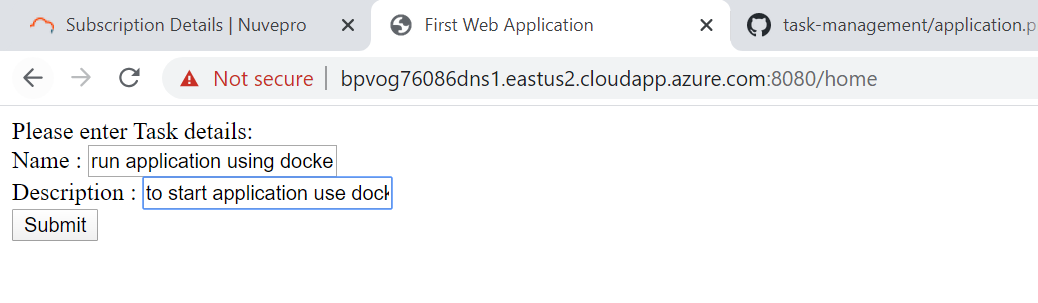
1. On submit, success page is displayed as shown below



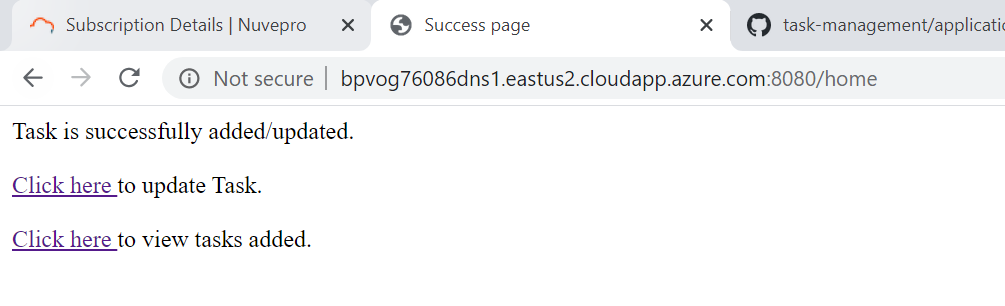
1. Now click on view tasks link to view the task that got added to application.



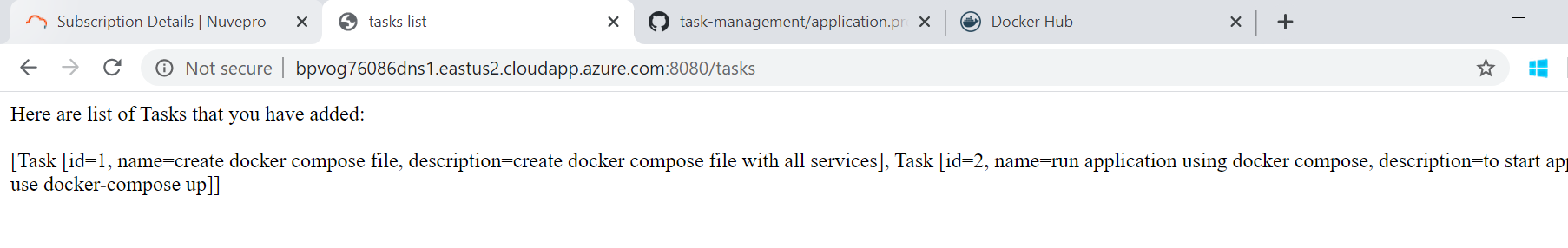
1. Again go to home page and add one more task



1. Now click on submit, on submit success page will appear.



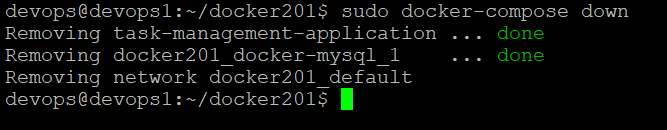
1. Now click on view task link to verify both tasks are available or not.



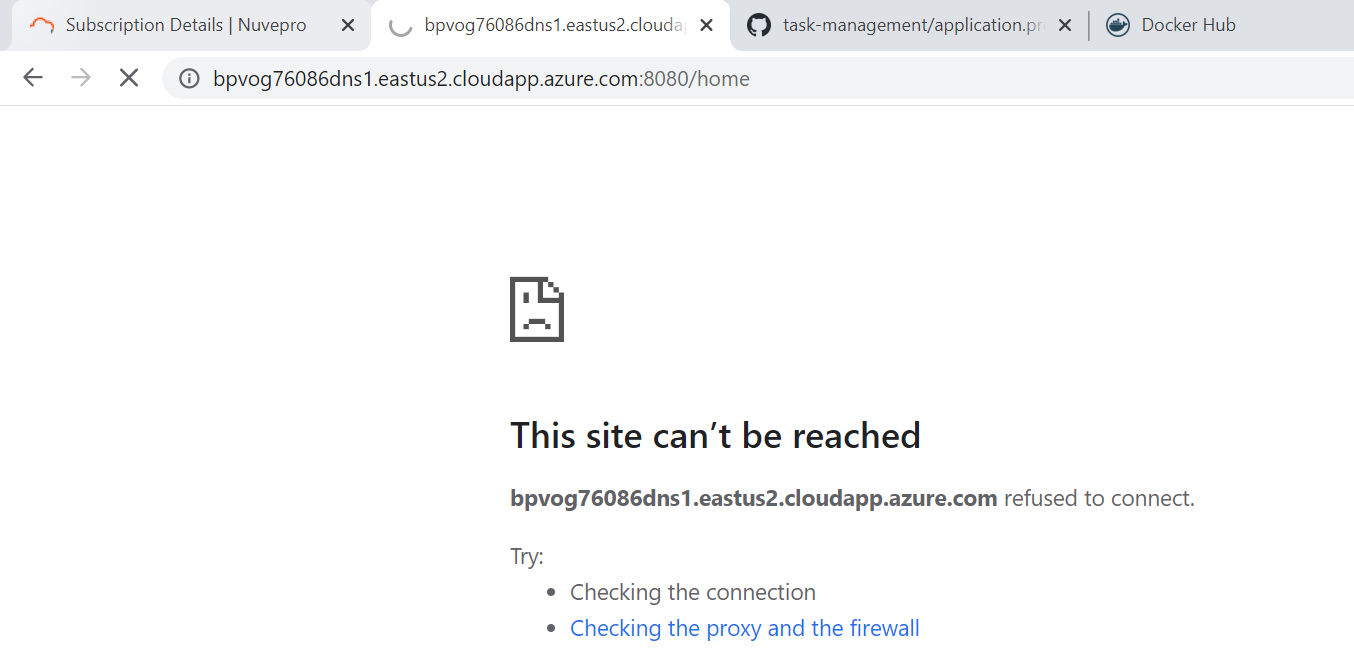
Tasks are successfully added and viewed.

**Scale down**: To scale down the containers, we can use docker-compose down command as shown below.

**sudo docker-compose down**



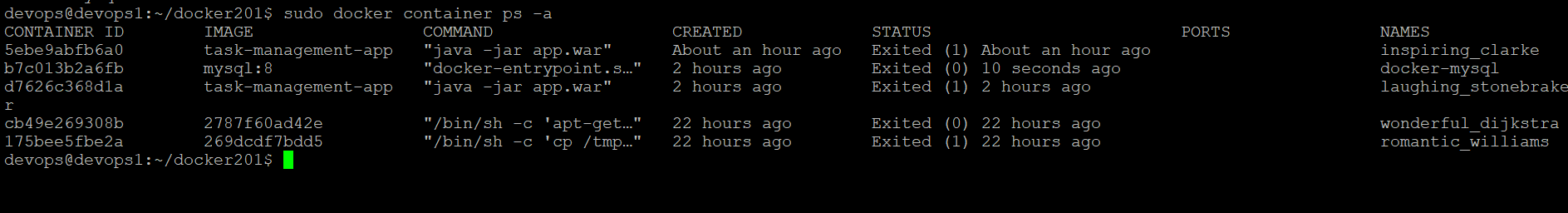
Now go to application home page and verify! Application can’t be accessed since containers are scaled down. Server is stopped.



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.

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

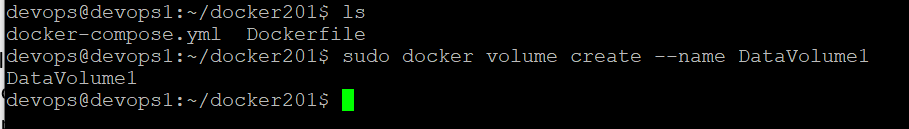
Since we are not maintaining any data volumes here, whenever containers stopped, the data also getting lost. Here the data is persisted within the container. So whenever container is started running, if we run the application , if we add tasks, only those tasks are getting added to the db which is within the containers. So, to make available this data to all containers or to separate data from containers we can use data volumes which can be achieved using docker volume.

Docker volume: Docker Volumes can be created and attached in the same command that creates a container, or they can be created independently of any containers and attached later. Here, we create Docker volume independently and attach it to existing containers.

Let’s create a docker volume and attach it to container.

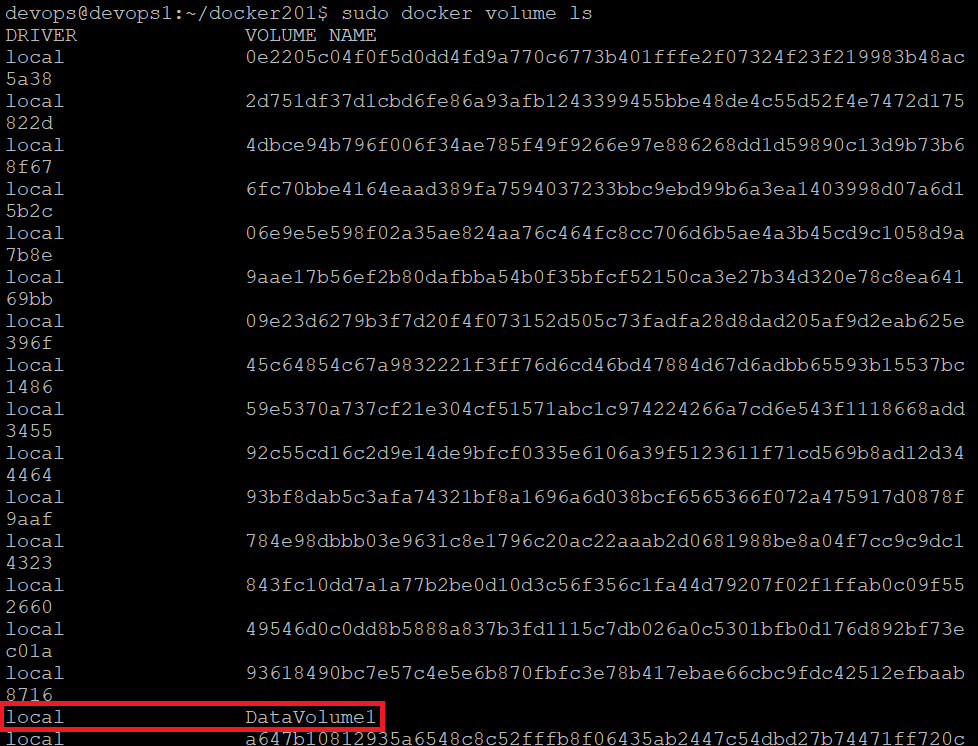
1. Create docker volume without relating it to any container using below command.

**sudo docker volume create –name DataVolume1**



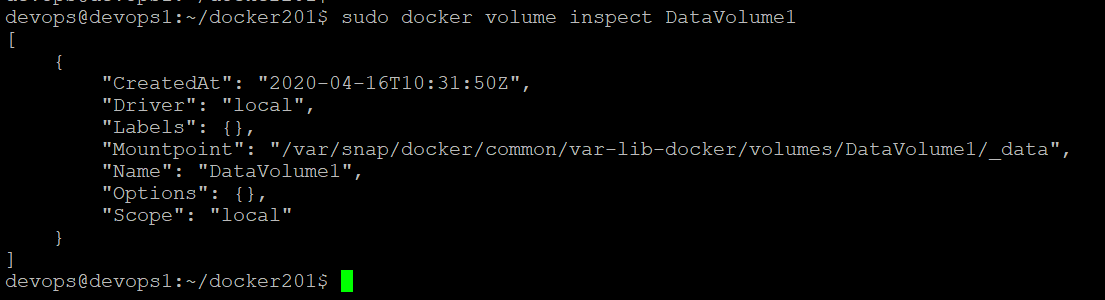
1. List volumes using below command

**sudo docker volume ls**



1. Inspect docker volume using below command

**sudo docker volume inspect DataVolume1**

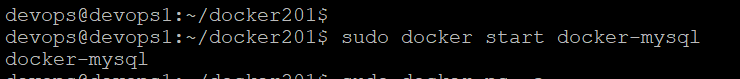


1. Start a container with data volume

In order to run task management application, we need task-management-app container for spring boot application and docker-mysql container for MySQL database. We will link these two containers and attach volume with ‘-v’ flag and run the container using docker run command. So to run the application first start ‘docker-mysql’ container since application is dependent on this container.

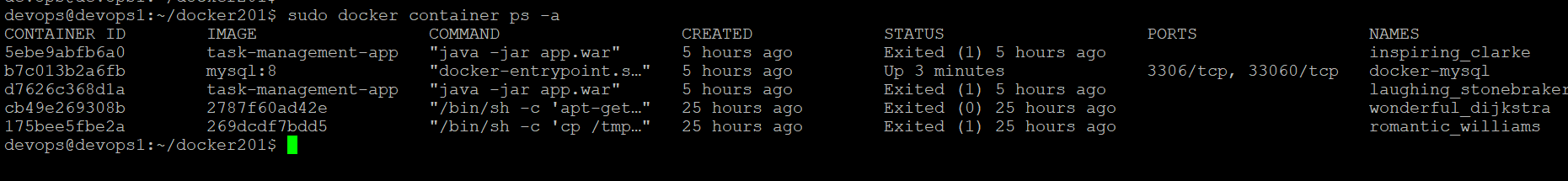
Start ‘docker-mysql’ container using below command.

**sudo docker start docker-mysql**



Now verify containers status using below command

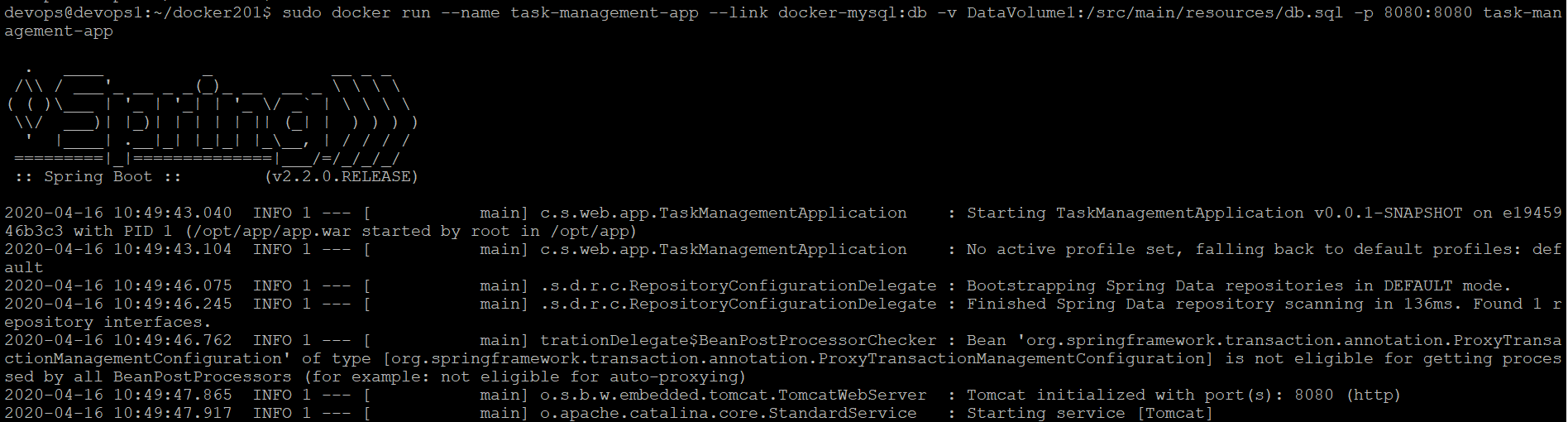
**sudo docker container ps –a**

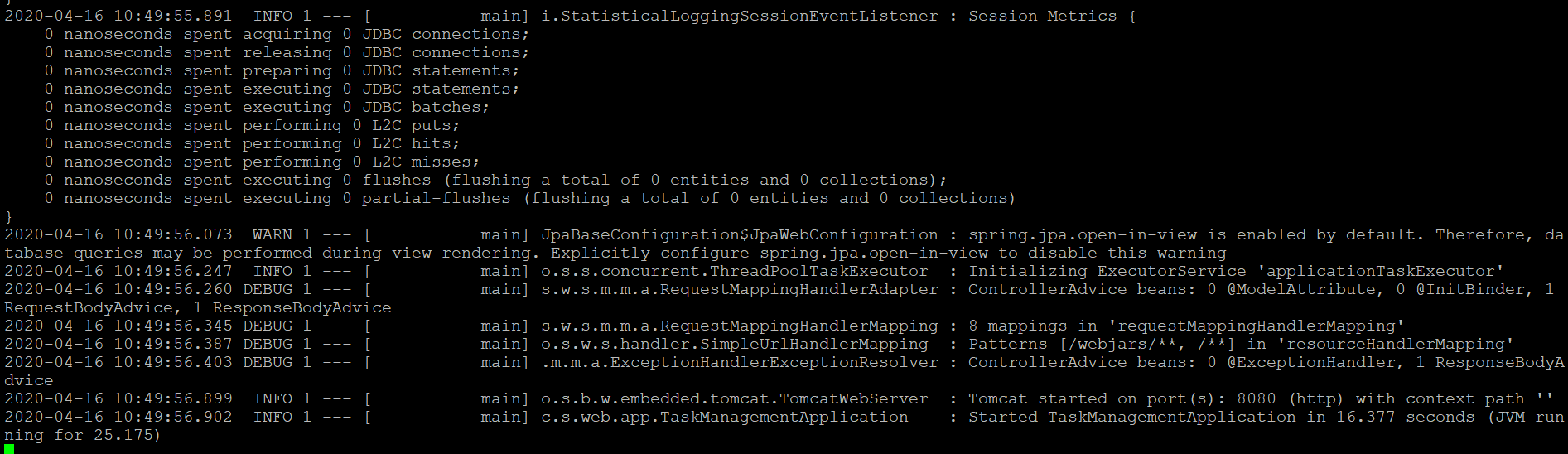


Now docker-mysql container is up. Now we will start the container to run the application.

Use below command to run the application by linking two containers and attaching data volume.

**sudo docker run --name task-management-app --link docker-mysql:db -v DataVolume1:/src/main/resources/db.sql –p 8080:8080 task-management-app**

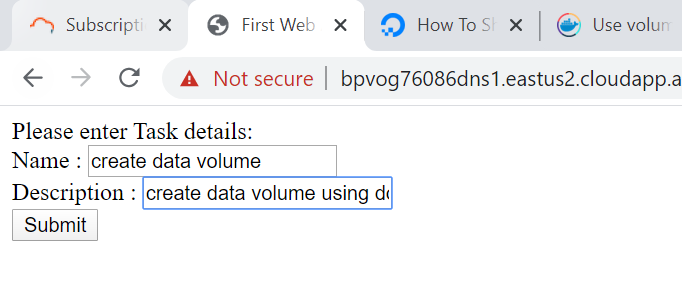




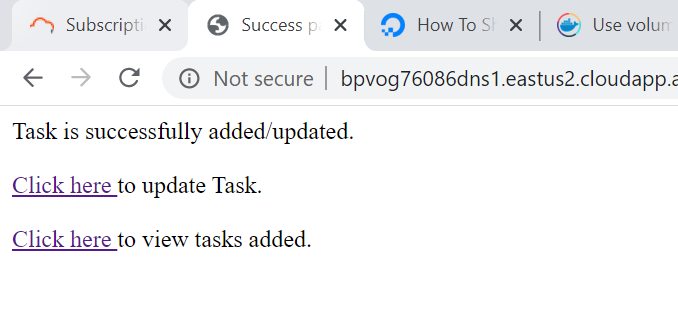
Now application is started at port 8080. Verify it in browser.



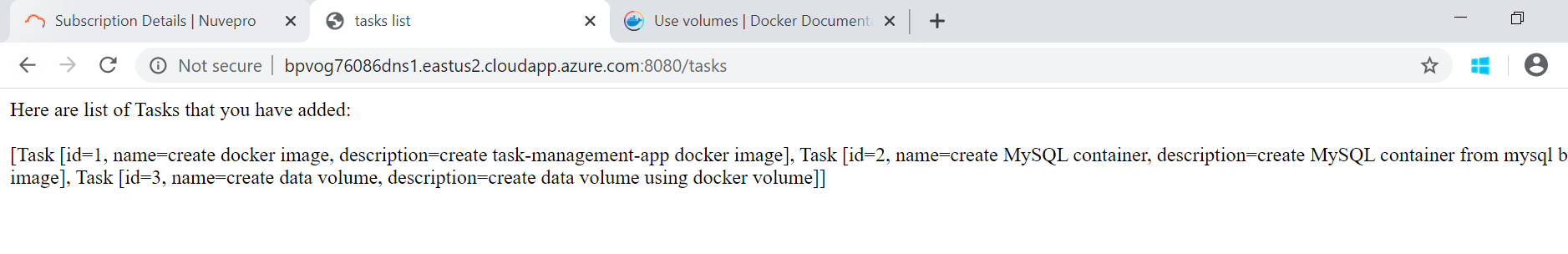
1. Add task details and submit.



1. On click on submit, success page is displayed.



1. Now click on view tasks link to verify the task got added or not.



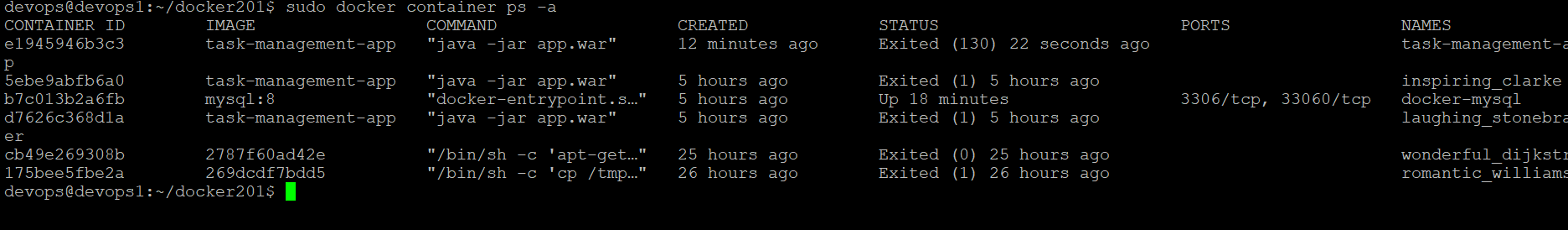
Task is successfully added.

Now stop the container and rerun the container again to see the data now we added is persisted or not. If we are not using volume, then data will be persisted within the container and whenever container is stopped the data will be lost. This scenario we have seen in above sections. Since here we are using docker volume, the data should not be lost.

1. Stop container

Stop running container using Ctrl+c and verify the status of container using docker container ps –a command.

**sudo docker container ps –a**



Now status of ‘task-management-app’ container is showing as ‘Exited’. Now we will remove this container using below command.

**sudo docker container rm –f e1945946b3c3**

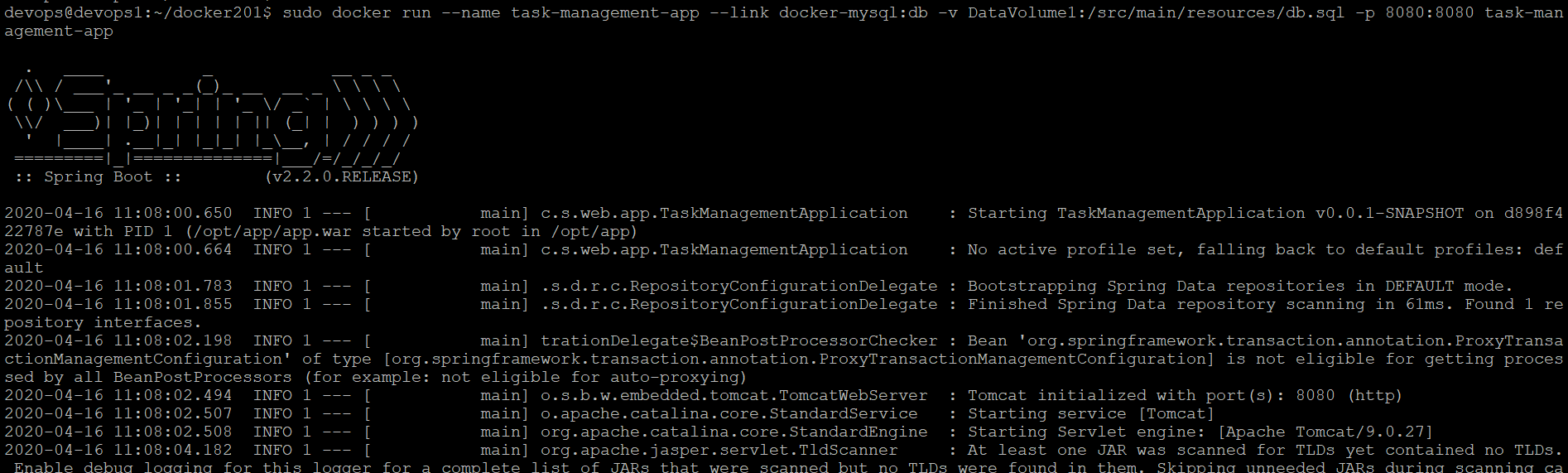


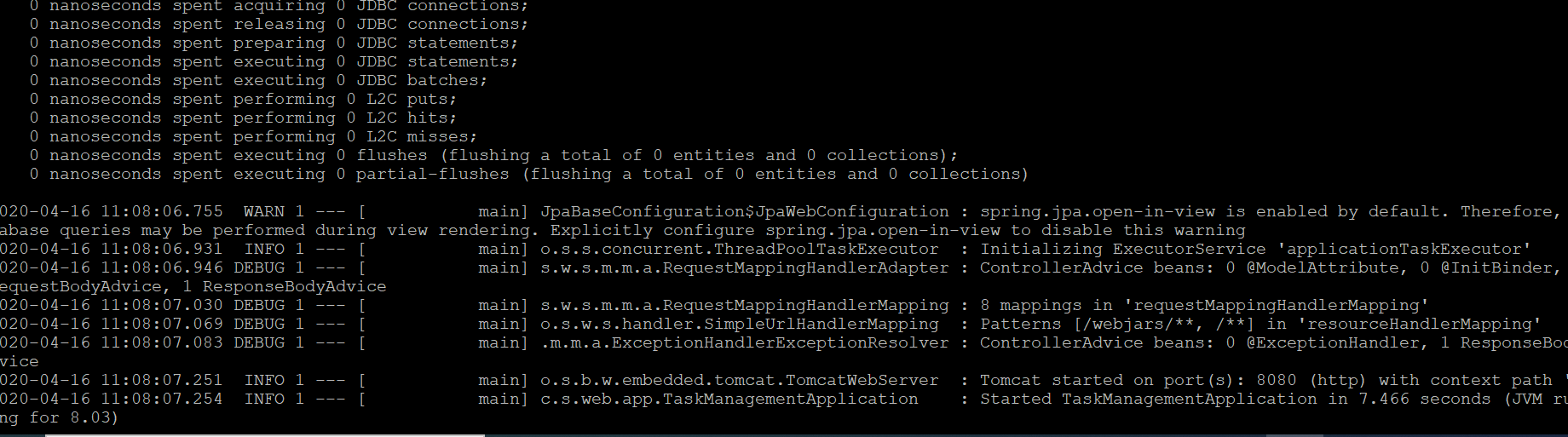
‘task-management-app’ container is successfully removed.

1. Re run the container with data volume

Use below command to run the container with volume

**sudo docker run --name task-management-app --link docker-mysql:db -v DataVolume1:/src/main/resources/db.sql –p 8080:8080 task-management-app**





Application is started at port 8080. Verify it in browser!

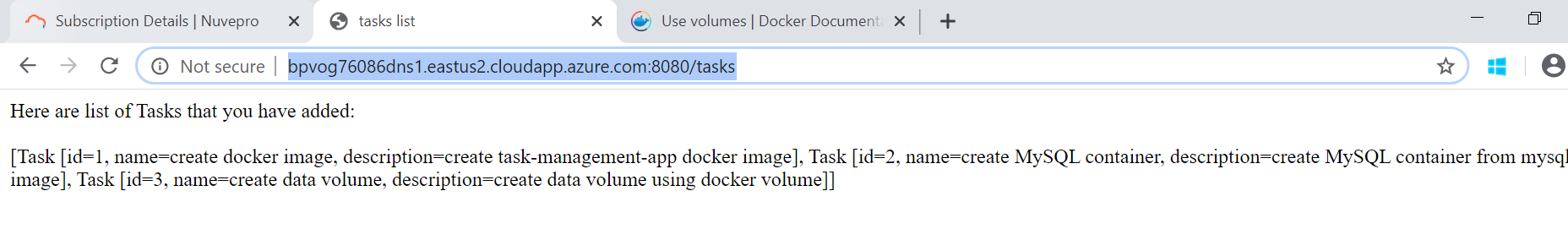
Go to home page using below url

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



Now verify previously added tasks, by using below url

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



Here the tasks added by another container is now available with this container due to docker volume. Here we have separated data from container, So the data is available for all the containers in the host machine.