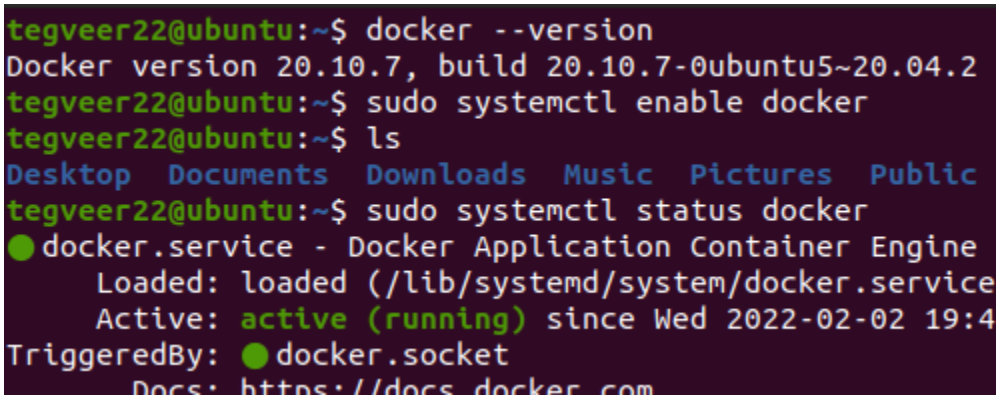
Project Milestone: Virtualization and Containerization

Name: Tegveer Singh
ID: 100730432

1. **Containerization VS Virtualization**

Virtualization	**Containerization**
Hardware layer followed by Hypervisor to create virtualized instances of processors, RAM, network cards	Hardware interfaced to Host OS by the Kernel
Isolation of independent machines	Isolation of processes running in the same environment sharing host resources
Hardware virtualized by Hypervisor for Virtual Machines	Namespaces create the illusion for each container OS and Cgroups limit the resource usage so as to not tax the OS
Extremely flexibility of hardware configurations	High portability due to use of dockerfile

2. **Install and run Docker on your system**



3. **Follow the following video to create containers from images**
   https://youtu.be/FzwIs2jMESM

4. **Answer the following questions:**
5. **What are docker image, container, and registry?**

Docker Image: A docker image is created using a docker file that contains the instructions to build a docker image. An image acts as a template to create a docker container

Docker container: A container is a running instance of a docker image. They are built using templates created through docker images. Image can be considered as a class and container can be considered as an object of that class

Docker registry: A registry contains named docker images that can be stored and distributed

6. **List the Docker commands used in the video with a brief description for each command and option.**

   The following command is used to build the docker image and specify the -t option to add the tag along with the directory the docker file is located in. Upon finishing the Dockerfile, the user enters this command to build a docker image. This pulls the OpenJDK image provided in the docker file from docker hub

   ```
   docker build -t hello-world:2.0 .
   ```

   Get the list of Docker images using the following command

   ```
   docker images
   ```

   Run the docker image for your code using the following command

   ```
   docker run hello-world:2.0
   ```

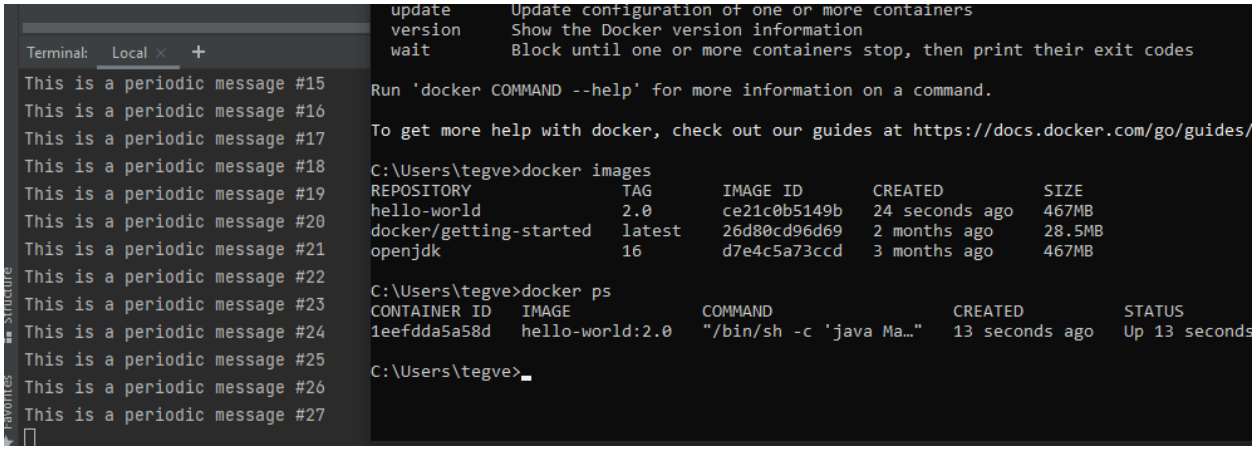
   Get the list of currently running containers using the following command. This will not show our hello-world image since our code prints hello world and terminates

   ```
   docker ps
   ```

   Get the list of all containers using the following command. The output of this command will include the hello-world docker image

   ```
   docker ps -a
   ```

   Following is a screenshot of output



Run the container in the background or detached mode using the following command

```
docker run -d hello-world:2.0
```

This video then proceeds to run more containers using the same docker image hello-world:2.0

Get a log of the container output using the following command

```
docker logs container_id
```

7. **At the end of the video, there are two running containers, what commands can be used to stop and delete those two containers?**

   Following is the sequence of steps to delete those containers

   Get the list of running containers:

   ```
   docker ps
   ```

   Stop the container:

   ```
   docker stop container_id
   ```

   Kill the container using the command:

   ```
   docker kill container container_id
   ```

   All stopped containers can be deleted using the following command:

   ```
   docker container prune
   ```

   Shorthand to completely delete a container whether running or not running is

   ```
   docker rm -f container_id
   ```

8. **Prepare a video showing the container(s) created on your machine, displaying their logs, stopping them, and then deleting them. (Note: the JDK version must match that installed in your machine and used to compile the java code. If you have a problem compiled it can download it from the repository from the path: "/v1/out/production/HelloWorldDocker/Main.class" and use OpenJDK:14 in your Dockerfile).**

   The link to the recorded video can be found here:

   https://drive.google.com/file/d/1b0rbgssuKb2wf6NXrY1dTY6uKRf_x5ID/view?usp=sharing

9. **Follow the following video to build a multi-container Docker application**

   https://youtu.be/_m9JYAvFB8s

10. **Answer the following questions:**
11. **What's a multi-container Docker application?**

    A multi-container docker application runs different containers providing different features as microservices. These features provide low coupling and lightweight processes. In our application, the app-db is connected with the web application using network bridges in multi-container application

12. **How these containers are communicated together?**

    The containers are connected together via a network bridge. Docker has multiple networks but to connect the 2 containers, a dedicated network is created called app-network and form a bridge between them

13. **What command can be used to stop the Docker application and delete its images?**

```
docker stop container_id
docker rm -f container_name
```

14. **List the new docker commands used in the video with a brief description for each command and option.**

    To pull the official MySQL default image from docker hub run the following command

```
docker pull mysql
```

Run the following command to create a container out of the mysql image
The --name option is followed by the name of the container
The -d option is specified to run the container in detached mode
The -e option specifies that it will be followed by a configuration environment variable. The first e flag is followed by the root password which is “tegveer”
Another environment variable is specified here with the database name mYDB

```
docker run --name app-db -d -e MYSQL_ROOT_PASSWORD=tegveer
-e MYSQL_DATABASE=myDB mysql
```

Docker allows the user to store data locally on the host OS using volumes

Run `mvn install` after navigating into the cloned repository to generate the war file. The output will look as follows

```
[INFO] Copying webapp resources [C:\Users\tegve\Java Spring Projects\v2\src\main\webapp]
[INFO] Building war: C:\Users\tegve\Java Spring Projects\v2\target\MyWebApp.war
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ MyWebApp ---
[INFO] Installing C:\Users\tegve\Java Spring Projects\v2\target\MyWebApp.war to C:\Users\tegve\.m2\repository\com\jetbrains\MyWebApp\1.0\MyWebApp-1.0.war
[INFO] Installing C:\Users\tegve\Java Spring Projects\v2\pom.xml to C:\Users\tegve\.m2\repository\com\jetbrains\MyWebApp\1.0\MyWebApp-1.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.259 s
[INFO] Finished at: 2022-02-03T01:42:10-05:00
[INFO] ------------------------------------------------------------------------
```

Create a docker image for the web application and create a container using that image by entering the following commands in the terminal

```
docker build -t my-web-app:1.0 .
docker run --name app -d my-web-app:1.0
```

Currently 2 different containers are running using Docker using 2 different images as templates as shown in the following image

```
C:\Users\tegve\Java Spring Projects\v2>docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                NAMES
c58782201ee3   my-web-app:1.0  "catalina.sh run"        3 seconds ago    Up 2 seconds    8080/tcp             app
854d4621be61   mysql           "docker-entrypoint.s…"   36 minutes ago   Up 36 minutes   3306/tcp, 33060/tcp  app-db
```

Make the following dockerfile

```
# Pull the specific version of tomcat

FROM tomcat:10-jdk16

# Add the application code from the host machine to the image file system
/user/local.. is where tomcat stores its information

ADD target/MyWebApp.war /usr/local/tomcat/webapps/MyWebApp.war
```

```
# Expose the port running the tomcat server within the container

EXPOSE 8080

# Invoke the catalina.sh script to run the tomcat server

CMD ["catalina.sh", "run"]
```

Following this, run a docker container publishing the port being exposed on the container using the following command

```
docker run --name app -d -p 8080:8080 my-web-app:1.0
```

Refer to the following screenshot for the running application

Create a dedicated bridge network between two containers and specify its name as app-network and List all the running networks using the following commands

```
docker network create app-network
docker network ls
```

Connect the database and the app to the dedicated bridge network using the following commands

```
docker network connect app-network app-db
docker network connect app-network app
```

Create the following docker-compose file

```yaml
version: "3"
services:
  app-db:
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=tegveer
      - MYSQL_DATABASE=myDB
  app:
    build: .
    ports:
      - "8080:8080"
    depends_on:
      - app-db
```

Run the container-system through the docker-compose file using the following command

```
docker-compose up -d
```

15. **Prepare a video showing the created application, run the webapp, stop the application and delete the application containers. (Note: if you have a problem generating the war file, you can download it from the repository from the path: "/v2/target/MyWebApp.war").**

    Refer to the following link for the video:

    https://drive.google.com/file/d/1dogxXCX3J0pmWlFaccG6a13YzhFcyyEQ/view?usp=sharing

16. **Create a free Google Cloud Account. The first two videos in the following playlist may be helpful**
    https://youtu.be/4D3X6Xl5c_Y?list=PLIivdWyY5sqKh1gDR0WpP9iIOY00IE0xL

17. **Watch the following videos to get familiar with Kubernetes:**
    https://youtu.be/Rl5M1CzgEH4

18. **Follow the following video to deploy dockers containers (valid until the shell session is expired) on GCP or by using Kubernetes (until you change it)**
    https://youtu.be/vIKy3pDz3jM

19. **Prepare a video showing how the container is deployed using Docker and Kubernetes in GCP.**

    Following is the link for the video on GKE:
    https://drive.google.com/file/d/1-tZsR1y_S9JSdzNZVzBAkISPb-wkS_9v/view?usp=sharing

20. **List all used GCP shell commands and their description in your report.**

    Create a Project and use that project in GCP. The following command is used to set the current project in the GCP terminal to be the required project cloud-computing-milestone

    ```
    gcloud config set project cloud-computing-milestone
    ```

    Run the nginx server (in detached version using -d) by exposing port 8080 of the cloud shell port 80 for the nginx web server using the following command

    ```
    docker run id -p 8080:80 nginx:latest
    ```

    Copy a sample index.html file onto the nginx server file system using the command

    ```
    docker cp index.html e2440ef1b0f1:/usr/share/nginx/html
    ```

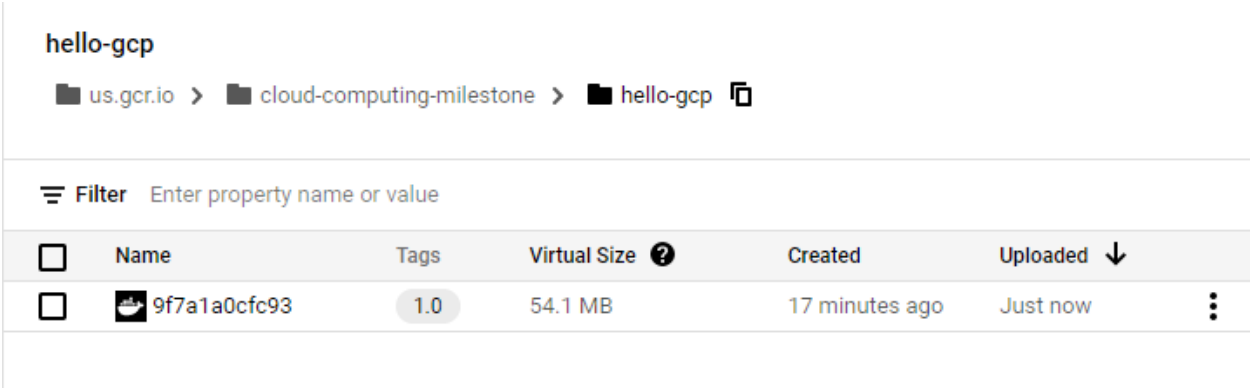
    Commit the newly added file changes to a newly created docker image using the following command

    ```
    docker commit e2440ef1b0f1 hello-gcp:1.0
    ```

    Tag the image and push it to the host using the following commands

    ```
    docker tag hello-gcp:1.0
    us.gcr.io/cloud-computing-milestone/hello-gcp:1.0
    docker push
    us.gcr.io/cloud-computing-milestone/hello-gcp:1.0
    ```

    Following is a screenshot of the image being successfully created in the container registry

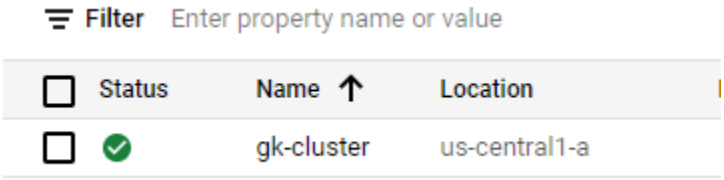


Set a default compute zone for GK cluster using the following command

```
gcloud config set compute/zone us-central1-a
```

After this create a Google Kubernetes GK Cluster using the following command. gk-cluster is the name of the cluster and the number of nodes is 1

```
gcloud container clusters create gk-cluster --num-nodes=1
```



The following command generates authentication credentials to deploy the container by provisioning kubectl which communicates with the API server

```
gcloud container clusters get-credentials gk-cluster
```

Deploy the application to the cluster. --image option specifies the container image

```
kubectl create deployment web-server
--image=us.gcr.io/cloud-computing-milestone/hello-gcp:1.0
```

Expose your application to the internet using the following command. Expose it with the name web-server and type load balancer to create a compute engine load balancer. --port specifies the public port and target-port specifies the Nginx port

```
kubectl expose deployment web-server --type LoadBalancer --port
80 --target-port 80
```

Display the status of the pods (Container groups)

```
kubectl get pods
```

```
kubectl get service web-server
```

21. **Prepare a Kubernetes YML (or YAML) file to load the webApp used in steps 6:8 and deploy it using the Kubernetes engine on GCP. The file is a little different than that used by docker-compose.**
    - **The hostname of all containers is the same and can be accessed by localhost, the address of the MySQL should be changed to localhost and recompiled. (Note: if you have a problem generating the war file, you can download it from the repository from the path "/KGS/target/MyWebApp.war").**
    - **Create a new image using the new war file and push it to Google Container Registry.**
    - **Follow the comments and fill the missing lines in the "/webApp.yml" file.**
    - **Apply the YML file into Kubernetes and run the server (what is the appropriate Cloud shell command?).**
22. **Prepare another video describing the YML file and showing how it's deployed on GCP.**

    Link to the video:
    https://drive.google.com/file/d/1-tZsR1y_S9JSdzNZVzBAkISPb-wkS_9v/view?usp=sharing

23. **Answer the following question:**
24. **What is Kubernetes' pod, service, node, and deployment?**

    Kubernetes pod: It is a unit managed entirely by Kubernetes and can contain one or more than one container
    Kubernetes service: Services provide a way to expose pods to the internet on the port specified by the kubectl command
    Kubernetes node: A node is a working entity to manage pods in Kubernetes. A node can have multiple pods
    Kubernetes deployment: Upon deployment, Kubernetes automatically manages schedules the pods that have multiple running containers

25. **What's meant by replicas?**

    Kubernetes replicas: A program that runs multiple pods so as to provide backup and increase scalability in the programs

26. **What are the types of Kubernetes' services? what is the purpose of each?**

Following are the different Kubernetes services

1. **ClusterIP:** Expose a service running within the cluster
2. **NodePort:** Exposure via individual node static ports
3. **LoadBalancer.** Expose a service through use of compute engine load balancer