Project Milestone-- IaaS: Virtualization and Containerization

- **Docker** is a software platform that allows you to run applications using containerization. A docker **container** is a standard component that allows you to package your application and its dependencies in an easy-to-share way. Some of the most important characteristics of a container is that they're portable, isolated, and lightweight. A docker **image** is what will be used to create your running container. It's basically a template which is used to create and run a container. In order to create an image, you need a Dockerfile which will include the instructions on how to build a docker image. A docker **registry** is like a bookshelf where images are stored and available to be pulled for building containers to run services or web apps.

- **FROM:** The first instruction of a Dockerfile specifies the base layer of your image using the FROM keyword then the name of the base image, i.e., "FROM openjdk:11" uses the OpenJDK 11 image as the base image.

- **RUN:** A dockerfile can have multiple RUN commands that allow you to execute commands. For example, it can be used to create a new app directory for your application files. The RUN keyword followed by the mkdir command with the directory name. Note that the directory will be created inside the container, not on the host machine.

- **COPY:** used to copy the application files from the host machine into the image filesystem. The COPY keyword is used followed by the specified file.

- **WORKDIR:** used to set the directory where future commands will be executed. This can be done by using the WORKDIR keyword then specifying the app directory.

- **CMD:** Every dockerfile needs to specify the command that will execute when you start a container from the image. The dockerfile must contain one CMD or ENTRYPOINT instruction specifying the default instruction for the container. For example, the CMD keyword can be used to run the main class when the container starts using CMD followed by java Main.

- **docker run:** used to instantiate an image and run a container based on that image. The command creates a docker container and runs it.

- **docker ps:** used to list the containers that are currently running.

- **docker ps -a:** used to list all containers, both running and stopped.

- The two running containers can be stopped by using "docker stop <Container_ID>" and deleted by using "docker rm <Container_ID>".

- **Video 1:** https://drive.google.com/file/d/1ukrMGIgMGRLEUi6FLZfQ8H9P15kjviSJ/view?usp=sharing

- A **multi-container docker application** is an application that integrates databases with the front-end so that they are easily managed and modified separately.

- Containers can communicate with each other when they are part of the same network. By default, docker creates a bridged virtual network that containers are connected to. Containers address each other by using the IP address assigned to them in the network.

- The docker application can be stopped using the commands mentioned earlier for stopping and removing a container and its images can be deleted using “docker image rm <image_name>”.

- **docker pull mysql**: used to pull the official mysql image available on DockerHub.

- **docker run --name app-db -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=myDB mysql:** used to create a container named app-db in detached mode.

- **docker build -t my-web-app:1.0 .:** used to build an image named my-web-app.

- **docker run --name app -d -p 8080:8080 my-web-app:1.0:** used to create a container based on the image above followed by the port on the host machine that is binded to the container port.

- **docker network create app-network:** used to create a network name app-network.

- **docker network ls:** used to show the list of networks existing.

- **docker network connect app-network app-db:** used to connect the app-db container to the app-network network.

- **Video 2:** https://drive.google.com/file/d/1_uls2hD1f_uYhCXzOQw-FAmfmcY05SJN/view?usp=sharing

- **Video 3:** https://drive.google.com/file/d/1efxK_K4yj5fdjmaVgBHsNmwP8D7tnsYv/view?usp=sharing

- **docker cp index.html 177d1128cac2:/usr/share/nginx/html/:** used to copy index.html file from my machine to the container folder at 177d1128cac2.

- **docker commit 177d1128cac2 cad/web:version1:** used to commit image to the container registry.

- **docker tag cad/web:version1 us.gcr.io/zippy-parity-340118/cad-site:version1:** used to create an image tag in the container registry.

- **docker push us.gcr.io/zippy-parity-340118/cad-site:version1:** used to push the image to the container registry.

- **gcloud config set compute/zone us-central1-a:** used to set the cloud instance's compute zone to us central.

- **gcloud container clusters create gk-cluster --num-nodes=1:** used to create a cluster for the running containers.

- **gcloud container clusters get-credentials gk-cluster:** used to fetch credentials for the running cluster.

- **kubectl create deployment web-server --image=us.gcr.io/zippy-parity-340118/cad-site:version1:** used to create a deployment named web-server using the image we created earlier in the container registry.

- **kubectl expose deployment web-server --type LoadBalancer --port 80 --target-port 80:** used to create a service for the web-server deployment that serves on port 80 and connects to the containers on port 80.

- **kubectl get pods:** used to show the list of current pods running.

- **kubectl get service web-server:** used to view information about a service named web-server.

- **Video 4 - Part 1:** https://drive.google.com/file/d/1aMMRSARqN2ikRLXg1N3H95nPhf4Gvfll/view?usp=sharing
- **Video 4 - Part 2:** https://drive.google.com/file/d/1oxvS-vhZf8-yyxnNyywZAek9Iyw420OP/view?usp=sharing

- **Pods** are the smallest deployable unit in Kubernetes. A Pod contains one or more containers and represents one instance of a running process in a cluster. When using multiple containers, Pods treat them as a single entity and the resources are shared among the containers.

- **Service** allows network access to a group of pods in a cluster that perform the same function. Pods are assigned a unique name and IP address through which they can communicate.

- **Node** is a virtual or physical machine inside a cluster that is used by Kubernetes to manage pods.

- **Deployment** is used to ensure that pods are running in a Kubernetes cluster. It tells Kubernetes how to create or update instances of these pods.

- **Replicas** are responsible for maintaining a set of replica Pods that are currently running in a cluster and ensures the availability of identical pods at any given time. It prevents users from losing access to their application when a Pod fails.

- The four types of Kubernetes' services are ClusterIP, NodePort, LoadBalancer, and ExternalName.

- **ClusterIP** is a default service type that provides network connectivity within your cluster and can only be accessed inside the cluster.

- **NodePort** is a service that can be accessed from outside the cluster by using <NodeIP>:<NodePort>. It is an extension of the ClusterIP service where ClusterIP will be created for each node with the node port.

- **LoadBalancer** is used when the service requires an external load balancer. The load balancer routes to the ClusterIP and NodePort services that are created.

- **ExternalName** is a service that uses DNS names instead of default names.