# SOFE 4360U- Cloud Computing

Project Milestone

IaaS: Virtualization and Containerization

Alex Dafoe 100599423

Ethan Bautista 100657250

Hamza Farhat Ali 100657374

Nick Kvrgic 100613386

Github Link: https://github.com/HamzaFarhat/CloudProject1Group8
Final Group 4 Videos link:
https://drive.google.com/drive/folders/1eKyWde8YIDOPRPaSd1NBX0AFHAXUFRJc?usp=sharing

**What are docker image, container, and registry?**

- *Docker image*
    - contains the files to run an application in a container
    - This docker image acts as a template of instructions to build the container
    - A docker image contains application code, libraries, tools, dependencies, and other files to run the application on the container.
- *Docker container*
    - is a virtualized runtime environment that makes it easier to run applications in different environments and is self contained.
    - This container instance is the operating system virtualization

- *Docker registry*
    - is a repository of all docker images that stores and distributes different versions of docker images.
    - It also holds the storage and content delivery system
    - Docker is the software package that allows one container to be built with all dependencies needed to run an application.
    - Instead of setting up the app server, and all other technologies beforehand to run an application.

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

- `docker build -t <tag> <path>`

- This command builds the Docker image
- -t option is to specify the name and optionally the version of the image

- `docker run [-d] <image>`

- This command runs the Docker container which means the code in the Docker image gets executed
- -d option is the running the container in detached mode which means the container will run in the background and not display anything in the console

- `docker ps [-a]`

- This command displays all currently running containers
- -a option shows all Docker containers including containers that already ran and terminated

- `docker images`

- This command displays all Docker images currently on the system. It displays information about the images such as the repository, tag, imageid, creation date and size.

- `docker logs <Container ID>`

- This command will show the console log of the container

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

- `docker stop <container name/ID>`
- This will stop the container from running
- `docker rm <Container name/ID>`
- This will remove the container. It will not show under directory

**What's a multi-container Docker application?**

- A multi-container Docker application is an application which consists of multiple containers usually managed by docker-compose.
- A multi-container application is usually used to host applications using the microservice architecture.
- This allows for multiple containers to communicate with each other instead of isolation
- An example of this would be a web application with a database. In this situation, there will be two containers. One for the web application and another container for the database.

**How do these containers communicate together?**

- The containers communicate with each other through a bridge network.
- This virtual bridge by default connects to all containers on the cloud instance.
- These containers are assigned IP addresses to communicate with each other.
- Lastly, for more control a user can create a defined bridge which gives that one docker more control for priorities to control other container ports.

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

- `docker-compose down -rmi all`

RMI refers to registry, deletes content

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

`- docker pull <image name>`

- Pulls an image from a registry

`- docker run -name <Name> -d -e <var_name=var_value> -p <host machine port:container port>`

- This command runs the Docker container
- -name option specifies a name of the container
- -e option sets the environmental variables of the container
- -p option publishes the containers port to the host

`- docker network create <name>`

- This command creates a bridge network for the containers to communicate on

`- docker network connect <network name> <container name>`

- This command creates a container to a network

`- docker network ls`

- This command lists all networks the docker daemon knows about

`- docker-compose up`

- This command initializes the containers defined in the docker-compose.yml file

**List all used GCP shell commands and their description in your report (new commands only).**

`- gcloud config set project <project name>`

- This command sets the project being used in the cloud console

`- docker cp <filename> <container-id>:<container-path>`

- This command copies files from the local filesystem to the container

`- docker commit <container id> <img-name>`

- This command creates a new image from the specified container
- Used for when creating images when changes to the container has been made

- `docker tag <source-img-name> <img-name>`

  - This command creates another image that refers to a source image

- `docker push <img-name>`

  - This command pushes the image to the container registry

- `gcloud config set compute/zone <region>`

  - This command changes the region of the project to the region specified

- `gcloud container clusters create <cluster-name> --num-nodes=<num-nodes>`

  - This command creates a cluster for containers with the specified name
  - –num-nodes option just lets the user specify the number of nodes to be created in the cluster

- `gcloud container clusters get-credentials <cluster-name>`

  - This command retrieves the credentials for a running cluster
  - This updates the kubeconfig file with the credentials of the cluster

- `kubectl create deployment <deployment-name> --image=<img-name>`

  - This command creates a deployment with the specified name and image

- `kubectl expose deployment <deployment-name> --type <service> --port <port> --target-port <port>`

  - This command exposes the deployment with the specified name
  - --type option allows the user to specify what type of kubernetes service like for example a LoadBalancer

- `kubectl get pods`

  - This command lists all pods

- `kubectl get service <deployment-name>`

  - This command lists the service with the specified name.
  - If used without the deployment name, it will list all services

**Apply the YML file into Kubernetes and run the server (what is the appropriate Cloud shell command?)**

- `kubectl apply -f webApp.yml`

**What is Kubernetes' pod, service, node, and deployment?**

- Pod
    - is a group of one or more containers with shared storage, scripts to run containers and network resources that run on a physical/logical host.
- Service
    - A way to expose an application(s) interface(s) running on a set of pods as a network service.
    - E.g., use a load balancer to direct traffic to containers running in the pods. ClusterIP, NodePort and ExternalName are additional ways one can expose an application's interface.
- Node
    - Physical or virtual machine where pods run
- Deployment
    - This provides the updates when describing the code in production.
    - This desired state in a deployment changes the current cloud state in accordance to deployment .
    - It is used to update the pods, replicas and whatever other dependencies in kubernetes

**What's meant by replicas?**

- Multiple instances of identical pods. Used to improve availability of a service.
- A replica is defined with replicaSets as well to control for a specified number of pod replicas running together.

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

- ClusterIP
    - This is the default type of service that exposes the service on a cluster-internal IP.
    - This provides network connectivity between containers
- NodePort
    - Exposes a service via a static port via each node's IP
    - This routes incoming traffic from NodePort to user's service, even when on a different node
- LoadBalancer
    - Exposes the service via a load balancer
    - The traffic gets integrated with the native cloud-based balancers when using a cloud balancer as well.
    - Integrated NodePort and load balancers for this scenario.
- ExternalName

- Similar to other services but can be accessed with DNS name instead of the clusterIP address
- This Extername help’s serve as a way to return an alias to an external service residing outside the cluster, for security purposes.