Cloud Computing

Project Milestone 1: IaaS Virtualization and Containerization

2/02/2022

Danial Asghar

100671850

Docker image: The Docker image is an immutable file that contains the source code, libraries, dependencies, tools, and all files required for a packaged application to run. They act as a set of instructions from which a Docker Container is built.

Docker container: A Docker container is a virtualized and sandboxed runtime environment used in application development. The containers are portable and compact, allowing to be started and stopped quickly. A container requires and depends upon a runnable Docker image as the container provides the runtime environment and the image provides the application to execute.

Docker registry: A registry is a stateless and highly scalable server-side application which provides storage and content delivery functionalities. The Docker Hub, Docker's registry, allowing users to store and distribute Docker images.

Video 1: Intro to Docker

The following lists the commands and their respective explanations from Video 1:

- `docker build -t helloworld:1.0 .`
  - This command builds Docker images from a Dockerfile and a context.
  - The -t parameter allows you to set the image name "helloworld" and the release tag "1.0"
  - The . at the end specifies the file where the Dockerfile is located
- `docker images`
  - This command allows you to see a list of all top-level images, their repository and tags, and their size.
- `docker run helloworld:1.0`
  - This command instantiates a container and runs the image "helloworld:1.0" using the specified entry point command.
- `docker ps`
  - Lists a list of currently running containers.
- `docker ps -a`
  - Lists all containers, running and stopped.
- `docker run -d helloworld:2.0`
  - Runs the container in detached mode, which allows the container to execute in the background and doesn't lock the terminal during execution.
- `docker logs fdfb47977acD`
  - This command batch retrieves the container's logs, identified by the container id, present at the time of execution.

The following command is used to stop a Docker container:

```
docker stop containerID
```

The following command is used to delete a Docker container:

```
docker rm containerID
```

My video 1 link is:
https://drive.google.com/file/d/1JtiLV35QIu11C8HPpdWysyNf2asCoMBH/view?usp=sharing

Multi-container Docker applications contain more than one container which have separate components required for the full application functionality. Modern, modular applications do not run in a single monolithic component. Within Docker each container should do one thing. With respect to the provided web app, one container was for MySQL and another for the Web App. These two containers communicate with one another using bridge networks.

Communications between containers within Docker are done using bridge networks. Docker uses a software bridge for bridge networks, allowing containers connected to the same bridge communicate with each other, while still isolating from containers not connected to the bridge.

The following command can be used to stop and delete a container:

```
docker rm -f app
```

The following command is used to delete an image:

```
docker rmi Image
```

Video 2: Intro to Docker 2 (Networking, Docker Compose)

The following lists the commands and descriptions presented in Video 2:

- `docker pull mysql`
  - Pulls an image or a repository from a registry, here the MySQL image is being pulled.
- `docker run --name app-db -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=myDB mysql`
  - Creates a container from the MySQL image, gives it the name “app-db”, passes in the root password using an environment variable, and sets up a default database using the environment variable.
- `docker rm -f app`
  - Stops and deletes the “app” container.
- `docker run --name app -d -p 8080:8080 my-web-app:1.0`
  - Starts a container with the name “app” in detached mode using the “my-web-app:1.0” image and binds the host port 8080 to the container port 8080.
- `docker network create app-network`
  - Creates a new bridge network to handle communication between containers

- `docker network ls`
  - Lists all the networks
- `docker network connect app-network app-db`
  - Connects the "app-db" app to the network
- `docker-compose up -d`
  - Builds, recreates, starts, and attaches to containers for a service. This command will also start any linked services.

My video 2 link is the following:
https://drive.google.com/file/d/1PdVGW4W1pMrYg6MuzQTtYK_h_sxk63V1/view?usp=sharing

Video 3: Google Kubernetes Engine: Create GKE Cluster and Deploy Website

The following video shows the steps to deploy using Docker and Kubernetes in GCP:
https://drive.google.com/file/d/175W-xvg7DtmmsE1UwCSSjxlXnKfyYl_t/view?usp=sharing

The follow contains the list of commands and their respective descriptions in Video 3:

- `gcloud config set project youtube-demo-255723`
  - Sets the project ID in the GCloud shell
- `docker run -d -p 8080:8080 nginx:latest`
  - Starts a container using the "nginx:latest" image from repository
- `docker cp index.html a2074e7675b3:/usr/share/nginx/html/`
  - Copies the sample index.html into the nginx server's folder within the container.
- `docker commit a2074e7675b3 cad/web:version1`
  - Creates a new image from the container's changes
- `docker tag cad/web:version1 us.gcr.io/cloudassignment1part3/cad-site:version1`
  - Creates a target image referring to a source image.
- `docker push us.gcr.io/cloudassignment1part3/cad-site:version1`
  - Pushes the tagged image to the Google Cloud Registry at the specified folder location.
- `gcloud config set compute/zone northamerica-northeast2-a`
  - Sets the default region and zone in the local client to Toronto servers
- `gcloud container clusters create gk-cluster –num-nodes=1`
  - Creates a GKE cluster for running containers, with one node.
- `gcloud container clusters get-credentials gk-cluster`
  - Fetches the credentials of the running cluster so that they can be used in kubectl.
- `kubectl create deployment web-server --image= us.gcr.io/cloudassignment1part3/cad-site:version1`
  - Creates a new deployment called "web-server" from the GCR image specified.

- `kubectl expose deployment web-server --type LoadBalancer --port 80 --target-port 80`
    - Creates a service for the web-server deployment, which serves on port 80 and connects to the containers on port 80
- `kubectl get pods`
    - Lists all the pods in the namespace and their status
- `kubectl get service web-server`
    - Lists all the services in the namespace and their internal/external IP addresses.

The following video shows how to deploy a container using a Kubernetes YAML file on Google Cloud Platform: https://drive.google.com/file/d/1FGows6H_wQKA01x6nbKhTlwA4bA2C-6W/view?usp=sharing

Kubernetes Pods: Kubernetes pods are the smallest deployment unit of execution which can be created or managed by Kubernetes. A pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

Kubernetes Service: Kubernetes services are a way to expose applications running on Pods as a network service. It defines a logical set of Pods and a policy by how they are accessible.

Kubernetes Node: Nodes are what Pods run upon. Kubernetes runs the workload by placing containers into Pods which run on top of Nodes. This node may be a virtual or physical machine, depending on the cluster, and is managed by the control plane.

Kubernetes Deployment: A Kubernetes Deployment represents a set of multiple identical Pods with no unique identities. The Deployment runs multiple replicas of the application and monitors health so that any Pods which fail get replaced. Deployments use a Pod template with a specification containing: what applications should run inside its containers, which volumes to mount upon, labels, etc.

Replicas: Replicas are just multiple numbers of pods of the same application executing. Kubernetes uses the ReplicaSet to ensure a specified number of pod replicas are running at a given time. Kubernetes schedules enough pods to meet the minimum availability defined.

Types of Kubernetes services: The following list include the different types of Kubernetes services and their explanation:

- ClusterIP: Exposes a service which is only accessible from within the cluster. This is the default type of service which exposes a service on an IP address internal to the cluster.
- NodePort: Exposes a service via a static port on each node's IP. NodePorts exist on every cluster and Kubernetes routes traffic coming to a NodePort to the service instead.

- LoadBalancer: Exposes a service via the cloud provider's load balancer. Using the LoadBalancer service gives a ClusterIP service but extends it to an external load balancer provided by the cloud provider.
- ExternalName: Maps a service to a predefined externalName field by returning a value for the CNAME record. This service is similar to the other types but instead it returns a CNAME record with a value which is used to access the service.