Distributed Systems: Lab 1
Introduction to Google Kubernetes Engine (GKE)

Due: Sep 18, 2022

Ridwan Hossain (100747897)

Discussion

Docker:

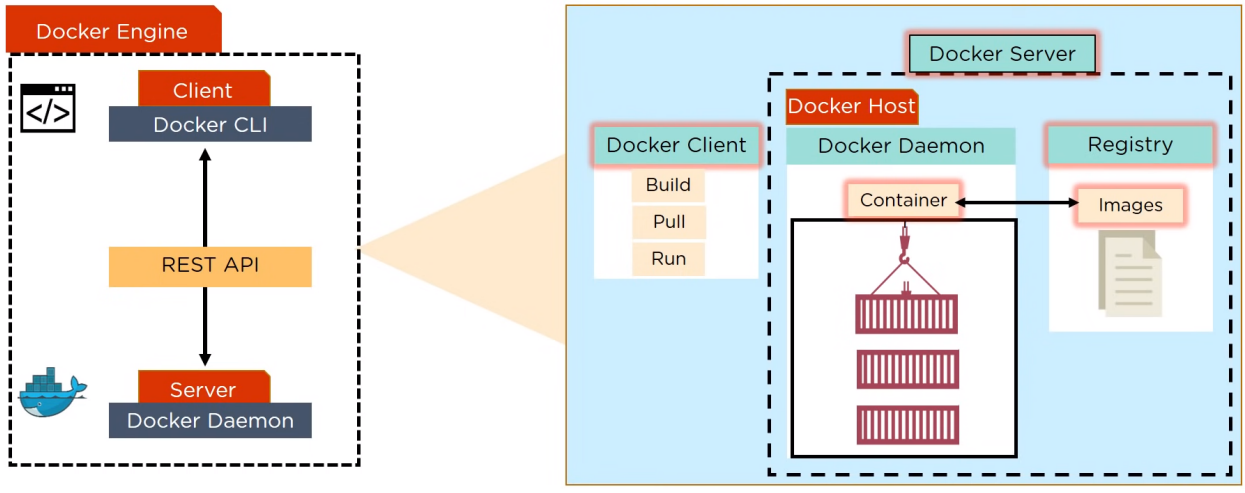
All images from this section have been captured from the following video: https://www.youtube.com/watch?v=rOTqprHv1YE

Docker is essentially a software delivery system which uses virtualization to containerize applications and their dependencies. This makes the deployment of software packages much more easy and streamlined, as it eliminates errors where different users may have differing operating systems, libraries, dependencies, etc.

In a traditional virtual machine approach, all applications and libraries sit on top of a guest operating system within the virtual machine, with each operating system having its own resources. This results in VMs occupying much more space, running much slower, having compatibility issues, inhibiting limited scalability and having restricted resources.

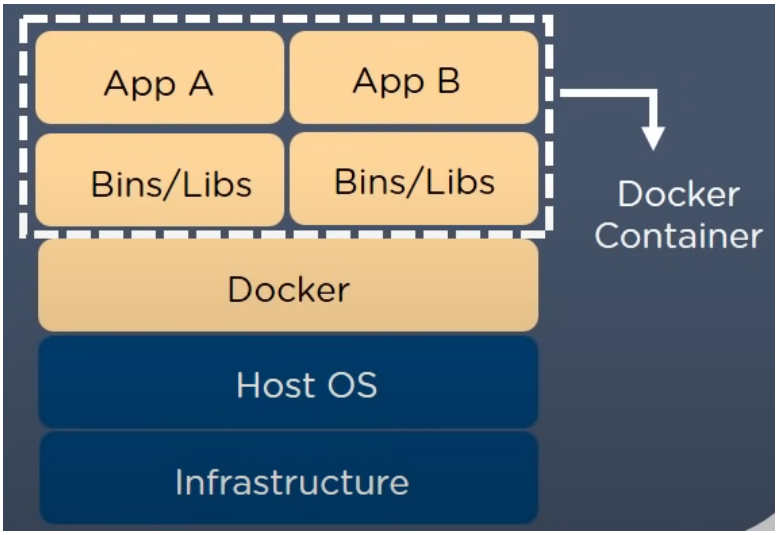
Docker eliminates the use of guest operating systems and is inherently more lightweight as a result. Docker's container approach provides the same functionality as virtual machines while removing most of the cons associated with it. Docker containers occupy less space, have better performance, are highly scalable, are easily portable, and have sharable resources.

To understand why docker is such an improvement relative to the virtual machine approach, it is important to examine how docker functions. The following diagram will aid in understanding:



Docker engine is installed on the host machine and is used to build and run containers. It utilizes a client-server architecture, where the client may communicate with a REST API using the docker command line interface. The commands sent through the API are sent to the docker daemon, which is a server which takes client requests and manages containers. Some instructions could be build instructions, pull instructions, and so on.

Docker images are essentially templates with instructions on how to create docker containers. Docker images are built using *docker files*, which are files with commands on how to build a docker image. A *docker container* is a standalone, executable software package that contains applications along with all the dependencies required to run each application, where each application runs in isolation. This can be visualized by the diagram below:



Multiple docker containers are able to be run on the same infrastructure and may share the same operating system and its resources.

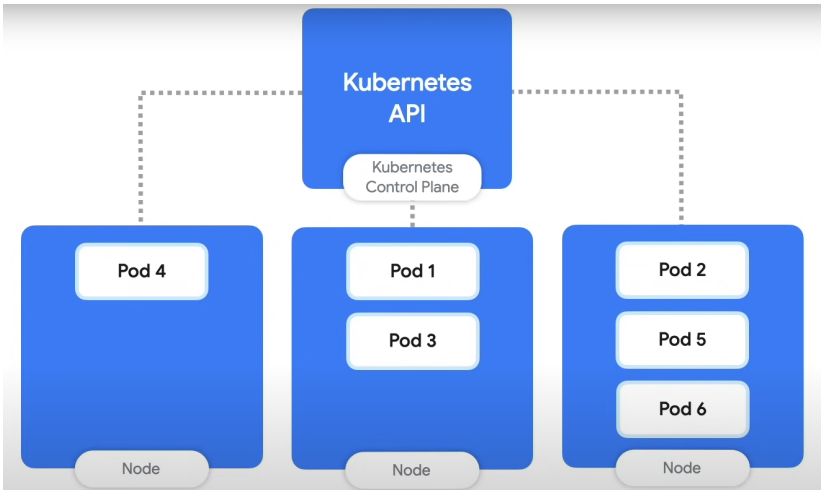
A *docker registry* is a service used to host and distribute images. Docker has its own default registry, known as *docker hub*.

Kubernetes:

All images from this section have been captured from the following video: https://www.youtube.com/watch?v=cC46cg5FFAM

Kubernetes is essentially a container management and orchestration platform which can be used to ensure that containers are where they're supposed to be, doing what they're supposed to, and that containers can work together.

Kubernetes facilitates excellent management of containerized workloads and services in an automated manner. This allows distributed systems to be run efficiently. Kubernetes focuses on managing software containers using an architecture based on **nodes**. This can be shown using the diagram below:



Each node consists of **pods**, where a pod is the smallest execution unit in kubernetes. A pod essentially encapsulates one or more applications. The nodes in containers can be grouped into **clusters**, where each container has endpoints, DNS, storage, and scalability.

While this orchestration could be done manually, kubernetes automates this process and lets the developer tell kubernetes what they want the cluster to look like.

Design

Some assistance was acquired from the following link to aid with creating a MongoDB deployment in the Google Kubernetes Engine: https://devopscube.com/deploy-mongodb-kubernetes/.

Similar to the mysql.yaml file, the mongodb.yaml file was broken into 2 sections. One section includes the deployment parameters for the MongoDB deployment. It was given the name “mongodb-deployment” and the replica count is set to 1, indicating that 1 pod will be created from the docker image. The template field contains environment variables as defined in the docker image documentation for the official mongo image, which can be found on docker hub: https://hub.docker.com/_/mongo.

The other section was for exposing a load balancer service for the deployment, similar to the mysql.yaml file. It was given the name “mongo-service” and was specified with the service port number 27017, which is the default port used by MongoDB. When the deployment is applied using the mongodb.yaml file, the deployment is given an external IP address. This IP address can be used to connect to the deployment from other machines.