DOCKER zzak00

Introduction to Containerization

According to IBM,
Containerization involves encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure.

How to avoid *"but, it runs on my machine"*?

1. Develop and run the application inside an isolated environment (known as a container) that matches your final deployment environment.

2. Put the application inside a single file (known as an image) along with all its dependencies and necessary deployment configurations.

3. And share that image through a central server (known as a registry) that is accessible by anyone with proper authorization.

What role does Docker play ?

Docker is an implementation of containerization idea. It's an open-source platform that allows to containerize applications, share them using public or private registries, and also to orchestrate them.

Docker Container

A container is an abstraction at the application layer that packages code and dependencies together. Instead of virtualizing the entire physical machine, containers virtualize the host operating system only.

Containers vs Virtual Machines

Containers and virtual machines are actually different ways of virtualizing your physical hardware. The main difference between these two is the method of virtualization. Virtual machines are usually created and managed by a program known as a hypervisor, like Oracle VM VirtualBox, VMware Workstation, KVM, Microsoft Hyper-V and so on. This hypervisor program usually sits between the host operating system and the virtual machines to act as a medium of communication.
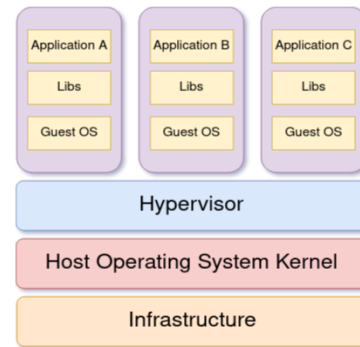
Each virtual machine comes with its own guest operating system which is just as heavy as the host operating system. The application running inside a virtual machine communicates with the guest operating system, which talks to the hypervisor, which then in turn talks to the host operating system to allocate necessary resources from the physical infrastructure to the running application.

As you can see, there is a long chain of communication between applications running inside virtual machines and the physical infrastructure. The application running inside the virtual machine may take only a small amount of resources, but the guest operating system adds a noticeable overhead. Unlike a virtual machine, a container does the job of virtualization in a smarter way. Instead of having a complete guest operating system inside a container, it just utilizes the host operating system via the container runtime while maintaining isolation – just like a traditional virtual machine.
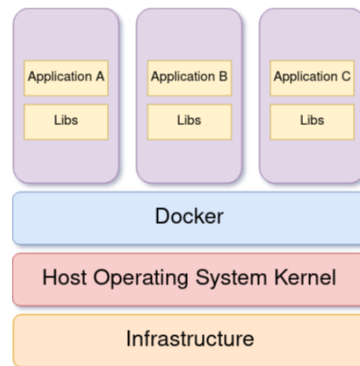
The container runtime, that is Docker, sits between the containers and the host operating system instead of a hypervisor. The containers then communicate with the container runtime which then communicates with the host

operating system to get necessary resources from the physical infrastructure.

As a result of eliminating the entire guest operating system layer, containers are much lighter and less resource-hogging than traditional virtual machines. *Containers virtualize the host operating system instead of having an operating system of their own.*

Docker Image

Images are multi-layered self-contained files that act as the template for creating containers. They are like a frozen, read-only copy of a container. Images can be exchanged through registries. In the past, different container engines had different image formats. But later on, the Open Container Initiative (OCI) defined a standard specification for container images which is complied by the major containerization engines out there. This means that an image built with Docker can be used with another runtime like Podman without any additional hassle.

Containers are just images in running state. When you obtain an image from the internet and run a container using that image, you essentially create another temporary writable layer on top of the previous read-only ones.

Docker Registry

An image registry is a centralized place where you can upload your images and can also download images created by others. Docker Hub is the default public registry for Docker. Another very popular image registry is Quay by Red Hat. Apart from Docker Hub or Quay, you can also create your own image registry for hosting private images. There is also a local registry that runs within your computer that caches images pulled from remote registries.

Docker Architecture

1. Docker Daemon: The daemon (dockerd) is a process that keeps running in the background and waits for commands from the client. The daemon is capable of managing various Docker objects.

2. Docker Client: The client (docker) is a command-line interface program mostly responsible for transporting commands issued by users.

3. REST API: The REST API acts as a bridge between the daemon and the client. Any command issued using the client passes through the API to finally reach the daemon.

According to the official docs,
"Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers".

Demonstration
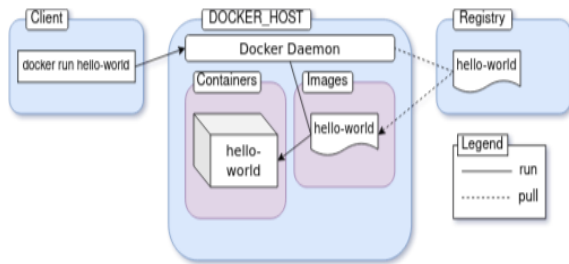
1. You execute *docker run hello-world* command where *hello-world* is the name of an image.

2. Docker client reaches out to the daemon, tells it to get the *hello-world* image and run a container from that.

3. Docker daemon looks for the image within your local repository and realizes that it's not there, resulting in the *Unable to find image 'hello-world:latest' locally* that's printed on your terminal.

4. The daemon then reaches out to the default public registry which is Docker Hub and pulls in the latest copy of the *hello-world* image, indicated by the *latest: Pulling from library/hello-world* line in your terminal.

5. Docker daemon then creates a new container from the freshly pulled image.

6. Finally Docker daemon runs the container created using the *hello-world* image outputting the wall of text on your terminal.

Docker Container Manipulation

Run a Container
```
docker <object> <command> <options>
```

In this syntax:

- *object* indicates the type of Docker object you'll be manipulating. This can be a container, image, network or volume object.

- *command* indicates the task to be carried out by the daemon, that is the run command.

- *options* can be any valid parameter that can override the default behavior of the command, like the –publish option for port mapping.

Publish Port

Containers are isolated environments. Your host system doesn't know anything about what's going on inside a container. Hence, applications running inside a container remain inaccessible from the outside.

To allow access from outside of a container, you must publish the appropriate port inside the container to a port on your local network.

```
--publish or -p <host port>:<container port>
```

8080:80 means any request sent to port 8080 of your host system will be forwarded to port 80 inside the container.

Use Detached Mode

Closing the terminal window also stopped the running container.

This is because, by default, containers run in the foreground and attach themselves to the terminal like any other normal program invoked from the terminal.

In order to override this behavior and keep a container running in background, you can include the –detach option with the run command.

```
docker container run --detach --publish
<local Network port>:<Container port> <image name>
```

List Containers
```
docker container ls -all
```

Name or Rename a Container
By default, every container has two identifiers. They are as follows:

- CONTAINER ID - a random 64 character-long string.

- NAME - combination of two random words, joined with an underscore.

```
docker container run --detach --publish <local Network port>:
<Container port> --name <name> <image name>
docker container rename <container identifier> <new name>
```

Stop or Kill a Container
The stop command shuts down a container gracefully by sending a *SIGTERM* signal. If the container doesn't stop within a certain period, a *SIGKILL* signal is sent which shuts down the container immediately.
In cases where you want to send a *SIGKILL* signal instead of a *SIGTERM* signal, you may use the container kill

```
docker container stop <container identifier>
docker container kill <container identifier>
```

Start or Restart a Container
```
docker container start <container identifier>
docker container restart <container identifier>
```

Create a Container Without Running
Container run command which is in reality a combination of two separate commands. These commands are as follows:

- container create command creates a container from a given image.

- container start command starts a container that has been already created.

```
docker container create --publish <local Network port>:
<Container port> --name <name> <image name>
```

Remove Dangling Containers
```
docker container rm <container identifier>
docker container prune
```