First Alpine Linux Containers

1. Running your first container

```
docker container run hello-world
```

This command try to find **hello-world** image, but we don't have this image locally. So the Docker will find it in default **Docker registry**. If find image there, it will pull it down, then run container.

Differences between Container and VM

VM is a hardware abstraction

Container is an application abstraction

1.1 Docker Images

Alpine Linux is a lightweight Linux distribution. And it's easy to pull down and run.

The **pull** command fetches the image from the **Docker registry** and saves it in our system.

```
docker image pull alpine
```

This command list all images in our system.

```
docker image ls
```

To run our **alpine** image, we need use **docker container run** command.

```
docker container run alpine ls -l
```

When we call **run**, the Docker client finds the **alpine image**, creates the container and then runs a command **ls -l** in that container.

If we try this command:

```
docker container run alpine echo "hello from alpine"
```

The Docker client will run **echo** command inside our alpine container then exited. This time, the command executed very fast.

If we try **echo** in VM, it's much slower. Because VM need to emulate full hardware stack, boot an operating system, and then launch. But Docker containers at application layer will skip most of steps. So that's the reason container is fast.

If we try this command:

```
docker container run alpine /bin/sh
```

Nothing happened! Because we didn't supply any additional commands to **/bin/sh**, so container just launched the shell, exited the shell, and then stopped the container. If we expected was an interactice shell, we need this:

```
docer container run -it alpine /bin/sh
```

After we already in Linux shell we can try few Linux commands. To exit to shell and container we can use **exit** command.

To check to container instances which are running, we use this command:

```
docker container ls
```

To check to container instances that ran, we use this command:

```
docker container ls -a
```

And the STATUS will show the when we exited these containers.

To get the flags of **run** command support:

```
docker container run --help
```

1.2 Container Isolation

In previous steps, we ran serveral containers which used same **alpine image**. But each execution was a seperate, isolated container. Each container has a seperate filesystem and runs in a different namespace. To learn more about isolation:

```
docker container run -it alpine /bin/ash
```

The **/bin/ash** is another type of shell available in alpine image. Once the container launches and we are at container's command prompt.

Type the command:

```
echo "hello world" > hello.txt
ls
```

The first **echo** command creates a file called **hello.txt** with the words **hello word** inside. The second command gives us a directory listing of the files and should how **hello.txt** file. Now **exit** this container.

Then run:

```
docker container run alpine ls
```

It is same **ls** command, but we can't see **hello.txt** file this time. Because the containers are isolation. The 2nd instance has no way to interact with 1st instance.

Isolation allows users to quickly create seperate, isolated test copies of an application or service and have them run side-by-side without interfering with one another.

To get back **hello.txt** file:

```
docker container ls -a
```

The container in which we created the **hello.txt** file is the same one where we used the **/bin/ash** shell. Then we can know the container ID:

```
docker container start <container ID>
```

Instead of using the full container ID, we can use just few characters, as long as they are enough to uniquely ID a container.

We used the ash shell this time so the rather than simply exiting the way **/bin/sh** did eariler, ash wait for command.

```
docker container exec <container ID> ls
```

This time we will get **hello.txt** file.

1.3 Terminology

Images

The file system and configuration of out application which are used to create containers. To find out more about a Docker image, run **docker image inspect alpine**.

Containers

Running instances of Docker images - containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS.

Docker daemon

The background service running on the host that manages building, runing and distributing Docker containers.

Docker client

The command line tool that allows the user to interact with the Docker daemon.

Docker Store

Store is, among other things, a. registry of Docker images. You can think of the registry as a directory of all available Docker images.