Docker Commands

Module Overview

- Container Commands
  - Start, stop, list, remove containers
- Container Information
  - Inspect, exec into a running container, view container logs
- Networking in Container
- Container Anatomy
- Docker Images

Container Commands

* List active containers
  * *docker container ls*
* List all containers
  * *docker container ls –a*
* Create container
  * *docker container run –p80:80 nginx*
* Stop Container
  * *docker container stop $Instance_ID*
* Start a container
  * *docker container start $Instance_ID*
* Remove Container
  * *docker container rm $Instance_ID*
* Inspect Container
  * *docker container inspect $Instance_ID*
  * Get a subsection in JSON format : *docker container inspect –f "{{json .Config}} " $Instance_ID*
  * Exec into a running container
    * *docker container exec –i -t $Instance_ID /bin/sh*
    * *apt-get update*
    * *apt-get install iputils-ping*
  * *Container logs*
    * *docker container logs $Instance_ID*
    * *docker container logs –tail 5 --follow $Instance_ID*
  * List Container Processes
    * *docker container top $Instance_ID*

Container Anatomy

- What is a container ?
  - Containers are specially encapsulated and secured processes running on the host system.
- Containers leverage a lot of features and primitives available in the Linux OS. The most important ones are :
  - **namespaces**
    - A namespace is an abstraction of global resources such as filesystems, network access, process tree (also named PID namespace) or the system group IDs, and user IDs
    - namespaces provide a layer of isolation
  - **cgroups**
    - Linux cgroups are used to limit, manage, and isolate resource usage of collections of processes running on a system. Resources are CPU time, system memory, network bandwidth, or combinations of these resources, and so on
    - Using cgroups, administrators can limit the resources that containers can consume
  - **UFS (Union File Systems)**
    - file systems that operate by creating layers, making them very lightweight and fast

Container Networking

How does containers communicate with each other ?

Network types a container uses:

- Bridge
  - Simple network based on Linux bridges allowing networking on a single host
  - Unless otherwise specified, newly created containers connect with default bridge network
- Macvlan
  - Use the macvlan network driver to assign a MAC address to each container’s virtual network interface, making it appear to be a physical network interface directly connected to the physical network
- Overlay
  - Creates a distributed network among multiple Docker daemon hosts
- Host
  - Container’s network stack is not isolated from the Docker host
  - Container does not get its own IP-address allocated
- None
  - Completely disables the networking stack on a container

Container Networking - Commands

```
docker network ls

docker network inspect bridge

docker network create my_app_net

docker network create —help

docker network inspect my_app_net

docker container run -d --name new_nginx --network my_app_net nginx

docker container run -d --name my_nginx --network my_app_net nginx

docker run --rm --network my_app_net busybox ping new_nginx

docker container run --detach --name nginx1 nginx

docker network connect my_app_net nginx1
docker container inspect nginx1

docker network disconnect my_app_net nginx1
docker container inspect nginx1
```

Docker Image

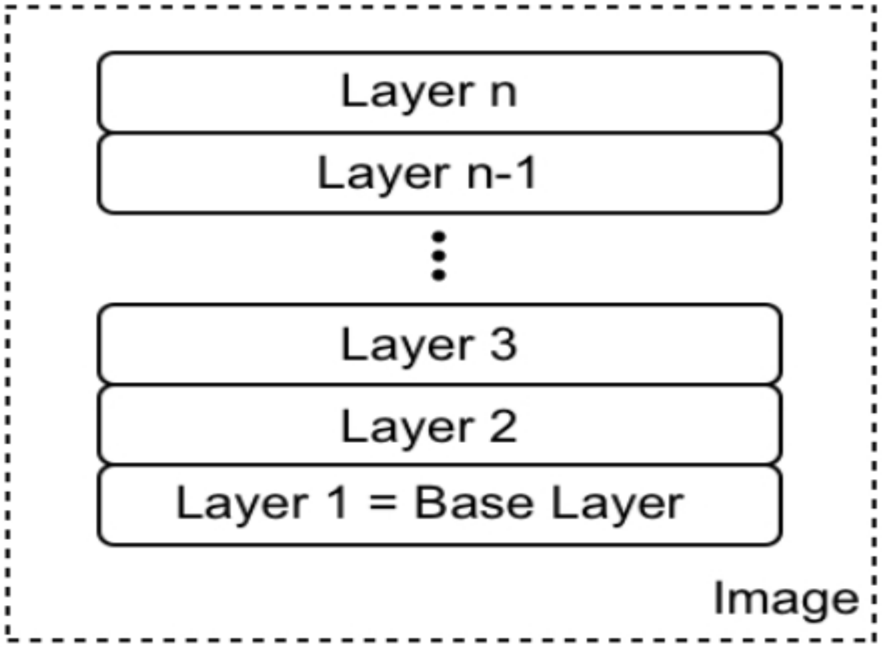
- A Docker image is containing everything needed to run an application as a container. This includes: code, runtime, libraries, environment variables, configuration files and image metadata
- An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime.
- Its not a complete OS

Creating Images

- Dockerfile
- Or committing container changes back to an image
- Stored In Docker Engine Image Cache
- Store permanently in Image Repository
- Docker Hub

Layered Filesystem

- Container images are templates from which containers are created. These images are not just one monolithic block, but are composed of many layers. The first layer in the image is also called the base layer



Create Image using commit

* The first way we can create a custom image is by interactively building a container. That is, we start with a base image that we want to use as a template and run a container of it interactively. Let's say that this is the alpine image. The command to run the container would then be as follows:
  * *$ docker container run -it --name sample alpine /bin/sh*
* By default, the alpine container does not have the ping tool installed. Let's assume we want to create a new custom image that has ping installed
  * *$ apk update && apk add iputils*
  * This uses the Alpine package manager apk to install the iputils library, of which ping is a part
* Use ping command as below:
  * *$ ping 127.0.0.1*
* Once we have finished our customization, we can quit the container by typing *exit* at the prompt
* List all containers
  * *$ docker container ls -a*
* Check what has changed in our container in relation to the base image
  * *$ docker container diff sample*
* Commit to persist changes
  * *$ docker container commit sample <<docker hub account id>>/my-alpine*
* List images
  * *docker image ls*
* How the image has been built
  * *docker image history my-alpine*
* Login to docker hub and push image
  * *docker login*
  * *docker push <<docker hub account id>>/my-alpine*

DockerFile

- The Dockerfile is a text file that is usually literally called Dockerfile. It contains instructions on how to build a custom container image. It is a declarative way of building images.

Dockerfile basics

- FROM (base image)
- ENV (environment variable)
- RUN (any arbitrary shell command)
- EXPOSE (open port from container to virtual network)
- CMD (command to run when container starts)
- docker image build (create image from Dockerfile)
  - `docker build –t <<>docker hub account>/devops_microservices –f <<Dockerfile name>> .`

Docker Hub

- Docker Hub, default docker registry
- Official images and how to use them
- How to discern "good" public images
- The recommended tagging scheme used by Official images
- Pushing custom images to hub

Quiz!!!

- If you wanted to view running containers as well as containers that you have stopped and not removed, what command would you use?
- What does the -d flag do in a docker run command?
- I ran 'docker container run -p 80:80 nginx' and my command line is gone and everything looks frozen. Why?
- Would the following two commands create a port conflict error with each other?
  docker container run -p 80:80 -d nginx
  docker container run -p 8080:80 -d nginx

Lab

* Create account on Docker Hub
* Create your own custom image
* Publish Image in docker hub