Author: Shivakumar Suresh *Date: June-13-2023*
Github: https://github.com/shivakumar96

Docker Quick Reference Guide

Author: Shivakumar Suresh

This is a Docker quick reference document for the people who want to start and understand about Docker and container technology. Let's learn about Docker by going through the following questionnaire.

1. What is VM?
   - A virtual Machine is a software, which simulates the environment on top of a different environment, without using any additional hardware using virtualization. Hypervisor, also known as Virtual Machine monitor, can be used to create multiple VMs on top of the physical host machine.

2. What is a Container?
   - layers of stacked images, it will have a lightweight base image mostly linux base image such as alpine. In simple words it is a package, which containers the application (source code) and dependencies. This package is a portable artifact.
   - Containers are stored in a place in cloud is called, container repository

3. What is Docker?
   - Docker is a container technology tool, which is used to automate the deployment process of an application leveraging container technology.

4. What is Docker Engine?
   - Docker Engine consists of server daemon, CLI and APIs. Which can be used for orchestration of containers.

5. What is Docker image?
   - It is the package which contains the application code, library and configuration. It is a movable artifact.

6. What is Docker Container?

- Running a docker image creates a container. So Docker container is the running environment along with the application. Multiple containers can be created with the same image.

7. What is Docker Registry?
   - It stores and distributes the docker image.
     Docker has public registry: https://hub.docker.com/

8. What is Difference b/w Container v/s VM?
   - Assume a VM is created for an application. If we want to create two applications, we have to create two VMs, each VM has to create a guest OS of its own. On Host OS, we will be creating multiple guest OS, which is time and resource consuming. In the case of containers, The OS kernel of the host machine will be used for the APP creation, thereby eliminating the creation of additional guest OS.

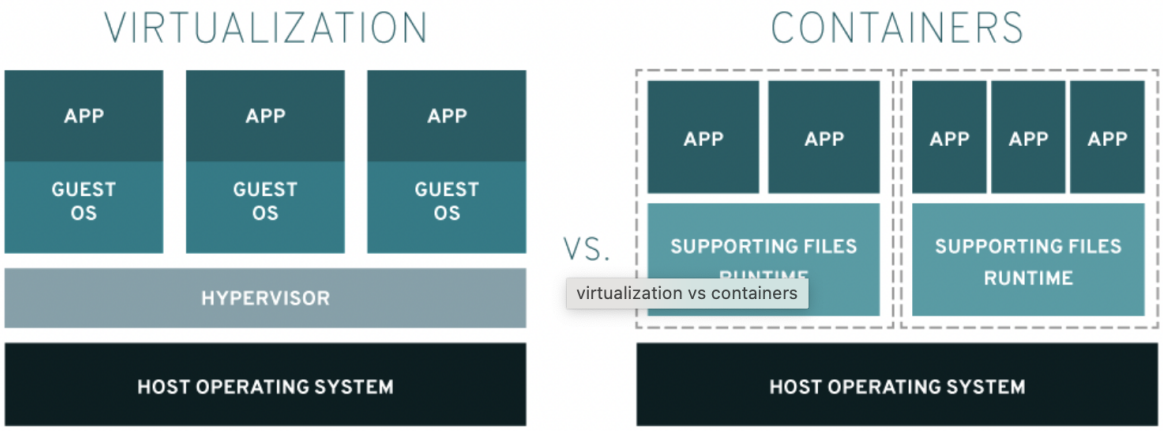


Image reference: (RedHat)
https://www.redhat.com/en/topics/containers/containers-vs-vms

9. What is the Docker toolbox?
   - Since the container, rely on the host OS Kernel. Containers created based on the one OS kernel can't be directly installed on top of the another OS kernel. I.e container created for Linux can't be directly installed on top of the windows. Docker provides a tool to bridge this gap and it is called the Docker Toolbox.

Docker Components:

- Docker Engine: The runtime that builds and runs containers. Docker Image: Read-only template used to create containers.
- Docker Container: Runnable instance of an image.
- Dockerfile: Text document that contains all the commands a user could call on the command line to assemble an image. -
-Docker Compose: A tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services.

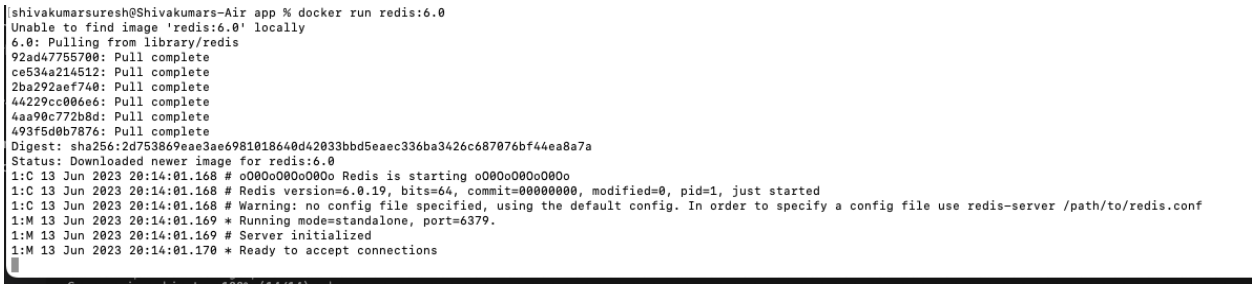
Basic Docker Commands:

1. **$ docker run image:<version/tag>**
   Eg: $docker run postgres:9.6

   $ **docker run -d image:<version/tag>**
   -d: will run the container in detached mode (background)

   This will pull layers of images (if not present locally) and create a container from the image and run it. If you don't specify the version it will pull the latest version.

   Run command is equivalent to pull and start. (see next commands)



2. **$ docker pull image:<version/tag>**
   Eg: $docker pull postgres:9.6
   This will pull the image from the docker repository and store it in the host machine.

   Note: Say in future if you want to pull a latest version, then the docker pull will only pull the layer of image which are different, if there is a common image it will not pull it.
   Eg:
   92ad47755700: Pull complete
   ce534a214512: Pull complete
   2ba292aef740: Pull complete
   44229cc006e6: Pull complete
   4aa90c772b8d: Pull complete
   493f5d0b7876: Pull complete

   The above are the layers of images pulled for a redis 6.0

   $ docker pull redis:6.2
   6.2: Pulling from library/redis
   92ad47755700: Already exists
   ce534a214512: Already exists
   2ba292aef740: Already exists
   0eec6c1f66a1: Pull complete
   a33e85a78410: Pull complete
   24ac515a2c02: Pull complete
   Digest:
   sha256:a3928dc86b022d7812dde2378bc0d0a0e85a447cc24345c528529a8f3d627ea5
   Status: Downloaded newer image for redis:6.2

   If a layer of image exists locally it will use it, instead of pulling it from the repository.

3. **$ docker ps**
   Lisplays the running containers,

   CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

```
600229ed13a6   redis        "docker-entrypoint.s…"   23 hours ago   Up 23 hours   6379/tcp   kind_khayyam
c5fe0f21e438   redis:6.2.0   "docker-entrypoint.s…"   24 hours ago   Up 24 hours   6379/tcp   bold_benz
```

Here, containerID is the unique ID created for a container by docker, image is the docker image file name, ports represents the ports used by the container in their environment, name is the randomly (unique) generated by docker, we can override it by giving the customized name for the container. The CONTINERID and NAME will be used for future references.

$ **docker ps -a**
Lists all the containers which are either in running/not running state.

4. **$ docker stop <conatinerid/name>**
   $ docker start <conatinerid/name>

   To stop and start the container. (it will work on conatines not on images)

5. Port binding during run time.
   $ docker run -p<host_port>:<conatiner_port> imagename:<tag>

   If you run docker ps, you will see:

```
CONTAINER ID   IMAGE        COMMAND                  CREATED        STATUS        PORTS                    NAMES
600229ed13a6   redis        "docker-entrypoint.s…"   23 hours ago   Up 23 hours   0.0.0.0:6001->6379/tcp   kind_khayyam
c5fe0f21e438   redis:6.2.0   "docker-entrypoint.s…"   24 hours ago   Up 24 hours   0.0.0.0:6000->6379/tcp   bold_benz
```

6. **$ docker images**
   Lists all the image present in the docker

7. **$ docker logs <conatinerid/name>**
   Displays the logs for the specified container.

8. Customize the container name
   $ docker run -d –name <new_conatiner_name> <image_name>

```
% docker run -d -p6002:6379 --name hello_redis redis
519779fcc7b596cc574edd585ae42d7c353c089e8f78e6a01313d60597bf1c58
% docker ps
```

```
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS         PORTS                    NAMES
519779fcc7b5   redis        "docker-entrypoint.s…"   9 seconds ago   Up 8 seconds   0.0.0.0:6002->6379/tcp   hello_redis
600229ed13a6   redis        "docker-entrypoint.s…"   24 hours ago    Up 24 hours    0.0.0.0:6001->6379/tcp   kind_khayyam
c5fe0f21e438   redis:6.2.0  "docker-entrypoint.s…"   24 hours ago    Up 24 hours    0.0.0.0:6000->6379/tcp   bold_benz
```

9. **$ docker exec -it <conatinerid/name> /bin/<bash or sh>**
   -it: means interactive terminal.
   This command can be used to enter inside the container, useful for debugging.

   Eg:

```
% docker exec -it hello_redis /bin/bash
root@519779fcc7b5:/data# ls
root@519779fcc7b5:/data# pwd
/data
root@519779fcc7b5:/data# exit
```

10. $ **docker network ls**
    This will list docker containers network, (this local to docker containers) and not related to the Host network.

    Eg:

```
% docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
3d77e303b959   bridge        bridge    local
5eb756ce3974   host          host      local
c3df1ec54399   mono-network  bridge    local
f5078190f4f3   none          null      local
```

11. **$ docker network create <network_name>**
    This will create a new network name. This network can be used to create a common network for the container.
    Eg:

```
% docker network create own-nw
39be9447710cac31768a9a5b7af36f9269688070d0fd2e8ffff1fe0c75b1cc40

% docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
3d77e303b959   bridge        bridge    local
```

```
5eb756ce3974   host          host      local
c3df1ec54399   mono-network  bridge    local
f5078190f4f3   none          null      local
39be9447710c   own-nw        bridge    local
```

12. Specifying the containers to use existing network
    Eg: **$ docker run -d -p600:6379 –network <n/w name> <image_name>**

13. Setting environment variables for containers, during creation.
    **$ docker run -d -p600:6379 –network <n/w name> **
    **-e <enviornmnet_key1>=<value1> **
    **-e <enviornmnet_key2>=<value2> **
    –name <new_name> <image_name>
    Here -e species the environment variables.

14. **docker rm <containerid>:**
    Removes container

15. **docker rmi <image>:**
    Removes docker image.

Docker Compose:

It is difficult to create and manage containers, using the command line, docker provides an easier method to run multiple containers at one using a yaml file called docker-compose file and using docker-compose command.

Eg:

```
version: '1'  // version of the docker compose file
services:     // containers list
    conatiner_1<user_name>:   // container specification.
        image: <conatiner_image>
        ports:
            - <host_port_1>:<conatiner_port_1>
            - <host_port_1>:<conatiner_port_1>
```

```
        environment:
            - <enviornmnet_key1>=<value1>
            - <enviornmnet_key2>=<value2>

    conatiner_2<user_name>:
        image: <conatiner_image>
        ports:
            - <host_port_1>:<conatiner_port_1>
            - <host_port_1>:<conatiner_port_1>
        environment:
            - <enviornmnet_key1>=<value1>
            - <enviornmnet_key2>=<value2>
```

This will create docker containers with common n/w, so we don't need to specify the n/w value.

How to run?
$ docker-compose -f <docker_compose_file_name>.yaml up
Here -f represents the docker compose file.

This will mix logs of all the containers output.

How to stop?
$ docker-compose -f <docker_compose_file_name>.yaml down
This will stops the continents and removes it (even network)

Dockerfile

Suppose, you have created an application and now you want to create a docker container out of it. Then we need to build a docker image using docker image using docker build command and Dockerfile.

Dockerfile:
It is a blueprint for building a docker image, similar to SQL.

An image has to be built using an underlying image base.
Syntax:

FROM <base_image> // same as install node

ENV <enviornmnet_key1>=<value1> \ // setting environment variables
<enviornmnet_key1>=<value1>

RUN <commands> // executes on the container machine.

COPY <host_file_path_soucer> <conatiner_file_path_target> // executes on host.

ENTRYPOINT [<entrypoint_file>] // entry point for the application

CMD [<command_1> <input>] // command to run the application inside the container
// can have multiple

How to build from Dockerfile?
$ docker build -t <image_name>:<tag> <path to Dockerfile>

eg: **$ docker build -t app:2.0 .** // assuming the Docker file is in the current location

Private Docker Registry:

The place to store and distribute the organization's private containers.
Eg: An organization can store the docker image on AWs cloud i.e ECR (Elastic container registry).

pushing an image to ECR:

Steps:

1. **$ docker login** // login to the private repository

2. Follow the naming conventions for private docker registry as
   registryDomain/imageName:tag

By default it will refer to default value i.e docker.io

Rename the image
$ docker tag <old> <*registryDomain/imageName:tag*>

3. Push the image // publish to private registry
   $ docker push <*registryDomain/imageName:tag*>

Pull from docker registry // before executing please login.
$ docker pull <*registryDomain/imageName:tag*>

Docker Volumes:

When a container is restarted the data in it will be lost, so *Docker Volumes* are the way to persist data generated by and used by Docker containers. Docker manages the volumes, and they are independent of the container lifecycle, meaning they won't get deleted even if the container is deleted.

This works by mounting/mapping a container's (virtual) file system to the host file system. This can be achieved during the creation of the container.

$ docker run -v <host_mount_path>:<conatiner_file_sytsem>
Here -v represents the volume.

Option:
<host_mount_path> can be
1. Complete path: uses the specified path
2. Empty: docker will create unique and random file systems and maps it.
3. Name: reference by name. (preferred method)

Volumes in docker compose file:

```
version: '1'  // version of the docker compose file
services:     // containers list
        conatiner_1<user_name>:   // container specification.
                image: <conatiner_image>
                ports:
                        - <host_port_1>:<conatiner_port_1>
                        - <host_port_1>:<conatiner_port_1>
                environment:
                        - <enviornmnet_key1>=<value1>
                        - <enviornmnet_key2>=<value2>
                volumes:
                        - <name_1>:<container virtual file 1>
                        - <name_2>:<container virtual file 2>

        conatiner_2<user_name>:
                image: <conatiner_image>
                ports:
                        - <host_port_1>:<conatiner_port_1>
                        - <host_port_1>:<conatiner_port_1>
                environment:
                        - <enviornmnet_key1>=<value1>
                        - <enviornmnet_key2>=<value2>
                volumes:
                        - <name_3>:<container virtual file 1>
                        - <name_4>:<container virtual file 2>
```

volumes:
    <name_1>: // this is a reference
        driver:<location>
    <name_2>:
        driver:<location>
    <name_3>:
        driver:<location>
    <name_4>:
        driver:<location>

References:

IBM Youtube: https://www.youtube.com/watch?v=0qotVMX-J5s
The Docker official Document: https://docs.docker.com/get-started/overview/
Nana Janashia YouTube: https://www.youtube.com/watch?v=3c-iBn73dDE