Docker Basics Commands

In this gist, two main classes of systems (machines) are referred many times:

- host system: the machine on which container runtime has been installed and containers (running and exited) reside,
- container system: the system running inside a container.

`dkr` is an alias to `docker` command.

1. Technologies for Linux Containers

Prior to the advent of Docker and Kubernetes, there have been three (among others) technologies for Linux Containers:

- LXC
- LXD
- CFS

2. What Docker EE offers over and above Docker CE Features

Docker Enterprise Edition addons include:

- image management
- image security
- universal control plane for managing and orchestrating container runtimes

3. Docker commands

1. to list running containers (active containers)

   ```
   docker ps
   ```

2. to list all containers (exited as well as active containers)

   ```
   docker ps -a
   ```

3. to start a container and run a specific command in that

   ```
   docker run ubuntu ls /

   docker start ubuntuContainerId ls /

   docker start ubuntuContainerName ls /
   ```

4. to run a specific command in an already running container

   ```
   docker exec deadcafe123abc ipconfig

   docker exec ubuntuContainerName ipconfig
   ```

5. to avoid keeping terminal busy with the output of a running container, next time start the container with `-d` option

   ```
   docker run -d redis
   ```

6. to connect a terminal again to a running container

   ```
   docker attach containerId
   ```

4. Some Advanced Commands and Options for Beginners

These commands and options may very well ease the work life of newbies.

1. `-v` option for specifying data volume location

   docker run -v /opt/datadir:/var/lib/mysql mysql

   - Here `/opt/datadir` is the directory on host system

2. Docker command `inspect`: to inspect much more details of a container

   dkr inspect containerId

   - In the output JSON, it's easy to view many pieces of important details and rectify errors related to networking etc.

3. Docker command `logs`: to view output listing and error listing of a detached container

   docker logs containerId

   docker logs containerName

5. Naming a Docker Image and Publishing It

- You may name an image while building it. This action adds a name and a version (a tag to signify some special milestone in the "life") to the image. This action is called 'tagging' and is important to perform before pushing it to some registry of repositories.
- Pushing an image to a repository (usually a remote repo) is called publishing.

Examples:

- Example 1:

  Dockerfile

  ==

  FROM alpine/ubuntu

  RUN apt-get update

  RUN apt-get install python

  RUN pip install flask

  RUN pip install flask-mysql

  COPY . /opt/srccode

  ENTRYPOINT FLASK_APP=/opt/srccode/app.py flask run

  ==

  docker build -t dockerhubAccount/imageTagname:1.2.0 .

docker push dockerhubAccount/imageTagname:1.2.0

- Example 2:

  Dockerfile

  ==

  FROM alpine/ubuntu

  RUN apt-get update && apt-get -y install python

  RUN pip install flask flask-mysql

  COPY . /opt/srccode

  ENTRYPOINT FLASK_APP=/opt/srccode/app.py flask run

  ==

  docker build Dockerfile -t dockerhubAccount/imageTagname:1.2.1

  docker push dockerhubAccount/imageTagname:1.2.1

6. Difference between ENTRYPOINT and CMD in Dockerfile

Whenever an image is "run" with `docker run` command, a container is created and started. A command must be executed upon creation of the container "to logically run the container" for user's need.

- The default command is specified with `ENTRYPOINT`.
- The default arguments to the ENTRYPOINT command are specified with `CMD` in Dockerfile.

  entrypoint=welcomingCommand

  cmd=arg0

- Specifying a value against `ENTRYPOINT` in `dkr run` command is a way to override the default (welcoming) command. And, the argument following the image name overrides the `CMD` setting when `ENTRYPOINT` is specified in the same command. See the below command as a simple example:

  dkr run ENTRYPOINT=sh imageName arg0

Example with Dockerfile and Commands

===

FROM Ubuntu

ENTRYPOINT ["man"] --default command

CMD ["ls"] -- default argument

dkr run helper

dkr run helper cd

dkr run --ENTRYPOINT timer helper /

- When arguments are passed to `dkr run` command following the image name without `ENTRYPOINT` command-line option, the first (leftmost) argument is designated as the welcoming command and the other arguments are passed to that command as its arguments.

7. Docker Networking and Network Types

There are three types of docker networks:

- bridge: for the networking of local containers with their host (with containers and host connected without manual effort by user for network setup). This is the default network type.
- host: for setting containers with their own ports, without port mapping. With this type, only one container may be run on a particular port. So, only one container of an image at maximum can be running at a time.
- none: for running containers without networking. These containers are not connected even to the host (for user data).

How to create multiple bridge networks

Multiple networks of containers for different applications can be created with the commands of format:

dkr network create \

--driver bridge \

--subnet 188.21.0.0/16

custom-bdg-net

The last argument in the command is the network name. It must be unique for every network.

Checking IP Addr of a Docker container

dkr inspect contId

- The preceding command lists the following details (among others) about the container:

networkSettings >> Networks >> bridge >> {Gateway,IPAddress,MacAddress}

- The code in one container can refer to another container by its name (service name in docker-compose file) as:

```
mysql.connect(mysql-container-name)
```

- Here Docker DNS resolves containers by name.

8. How a container is created (Layered approach)

"copy on write" mechanism of containers betweeen container layer and image layers. This mechanism explains how a container data consists of two types of layers or say two data blocks:

- read-write layer of container-specific data and user data (which are volatile), and
- read-only layer of the data and configuration out of image definition.

9. Docker volumes

Docker volumes are useful when user's data is to be persisted and shared across container executions (multiple runs of a container and/or across the runs of different containers as well).

- A docker volume is created as:

  ```
  docker volume create my_vol
  ```

- The above command creates a Docker volume (to store data from container) at location on host machine:

  ```
  /var/lib/docker/volumes/my_vol
  ```

- The volume created above can be reference by its logical name `my_vol` in Docker commands as:

  ```
  docker run -v my_vol:/var/lib/mysql mysqlimage
  ```

- This notation looks similar to relative paths in bash as it does not start with `/`. Such volumes are all placed in `/var/lib/docker/volumes` sub-dirs on host machine.
- For volumes in other locations, specify absolute paths as:

  ```
  docker run -v /var/db1/my_vol:/var/lib/mysql mysqlimage
  ```

  ```
  docker run --mount type=bind,source=/var/db1/my_vol,target=/var/lib/mysql
  mysqlimage
  ```

10. Docker Compose

Docker Compose has already been through tremendous modifications and upgrades till now. In present times, it is almost inevitable to write a YAML file for docker-compose with `version` in beginning of the file content.

The journey from version 1 via `version: "2"` to `version: "3.8"` (well, this is our way of hinting code level usage format :-)):

- version 1: It didn't have `depends_on` for startup ordering among containers and creating separate networks (esp. the types other than 'bridge')
- version 2: started creating dedicated bridge networks to be shared by all the containers for a YAML, and introduced `depends_on` to order their startup sequence.

11. Registry and Repositories

- In order to pull or push an image using a command-line tool or a terminal, a user needs to authenticate a priori. And, this authentication is performed with the folling commend issued using that very CLI:

```
docker login reg.newhost.com
```

- to create a private Image Registry, you may use the app named "registry" available on [Docker Hub](https://hub.docker.com/_/registry):

  ```
  dkr run -d -p 4000:5000 --name reg registry
  ```

- to push the image to the registry (collection of repositories) on your local host:

  ```
  dkr image tag web-app:1.0 localhost:4000/web-app:1.0
  dkr push localhost:4000/web-app:1.0
  ```

- to push to and pull from a registry over your network:

  ```
  dkr push 192.168.34.22:4000/web-app
  dkr pull 192.168.34.22:4000/web-app
  ```

The following articles are optional to read and no need to master upon.

12. Docker Engine

- Docker Engine has three main components:
  - Docker CLI: A user uses it to issue command to Docker daemon.
  - REST API: API makes the communication between Docker CLI and Docker Daemon possible.
  - Docker Daemon: Docker daemon performs the requested actions in background (if possible and allowed).

- Docker CLI on another host may access the REST API on first host by:

  ```
  docker -H=remoteengine:2375 run imagename
  ```

- Control Groups (cgroups) are used for imposing resource limiting on containers:

  ```
  dkr run --cpus=.4 imgName
  dkr run --memory=248m imgName
  ```

13. Orchestration of Containers

For managing (starting, networking, stopping) containers within a single network on a single host, docker-compose YAML file is very versatile and convenient. On other hand, for managing the containers and their clustering across multipel hosts, you need to manage their complex network. Here comes into view the action: orchestration.

Among others, two favorite mechanisms for container orchestration are:

- docker swarm, and
- Kubernetes.

Note

Only some of the commands have been tested recently. Very few of the commands may fail. Feel free to report such deviations to reply to LinkedIn post or by adding issues in GitHub repository.

You May Mark your feedback at the Following Contact Points

- ****LinkedIn:**** <https://www.linkedin.com/in/rishirajopenminds>

- ****X:**** <https://twitter.com/RishiRajDevOps>

- ****Start Page:**** <https://bio.link/rishiraj49de>

- ****GitHub:**** <https://github.com/rishiraj88>

Credits and Gratitude

- I thank all who have mentored, taught and guided me. Also, I appreciate who have supported my work with pair programming and more.

- Special thanks to FreeCodeCamp this time!