Docker_Handbook Documentation

Daniel Raj

Oct 29, 2019

CONTENTS:

CHAPTER

ONE

GETTING STARTED WITH DOCKER

1.1 Installing Docker on fedora

- Install using the repository
  1. Set up the repository
     - Install the dnf-plugins-core package

       ```
       $ sudo dnf -y install dnf-plugins-core
       ```

     - Add repo

       ```
       $ sudo dnf config-manager --add-repo \
       https://download.docker.com/linux/fedora/docker-ce.repo
       ```

  2. Install Docker Engine - Community

     ```
     $ sudo dnf install docker-ce docker-ce-cli containerd.io
     ```

  3. Start Docker

     ```
     $ sudo systemctl start docker
     ```

  4. Verify that Docker Engine - Community is installed correctly by running the hello-world image.

     ```
     $ sudo docker run hello-world
     ```

- Running docker as non-root user
  - The docker daemon always runs as the root user.

  If you would like to use Docker as a non-root user, you can add your user to the "docker" group with something like:

  ```
  $ sudo usermod -aG docker <your-user>
  ```

- For more details refer https://docs.docker.com/install/linux/docker-ce/fedora/

1.2 Docker Basic Operations

- **docker create** - creates a container but does not start it.

  ```
      Usage:  docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
  ```

  ```
  $ docker create -it fedora bash
  ```

  ```
  $ docker create --name daniel -it fedora bash
  ```

- **docker start** - starts a container.

  ```
  $ docker start -a -i <container-name/id>
  ```

- **docker stop** - stops a running container.

  ```
  $ docker stop <container-name/id>
  ```

- **docker run** - creates and starts a container in one operation.

  ```
  $ docker run --name test -it fedora
  ```

- **docker exec** - creates and starts a container in one operation.

  ```
  $ docker exec --name test -it fedora touch testing.txt
  ```

- **docker rm** - removes one or more containers.

  ```
  $ docker rm --force redis
  ```

  - the above command will force remove redis container.

- **docker rename** - renames a containers.

```
$ docker rename my_container my_new_container
```

- **docker pull** - pulls a image for docker registery.

  ```
  $ docker pull fedora
  ```

- **docker images** - will display all the images.

  ```
  $ docker images
  ```

- **docker rmi** - deletes one or more images

  ```
  $ docker rmi fd484f19954f
  ```

- **docker ps** - displays running containers.

  ```
  $ docker ps
  ```

- **docker ps –all** - displays all containers(running and exited).

  ```
  $ docker ps --all
  ```

- **docker commit** - Create a new image from a container's changes.

Notes:

- Create comamnd will pull the image from repository(if the image does not exists) and will then create a container
- Normally if you run a container without options it will start and stop immediately, if you want keep it running you can use the command, docker **run -td container_id** this will use the option -t that will allocate a pseudo-TTY session and -d that will detach automatically the container (run container in background and print container ID)
- To Remove all stopped containers in one go run **$ docker rm $(docker ps -a -q)**

For detailed help

- run `man docker-<cmd>` eg: man `docker-create` or man `docker-run`

    - or

- run `docker <cmd> --help` eg: *docker create –-help*

1.3 Publish or Expose Ports

By default Docker containers can make connections to the outside world, but the outside world cannot connect to containers. Each outgoing connection will appear to originate from one of the host machine's own IP addresses thanks to an iptables masquerading rule on the host machine that the Docker server creates when it starts.

The Docker server creates a masquerade rule that lets containers connect to IP addresses in the outside world. If you want containers to accept incoming connections, you will need to provide special options when invoking **docker run**.

By default, when you create a container, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which are not connected to the container's network, use the –publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host. Something like below

- **-p host_port:container_port**
- **-p 8080:80** Map TCP port 80 in the container to port 8080 on the Docker host.

If you want to expose multiple ports use the following command

```
$ docker run --name test -it -p 8080:80 -p 2020:22 fedora
```

roamware@kb:~$ sudo docker port dbserver 22/tcp -> 0.0.0.0:4203 3306/tcp -> 0.0.0.0:3306 3306/tcp -> 0.0.0.0:4504 roamware@kb:~$

1.4 Copy-on-Write(CoW) Strategy

When we launch an image, the Docker engine does not make a full copy of the already stored image. Instead, it uses something called the copy-on-write mechanism. This is a standard UNIX pattern that provides a single shared copy of some data, until the data is modified.

A Docker image is built up from a series of layers.

When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer.

The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged.

1.4.1 `docker commit`

Create a new image from a container's changes.

The commit operation will not include any data contained in volumes mounted inside the container.

By default, the container being committed and its processes will be paused while the image is committed. This reduces the likelihood of encountering data corruption during the process of creating the commit.

Example

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             ␣
↪STATUS              PORTS               NAMES
```

(continues on next page)

(continued from previous page)

```
c3f279d17e0a        ubuntu:12.04        /bin/bash           7 days ago          Up␣
↪25 hours                             desperate_dubinsky
197387f1b436        ubuntu:12.04        /bin/bash           7 days ago          Up␣
↪25 hours                             focused_hamilton

$ docker commit c3f279d17e0a  svendowideit/testimage:version3

f5283438590d

$ docker images

REPOSITORY                        TAG                 ID                  CREATED ␣
↪             SIZE
svendowideit/testimage            version3            f5283438590d        16␣
↪seconds ago       335.7 MB
```

1.5 Manage data in Docker

By default all files created inside a container are stored on a writable container layer. This means that data is not persistent and it will be lost when the conatiner no longer exists.

There are two ways to manage data

1.5.1 #. Map a directory on host to a container

If you want to map a directory on the host to a docker container

```
$ docker run -v $HOSTDIR:$DOCKERDIR <container-name/id>
```

1.5.2 #. Use volumes

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure of the host machine, volumes are completely managed by Docker.

Create and manage volumes

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

Create a volume:

```
$ docker volume create my-vol
```

List volumes:

```
$ docker volume ls
```

Inspect a volume:

:: $ docker volume inspect my-vol

Remove a volume:

```
$ docker volume rm my-vol
```

Start a container with a volume

The following example mounts the volume “myvol2”into “/app/” in the container.

```
$ docker run -d \
  --name devtest \
  --mount source=myvol2,target=/app \
  nginx:latest
```

or

```
$ docker run -d \
  --name devtest \
  -v myvol2:/app \
  nginx:latest
```

Note: Use docker inspect devtest to verify that the volume was created and mounted correctly. Look for the Mounts section

1.6 Privileged Docker

If you see this error **Failed to get D-Bus connection: Operation not permitted** refere this section

By default, Docker containers are "unprivileged" and cannot, for example, run a Docker daemon inside a Docker container. This is because by default a container is not allowed to access any devices, but a "privileged" container is given access to all devices (see the documentation on cgroups devices).

When the operator executes **docker run –privileged**, Docker will enable access to all devices on the host as well as set some configuration in AppArmor or SELinux to allow the container nearly all the same access to the host as processes running outside containers on the host. Additional information about running with –privileged is available on the Docker Blog.

If you want to limit access to a specific device or devices you can use the –device flag. It allows you to specify one or more devices that will be accessible within the container.

```
$ docker run --device=/dev/snd:/dev/snd ...
```

By default, the container will be able to read, write, and mknod these devices.

In addition to –privileged, the operator can have fine grain control over the capabilities using –cap-add and –cap-drop. By default, Docker has a default list of capabilities that are kept.

In Short

If you want to use systemtcl to start/stop serivce inside container run with privileged you should be able to start/stop serivce after that.

```
$ docker run --privileged -t -i --rm ubuntu:latest bash
```

1.7 Debugging help

- **docker logs** - Fetch the logs of a container.

  ```
  $ docker logs hello-world
  ```

  ```
  $ docker logs --tail 10 3f840a82aabe
  ```

- **docker inspect** - Return low-level information on Docker objects

  ```
  $ docker inspect <container-name/id>
  ```

  – If you want to get the ip address of container

  ```
  $ docker inspect --format='{{range .NetworkSettings.
  Networks}}{{.IPAddress}}{{end}}' $INSTANCE_ID
  ```

- **docker attach** - Attach local standard input, output, and error streams to a running container.

- If you want to login to the container you can use attach.