Docker with Kubernetes + Swarm

Table of Contents

1. Creating and Using Containers

- Check versions of our docker cli and engine
- Create a Nginx (web server) container
- Common container management commands
- Docker Networking Basics

2. Docker Commands

- ***`docker version`***

This command simply returns the version of our client and the server (also called the engine) which runs in the background on our machine - called a daemon on macOS. Our docker command line is actually talking to the server on the machine and returning its values as well as the client's values.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker version
Client: Docker Engine - Community
 Version:           19.03.8
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        afacb8b
 Built:             Wed Mar 11 01:21:11 2020
 OS/Arch:           darwin/amd64
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          19.03.8
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.17
  Git commit:       afacb8b
  Built:            Wed Mar 11 01:29:16 2020
  OS/Arch:          linux/amd64
  Experimental:     true
 containerd:
  Version:          v1.2.13
  GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

- ***docker info***

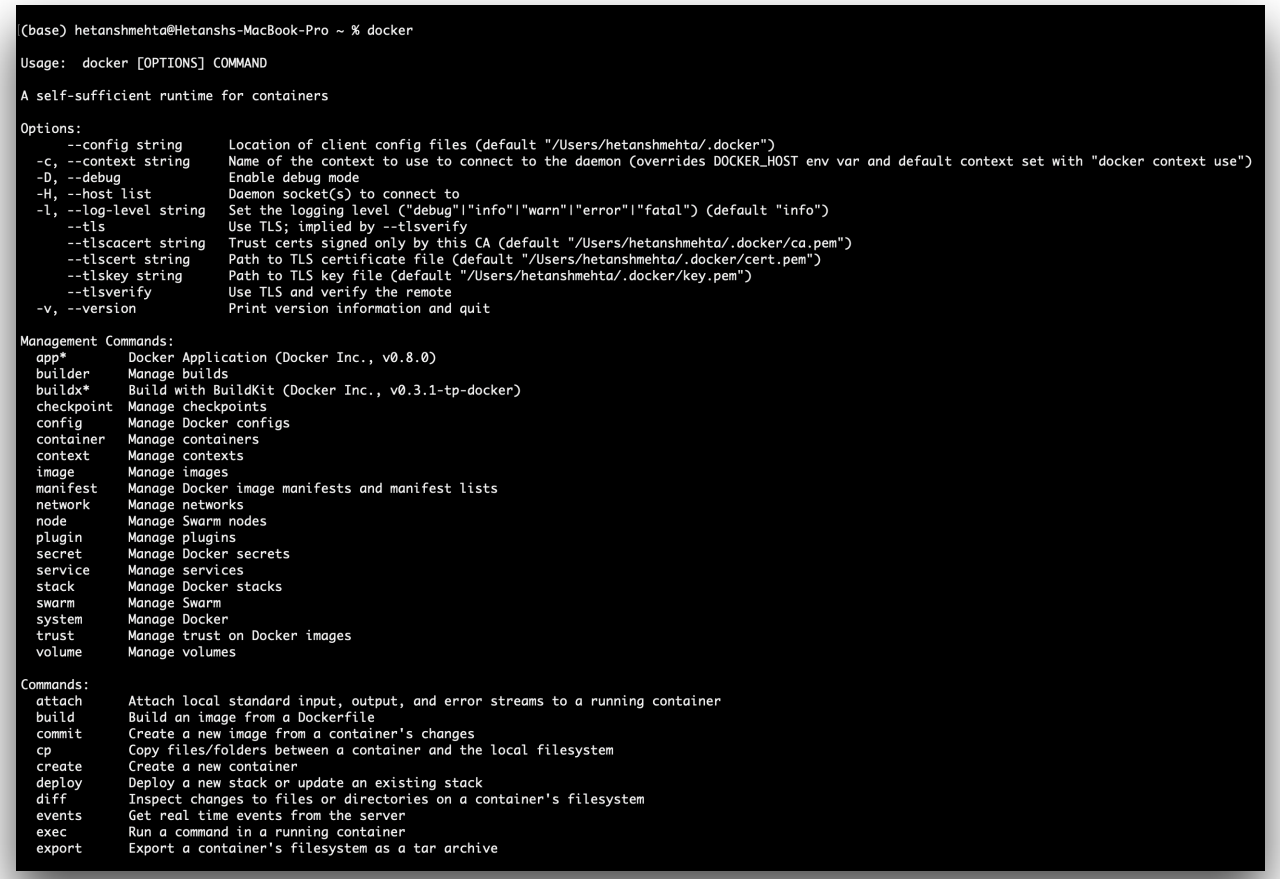
This command returns a lot more stuff than just the version. It'll actually give us a lot of details about our configuration and setup for our engine.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker info
Client:
 Debug Mode: false
 Plugins:
  app: Docker Application (Docker Inc., v0.8.0)
  buildx: Build with BuildKit (Docker Inc., v0.3.1-tp-docker)

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 19.03.8
 Storage Driver: overlay2
  Backing Filesystem: <unknown>
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 4.19.76-linuxkit
 Operating System: Docker Desktop
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.848GiB
 Name: docker-desktop
 ID: 6UPS:2KP3:FQBB:LDQL:4TYV:FLBN:G3O6:2VO3:3KE6:AJBX:JKWM:DITI
 Docker Root Dir: /var/lib/docker
 Debug Mode: true
  File Descriptors: 37
  Goroutines: 43
  System Time: 2020-06-06T21:41:09.570812627Z
  EventsListeners: 3
 HTTP Proxy: gateway.docker.internal:3128
 HTTPS Proxy: gateway.docker.internal:3129
 Registry: https://index.docker.io/v1/
 Labels:
```

- ***docker***

This returns a list of all the commands we can use in Docker.



```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/Users/hetanshmehta/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/Users/hetanshmehta/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/Users/hetanshmehta/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/Users/hetanshmehta/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  app*        Docker Application (Docker Inc., v0.8.0)
  builder     Manage builds
  buildx*     Build with BuildKit (Docker Inc., v0.3.1-tp-docker)
  checkpoint  Manage checkpoints
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  deploy      Deploy a new stack or update an existing stack
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
```

Syntax for docker management commands:
NEW: docker <management-command> <sub-command> (options)
OLD: docker <command> (options)

3. Starting a Nginx Web Server

- Image vs. Container
- Run/Stop/Remove Containers
- Check container logs and processes

3.1. Image vs Container

- An **image** is the binaries and libraries and source code that make up our application, i.e., image is the application we want to run.
- A **container** is the running instance of the image. We can have many containers based off the same image.

We're going to use the Open Source Nginx web server. So we'll be starting our containers based off that Nginx image. We get all our images from registries.

Docker's default image registry is called **Docker Hub** (hub.docker.com)

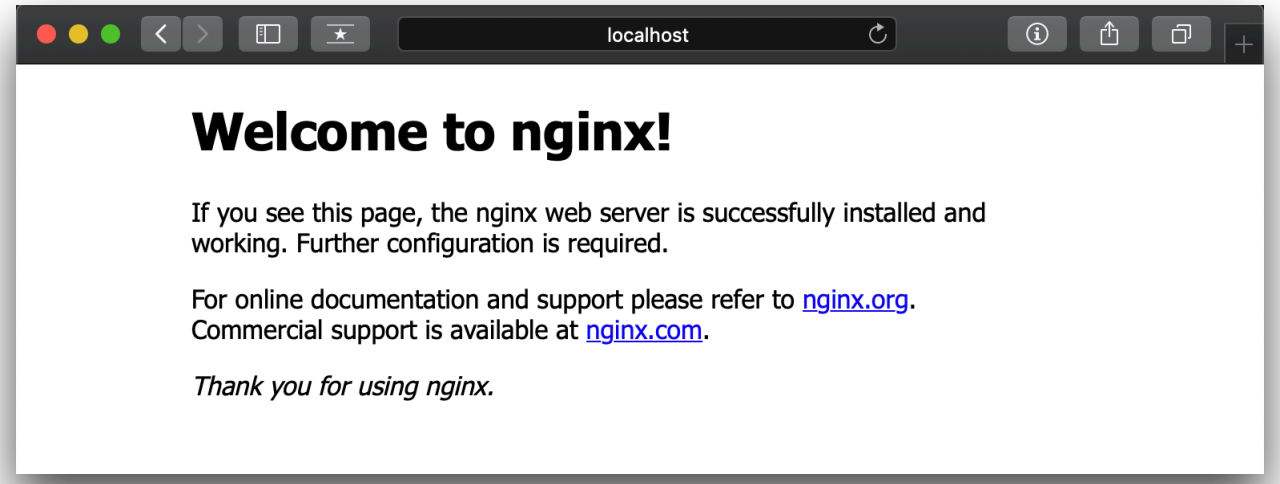
3.2. Run/Stop/Remove Container

- ***`docker container run –publish 80:80 nginx`***
- Downloads image 'nginx' from Docker Hub
- Starts a new container from the image
- Opened port 80 on the host IP (local machine) and sends all traffic from it to the executable running inside that container on the container (container IP) on port 80.
- Since Nginx is a web server, the traffic just forwards automatically through the web browser to the localhost and into that container

Note that we'll get a "bind" error if the left number (host port) is being used by anything else, even another container. We can use any port we want on the left, like **`8080:80`** or **`8888:80`**, then use **`localhost:8888`** when testing.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run --publish 80:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
afb6ec6fdc1c: Pull complete
dd3ac8106a0b: Pull complete
8de28bdda69b: Pull complete
a2c431ac2669: Pull complete
e070d03fd1b5: Pull complete
Digest: sha256:c870bf53de0357813af37b9500cb1c2ff9fb4c00120d5fe1d75c21591293c34d
Status: Downloaded newer image for nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

Switch over to a web browser, and type **localhost**.



Our Nginx server is listening.

We can hit refresh several times and we'll see the log entries in our command line.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run --publish 80:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
afb6ec6fdc1c: Pull complete
dd3ac8106a0b: Pull complete
8de28bdda69b: Pull complete
a2c431ac2669: Pull complete
e070d03fd1b5: Pull complete
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container --help

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

But we don't always want this running in the foreground inside of our command line.

- ***docker container run –publish 80:80 –detach nginx***

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run --publish 80:80 --detach nginx
dab0d9a7e0b5f38f3c6a2c644a2ad193e4c3fc52296f828c3a87bcd1dde572a0
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

The detach tells Docker to run it in the background. And we get back the unique container ID of our container. Every time we run a new container, we get a new container ID.

- ***docker container ls***

Command to list all our containers. And we'll see the one that's still running here. Only shows the running containers.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
dab0d9a7e0b5        nginx               "/docker-entrypoint.…"   About an hour ago   Up About an hour    0.0.0.0:80->80/tcp   confident_robinson
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***docker container stop <container_ID>***

Stops the container matching the container id.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container stop dab0d
dab0d
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

Now when we list our containers, nothing shows up as no containers are running.

- ***docker container ls -a***

- When we ran each time the docker container run command, it started a new container from that Nginx image. One the right we notice that there's these random names which we didn't use.
- The container IDs and the names are required to be unique and if we don't specify it, then it will be created for us. We can always name something ourselves, or let it pick one on its own.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
dab0d9a7e0b5        nginx               "/docker-entrypoint.…"   About an hour ago   Exited (0) 2 minutes ago                        confident_robinson
4bd9cf156353        nginx               "/docker-entrypoint.…"   2 hours ago         Exited (0) About an hour ago                    crazy_antonelli
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- Start a new container under **`detach`** mode. Open the web browser and type **`localhost`**. Refresh it a few times.

- By running in detached mode, we are able to have access to our command line when the container spins up and runs. Without it, we would have logs constantly fed onto the screen.

- ***`docker container run –publish 80:80 –detach –name webhost nginx`***

Creates a new container with name webhost.

- ***`docker container logs <container_name>`***

It spits out the latest logs in the web host.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run --publish 80:80 --detach --name webhost nginx
d42da8ec7b15bc1b6f4baf1ec54590b0f2e3a399f79ce2b46a0cf4493ca5d66a
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
d42da8ec7b15        nginx               "/docker-entrypoint.…"   6 seconds ago       Up 5 seconds        0.0.0.0:80->80/tcp   webhost
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
d42da8ec7b15        nginx               "/docker-entrypoint.…"   15 seconds ago      Up 14 seconds               0.0.0.0:80->80/tcp   webhost
dab0d9a7e0b5        nginx               "/docker-entrypoint.…"   2 hours ago         Exited (0) 49 minutes ago                        confident_robinson
4bd9cf156353        nginx               "/docker-entrypoint.…"   3 hours ago         Exited (0) 2 hours ago                           crazy_antonelli
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container logs webhost
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
172.17.0.1 - - [07/Jun/2020:01:27:59 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Sa
fari/605.1.15" "-"
172.17.0.1 - - [07/Jun/2020:01:28:00 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Sa
fari/605.1.15" "-"
172.17.0.1 - - [07/Jun/2020:01:28:00 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Sa
fari/605.1.15" "-"
172.17.0.1 - - [07/Jun/2020:01:28:01 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Sa
fari/605.1.15" "-"
172.17.0.1 - - [07/Jun/2020:01:28:01 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Sa
fari/605.1.15" "-"
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***`docker container top <container_name>`***

Helps to identify the processes running inside the container. In case of Nginx, there's actually a master process and then it spawns worker processes based on the configuration.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container top webhost
PID                 USER                TIME                COMMAND
2109                root                0:00                nginx: master process nginx -g daemon off;
2164                101                 0:00                nginx: worker process
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***`docker container –help`***

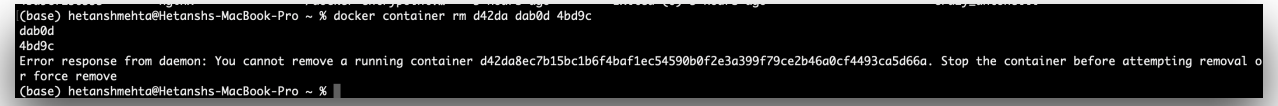
Get a list of commands we could try on that container

We will now try to remove our containers. To list the containers, we can type in ***`docker container ls -a`***

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
d42da8ec7b15        nginx               "/docker-entrypoint.…"   58 minutes ago      Up 58 minutes               0.0.0.0:80->80/tcp   webhost
dab0d9a7e0b5        nginx               "/docker-entrypoint.…"   3 hours ago         Exited (0) 2 hours ago                           confident_robinson
4bd9cf156353        nginx               "/docker-entrypoint.…"   3 hours ago         Exited (0) 3 hours ago                           crazy_antonelli
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***docker container rm <container_ID_1> <container_ID_2> … <container_ID_n>***

Remove all the 3 containers at the same time.



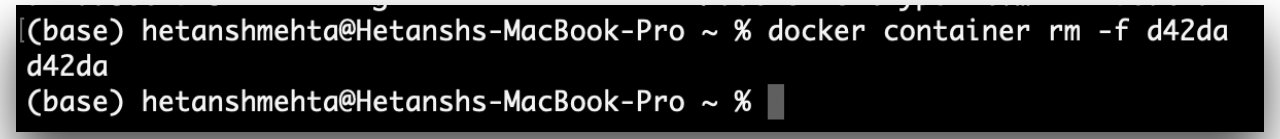
We got an error. The first two were safely deleted. We cannot remove a running container because it's a safety measure. It makes sure that we don't accidentally remove the actual process that's still running.

We can simply stop the container first and then remove it.

Alternatively,

- ***docker container rm -f <container_ID>***

-f is for force. We will forcefully remove our active container.



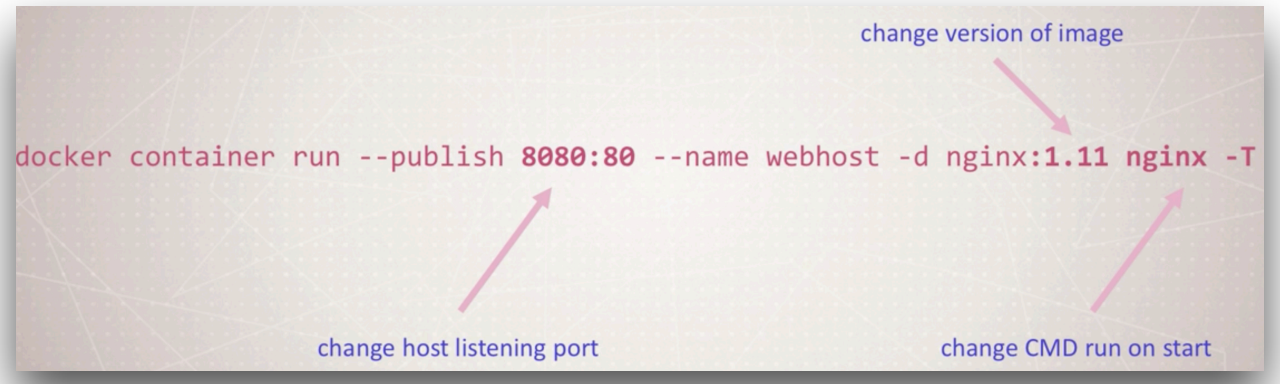
All our containers have been cleaned.

4. What happens when we run a Container?

When we type ***`docker container run`***, in the background it's actually going to look for the image that we specified at the end of the command.

When we typed nginx at the very end, that was the name of the image we wanted to run as a new container.

1. Looks for that image locally in image cache, doesn't find anything
2. Then looks in remote image repository (defaults to Docker Hub)
3. Downloads the latest version (nginx: latest version by default) and stores it in the image cache.
4. Creates new container based on that image. It's not going to make a copy of the image. It's actually going to just start a new layer of changes, right on top of where that image left off.
5. It's going to customize the networking - give it a specific virtual IP address that's inside a Docker virtual network. Gives it a virtual IP on a private network inside docker engine.
6. Opens up port we specified on host and forwards to port 80 in container. If we didn't specify the **Publish** command, i.e., **`–publish`**, it's not going to open up any ports at all. Since we did the **`80:80,`** that's telling it to take the port 80 on the host and forward all that traffic to the port 80 in the container.
7. The container finally starts by using the command (CMD) specified in the Dockerfile.



5. Containers vs VM

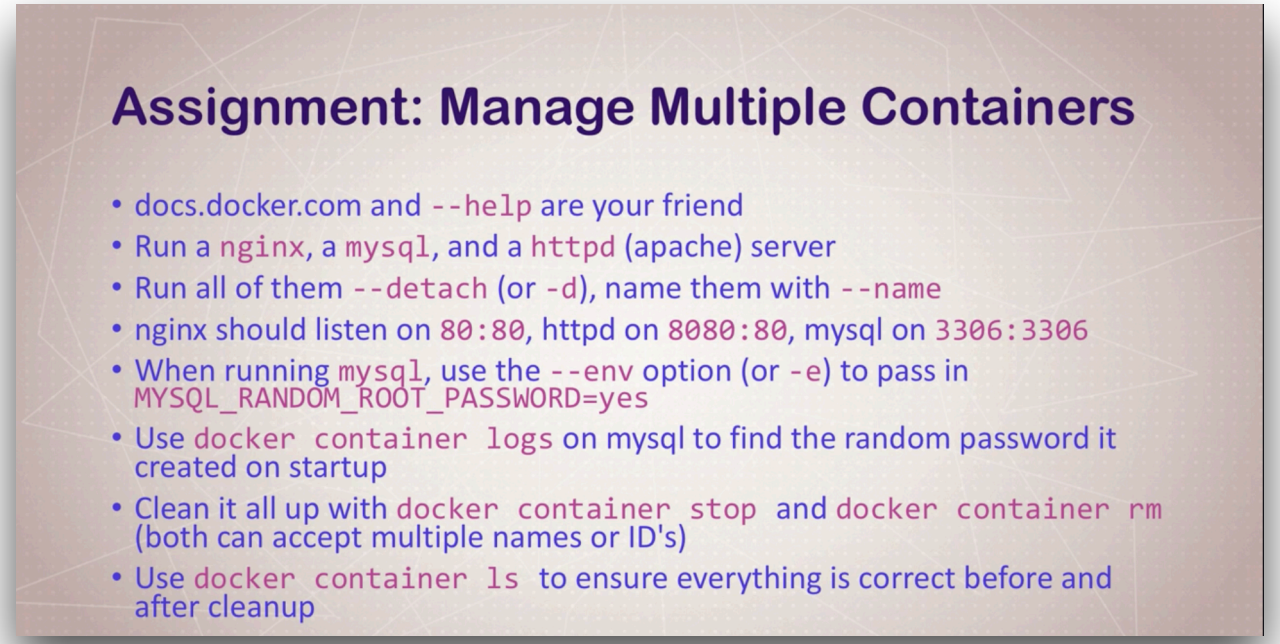
Often people compare containers to VMs. True, there are some similarities, but there's so many things that aren't the same.

But actually, containers are really just a process running on your host operating system and are nothing like a virtual machine.

A few useful links:

1) https://github.com/mikegcoleman/docker101/blob/master/Docker_eBook_Jan_2017.pdf
2) https://www.youtube.com/watch?v=sK5i-N34im8&feature=youtu.be&list=PLBmVKD7o3L8v7Kl_XXh3KaJl9Qw2lyuFl

Assignment 1

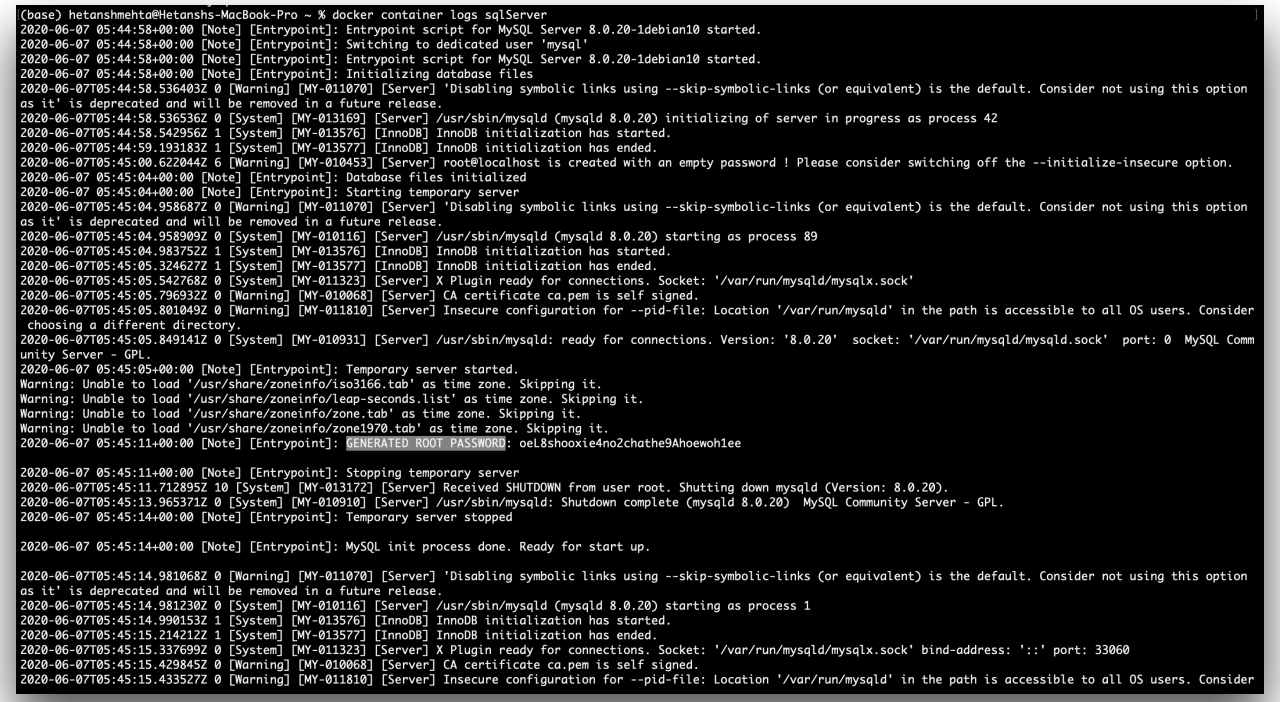


Solution:

Initializing containers

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -d -p 80:80 --name nginxServer nginx
d4f7cb3401c4403c19a3ee9239671d944c5bdb39e49cf58b8295b90c95a3b601
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -d -p 8080:80 --name apacheServer httpd
72794e168334ca8f367ce9140184ef015cbe099721e48aecbfbd9096b91c1d86
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -d -p 3306:3306 --name sqlServer --env MYSQL_RANDOM_ROOT_PASSWORD=yes mysql
771db4b77aa21e727f9135d24aceb146a8c186a873d111dec916d05d7c150a8a
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
771db4b77aa2        mysql               "docker-entrypoint.s…"   3 seconds ago       Up 3 seconds        0.0.0.0:3306->3306/tcp, 33060/tcp   sqlServer
72794e168334        httpd               "httpd-foreground"       About a minute ago  Up About a minute   0.0.0.0:8080->80/tcp                apacheServer
d4f7cb3401c4        nginx               "/docker-entrypoint.…"   About a minute ago  Up About a minute   0.0.0.0:80->80/tcp                  nginxServer
```

SQL Password from logs:



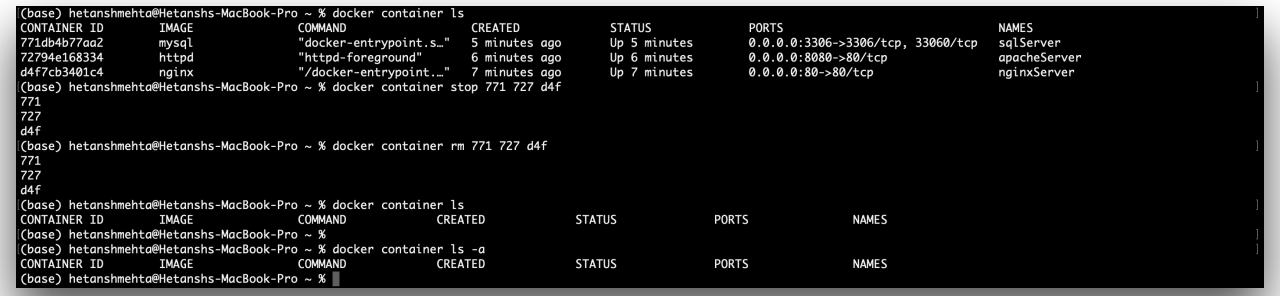
```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container logs sqlServer
2020-06-07 05:44:58+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.20-1debian10 started.
2020-06-07 05:44:58+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2020-06-07 05:44:58+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.20-1debian10 started.
2020-06-07 05:44:58+00:00 [Note] [Entrypoint]: Initializing database files
2020-06-07T05:44:58.536403Z 0 [Warning] [MY-011070] [Server] 'Disabling symbolic links using --skip-symbolic-links (or equivalent) is the default. Consider not using this option
as it' is deprecated and will be removed in a future release.
2020-06-07T05:44:58.536536Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.20) initializing of server in progress as process 42
2020-06-07T05:44:58.542956Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2020-06-07T05:44:59.193183Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2020-06-07T05:45:00.622044Z 6 [Warning] [MY-010453] [Server] root@localhost is created with an empty password ! Please consider switching off the --initialize-insecure option.
2020-06-07 05:45:04+00:00 [Note] [Entrypoint]: Database files initialized
2020-06-07 05:45:04+00:00 [Note] [Entrypoint]: Starting temporary server
2020-06-07T05:45:04.958687Z 0 [Warning] [MY-011070] [Server] 'Disabling symbolic links using --skip-symbolic-links (or equivalent) is the default. Consider not using this option
as it' is deprecated and will be removed in a future release.
2020-06-07T05:45:04.958909Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.20) starting as process 89
2020-06-07T05:45:04.983752Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2020-06-07T05:45:05.324627Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2020-06-07T05:45:05.542768Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/var/run/mysqld/mysqlx.sock'
2020-06-07T05:45:05.796932Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2020-06-07T05:45:05.801049Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider
 choosing a different directory.
2020-06-07T05:45:05.849141Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.20'  socket: '/var/run/mysqld/mysqld.sock'  port: 0  MySQL Comm
unity Server - GPL.
2020-06-07 05:45:05+00:00 [Note] [Entrypoint]: Temporary server started.
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
2020-06-07 05:45:11+00:00 [Note] [Entrypoint]: GENERATED ROOT PASSWORD: oeL8shooxie4no2chathe9Ahoewoh1ee

2020-06-07 05:45:11+00:00 [Note] [Entrypoint]: Stopping temporary server
2020-06-07T05:45:11.712895Z 10 [System] [MY-013172] [Server] Received SHUTDOWN from user root. Shutting down mysqld (Version: 8.0.20).
2020-06-07T05:45:13.965371Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 8.0.20)  MySQL Community Server - GPL.
2020-06-07 05:45:14+00:00 [Note] [Entrypoint]: Temporary server stopped

2020-06-07 05:45:14+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start up.

2020-06-07T05:45:14.981068Z 0 [Warning] [MY-011070] [Server] 'Disabling symbolic links using --skip-symbolic-links (or equivalent) is the default. Consider not using this option
as it' is deprecated and will be removed in a future release.
2020-06-07T05:45:14.981230Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.20) starting as process 1
2020-06-07T05:45:14.990153Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2020-06-07T05:45:15.214212Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2020-06-07T05:45:15.337699Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/var/run/mysqld/mysqlx.sock' bind-address: '::' port: 33060
2020-06-07T05:45:15.429845Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
2020-06-07T05:45:15.433527Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider
```

Remove and Clean-Up



```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
771db4b77aa2        mysql               "docker-entrypoint.s…"   5 minutes ago       Up 5 minutes        0.0.0.0:3306->3306/tcp, 33060/tcp   sqlServer
72794e168334        httpd               "httpd-foreground"       6 minutes ago       Up 6 minutes        0.0.0.0:8080->80/tcp                apacheServer
d4f7cb3401c4        nginx               "/docker-entrypoint.…"   7 minutes ago       Up 7 minutes        0.0.0.0:80->80/tcp                  nginxServer
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container stop 771 727 d4f
771
727
d4f
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container rm 771 727 d4f
771
727
d4f
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

6. What's Going On In Containers

- We will now run Docker commands to understand what's going on inside a container

Start a few containers

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -d --name nginx nginx
d0bcc6e3a6b8bc753d9462b281e864a877e769bb94a83ae851e454122e8457be
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -d --name mysql -e MYSQL_RANDOM_ROOT_PASSWORD=true mysql
a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
a238f804acaf        mysql               "docker-entrypoint.s…"   5 seconds ago       Up 4 seconds        3306/tcp, 33060/tcp   mysql
d0bcc6e3a6b8        nginx               "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp              nginx
```

- ***`docker container top <container_name>`***

Process list in one container

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container top mysql
PID                 USER                TIME                COMMAND
1806                999                 0:01                mysqld
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container top nginx
PID                 USER                TIME                COMMAND
1716                root                0:00                nginx: master process nginx -g daemon off;
1772                101                 0:00                nginx: worker process
```

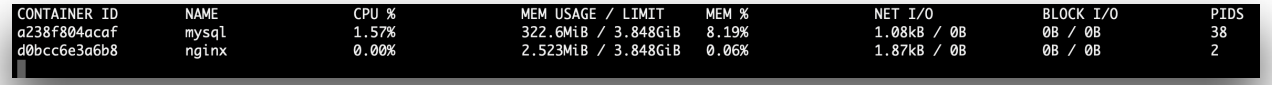
- ***`docker container inspect <container_name>`***

Details of how the container started and how it's configured and all of its metadata (startup, config, volumes, networking, etc). Format: JSON

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container inspect mysql
[
    {
        "Id": "a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4",
        "Created": "2020-06-08T22:06:16.4673558Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "mysqld"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1806,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-06-08T22:06:16.8948519Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:30f937e841c82981a9a6363f7f6f35ed6b9d5e3f16df50a72207e4a2a389983f",
        "ResolvConfPath": "/var/lib/docker/containers/a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4/hostname",
        "HostsPath": "/var/lib/docker/containers/a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4/hosts",
        "LogPath": "/var/lib/docker/containers/a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4/a238f804acafc82b53463bad1c98eaba33beb61771841c41578043463e3a93f4-j
son.log",
        "Name": "/mysql",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
```

- ***docker container stats***

Gives us a view of all our containers and the performance statistics for each in a real time stream.



We get a streaming view of live performance data about our containers. Obviously in a production environment, we're going to have a more complicated monitoring and performance.

7. Getting a Shell Inside Containers

This is all about getting into a container to mess around with the inside. No SSH needed, the docker cli is a great substitute for adding SSH to containers.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
87f2b708fbe9        mysql               "docker-entrypoint.s…"   6 seconds ago       Up 5 seconds        3306/tcp, 33060/tcp     mysql
77f26a92c9f4        nginx               "/docker-entrypoint.…"   14 seconds ago      Up 13 seconds       80/tcp                  nginx
```

- ***`docker container run -it <image> [COMMANDS] [ARG …]`***

Start a new container interactively. The ***`-it`*** is actually two separate options. The '***`t`***' actually gives us a `pseudo-tty` which simulates a real terminal, like what SSH does. The '***`i`***' (interactive) allows us to keep that session open so that we can keep running more commands. When we're doing a docker container run, we have optional command and arguments that we can send in to the new container we're about to start to tell it what to run.

`Command`: `bash`

If the above command is run with ***`-it`***, it will give us a terminal inside the running container. Notice when we run the command, we act as root on the container (and not the host). The number following it is actually the container ID. We can enter the standard bash commands: ***`ls -al, pwd, cd, ….`*** Type ***`exit`*** to stop the container.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -it --name proxy nginx bash
root@24f25da60871:/# ls
bin  boot  dev  docker-entrypoint.d  docker-entrypoint.sh  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@24f25da60871:/# pwd
/
root@24f25da60871:/# ls -al
total 80
drwxr-xr-x   1 root root 4096 Jun  9 21:50 .
drwxr-xr-x   1 root root 4096 Jun  9 21:50 ..
-rwxr-xr-x   1 root root    0 Jun  9 21:50 .dockerenv
drwxr-xr-x   2 root root 4096 May 14 14:50 bin
drwxr-xr-x   2 root root 4096 May  2 16:39 boot
drwxr-xr-x   5 root root  360 Jun  9 21:50 dev
drwxr-xr-x   1 root root 4096 Jun  2 16:23 docker-entrypoint.d
-rwxrwxr-x   1 root root 1087 Jun  2 00:35 docker-entrypoint.sh
drwxr-xr-x   1 root root 4096 Jun  9 21:50 etc
drwxr-xr-x   2 root root 4096 May  2 16:39 home
drwxr-xr-x   1 root root 4096 Jun  2 00:35 lib
drwxr-xr-x   2 root root 4096 May 14 14:50 lib64
drwxr-xr-x   2 root root 4096 May 14 14:50 media
drwxr-xr-x   2 root root 4096 May 14 14:50 mnt
drwxr-xr-x   2 root root 4096 May 14 14:50 opt
dr-xr-xr-x 151 root root    0 Jun  9 21:50 proc
drwx------   2 root root 4096 May 14 14:50 root
drwxr-xr-x   3 root root 4096 May 14 14:50 run
drwxr-xr-x   2 root root 4096 May 14 14:50 sbin
drwxr-xr-x   2 root root 4096 May 14 14:50 srv
dr-xr-xr-x  12 root root    0 Jun  9 21:50 sys
drwxrwxrwt   1 root root 4096 Jun  2 00:35 tmp
drwxr-xr-x   1 root root 4096 May 14 14:50 usr
drwxr-xr-x   1 root root 4096 May 14 14:50 var
root@24f25da60871:/# exit
exit
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

Let's see the command that it ran. The default command for an Nginx container is to run the Nginx program itself. But when we typed our last command, we changed that default program to actually be Bash, which gave us our shell. When we exited the shell, the container stopped. Because containers only run as long as the command that it ran on startup runs. Since we changed it to Bash, simply exiting Bash quit it.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS                  NAMES
24f25da60871        nginx               "/docker-entrypoint.…"   12 minutes ago      Exited (0) 6 minutes ago                           proxy
87f2b708fbe9        mysql               "docker-entrypoint.s…"   27 minutes ago      Up 27 minutes               3306/tcp, 33060/tcp    mysql
77f26a92c9f4        nginx               "/docker-entrypoint.…"   27 minutes ago      Up 27 minutes               80/tcp                 nginx
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

The command is different from the first Nginx server. Let's set up a container with a full distribution of Linux instead of actually doing the Nginx.

- ***docker container run -it –name ubuntu ubuntu***

The above command is going to download the Ubuntu image and place us in the prompt of this new container. Its default CMD is bash, so we don't have to specify it. We can use the apt package manager just like how we would in a standard Ubuntu server. We can actually install something: ***apt-get install -y curl***

The thing about Ubuntu and other distributions inside a container is that they're usually very minimal. A live CD or the ISO of Ubuntu which we would normally put on a Virtual Machine is going to have a lot more software installed by default than a Ubuntu container. We can always add more software to with the apt package manager.

Once we exit the shell again, it will actually stop the container. If we start it again, it would have curl installed in it. But if we create a new container from Ubuntu image, that different container wouldn't have curl command line installed.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -it --name ubuntu ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d51af753c3d3: Pull complete
root@7288bcc1bd9e:/# curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
root@7288bcc1bd9e:/#
```

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container exec -it mysql bash
root@87f2b708fbe9:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
mysql        1  0.9  8.7 1724908 353268 ?      Ssl  21:35   0:47 mysqld
root       602  1.0  0.0   3868  3248 pts/0    Ss   22:57   0:00 bash
root       608  0.0  0.0   7640  2660 pts/0    R+   22:57   0:00 ps aux
root@87f2b708fbe9:/# exit
exit
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

```
root@7288bcc1bd9e:/# apt-get update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [118 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [40.7 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [10.6 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [1077 B]
Get:7 http://archive.ubuntu.com/ubuntu focal-updates InRelease [107 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
Get:9 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:10 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:12 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [237 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [1077 B]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [13.1 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [129 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [2903 B]
Fetched 14.0 MB in 3s (4816 kB/s)
Reading package lists... Done
```

```
root@7288bcc1bd9e:/# apt-get install -y curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

We can start our container again using

- ***docker container start -ai <container_name>***

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
7288bcc1bd9e        ubuntu              "/bin/bash"              20 minutes ago      Exited (127) 3 minutes ago                      ubuntu
24f25da60871        nginx               "/docker-entrypoint.…"   About an hour ago   Exited (0) 55 minutes ago                       proxy
87f2b708fbe9        mysql               "docker-entrypoint.s…"   About an hour ago   Up About an hour            3306/tcp, 33060/tcp   mysql
77f26a92c9f4        nginx               "/docker-entrypoint.…"   About an hour ago   Up About an hour            80/tcp              nginx
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container start --help

Usage:  docker container start [OPTIONS] CONTAINER [CONTAINER...]

Start one or more stopped containers

Options:
  -a, --attach               Attach STDOUT/STDERR and forward signals
      --checkpoint string    Restore from this checkpoint
      --checkpoint-dir string   Use a custom checkpoint storage directory
      --detach-keys string   Override the key sequence for detaching a container
  -i, --interactive          Attach container's STDIN
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container start -ai ubuntu
root@7288bcc1bd9e:/# ls -a
.  ..  .dockerenv  bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@7288bcc1bd9e:/#
```

- ***docker container exec [OPTIONS] CONTAINER COMMAND [ARG …]***

Run additional command in existing container. This helps us run the shell inside a container that's already running something like Nginx or MySQL.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container exec --help

Usage:  docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container

Options:
  -d, --detach               Detached mode: run command in the background
      --detach-keys string   Override the key sequence for detaching a container
  -e, --env list             Set environment variables
  -i, --interactive          Keep STDIN open even if not attached
      --privileged           Give extended privileges to the command
  -t, --tty                  Allocate a pseudo-TTY
  -u, --user string          Username or UID (format: <name|uid>[:<group|gid>])
  -w, --workdir string       Working directory inside the container
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

When we exit this bash shell, the mysql container is still running. Because the ***docker container exec*** actually runs an additional process on an existing running container, it's not going to affect the root process for the MySQL daemon.

This is very useful for jumping into containers when we need to troubleshoot or when we need to change something slightly on a running system, as well as using containers of different distributions to give us the environment we would have if we had a full machine like Ubuntu or an Alpine.

- Different Linux distros in containers

  Alpine is a small security-focused distribution of Linux.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker pull alpine
Using default tag: latest
dockelatest: Pulling from library/alpine
df20fa9351a1:  Already exists
imaDigest: sha256:185518070891758909c9f839cf4ca393ee977ac378609f700f60a771a2dfe321
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              4392e5dad77d        7 days ago          132MB
bash                latest              0980cb958276        10 days ago         13.1MB
alpine              latest              a24bb4013296        11 days ago         5.57MB
mysql               latest              30f937e841c8        2 weeks ago         541MB
httpd               latest              d4e60c8eb27a        3 weeks ago         166MB
ubuntu              latest              1d622ef86b13        6 weeks ago         73.9MB
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

Let's do a Bash shell inside of Alpine. We get the following error:

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run it alpine bash
Unable to find image 'it:latest' locally
docker: Error response from daemon: pull access denied for it, repository does not exist or may require 'docker login': denied: requested access to the resource is denied.
See 'docker run --help'.
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

This is because Bash isn't in the container. And this goes back to the concept that we can only run things in the container that already exists in its image when we start it or maybe something we've added through the exec or run commands.

Alpine does contain ***'sh'*** which we can make use of and install Bash by using its package manager ***'apk'***.

Resources:

Package Management:
https://www.digitalocean.com/community/tutorials/package-management-basics-apt-yum-dnf-pkg

8. Docker Networks

8.1. Concepts

- ***docker container run -p***

We've seen this command before. It is used to publish a container's port(s) to the host.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container run -p 80:80 --name webhost -d nginx
f300fdab6f326e14e91f3984490721fd9132d2c80c544957d993dab23e640b29
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***docker container port CONTAINER [PRIVATE_PORT[/[PROTO]]***

It shows which ports are forwarding traffic to that container from the host.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container port --help

Usage:  docker container port CONTAINER [PRIVATE_PORT[/PROTO]]

List port mappings or a specific mapping for the container
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container port webhost
80/tcp -> 0.0.0.0:80
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***docker container inspect –format '{{.NetworkSettings.IPAddress}}'CONTAINER***

But we haven't talked about the IP address of the container. So we might assume that the container is using the same IP as the host. But by default, that's not true. We can easily get the Docker IP of that container by inspecting it.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker container inspect --format '{{.NetworkSettings.IPAddress}}' webhost
172.17.0.2
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

And that's not the IP of the Mac. So the container and the host are not on the same network.

Default Configs:

- When we actually start a container, we're really in the background connecting to a particular Docker network. By default, that is the "bridge" network. That is, each container is connected to a private virtual network "bridge".
- Each of those networks routs through a NAT firewall, which is actually the Docker daemon configuring the host IP address on its default interface.
- All containers on a virtual network ("bridge" by default) can talk to each other without -p.

However, we can change the default configs:

- Make new virtual networks - maybe one per app, or different ones based on different security requirements.
- Attach containers to more than one virtual network (or none).
- Skip virtual networks and use host IP (—net=host).

How things work (Refer to diagram below):

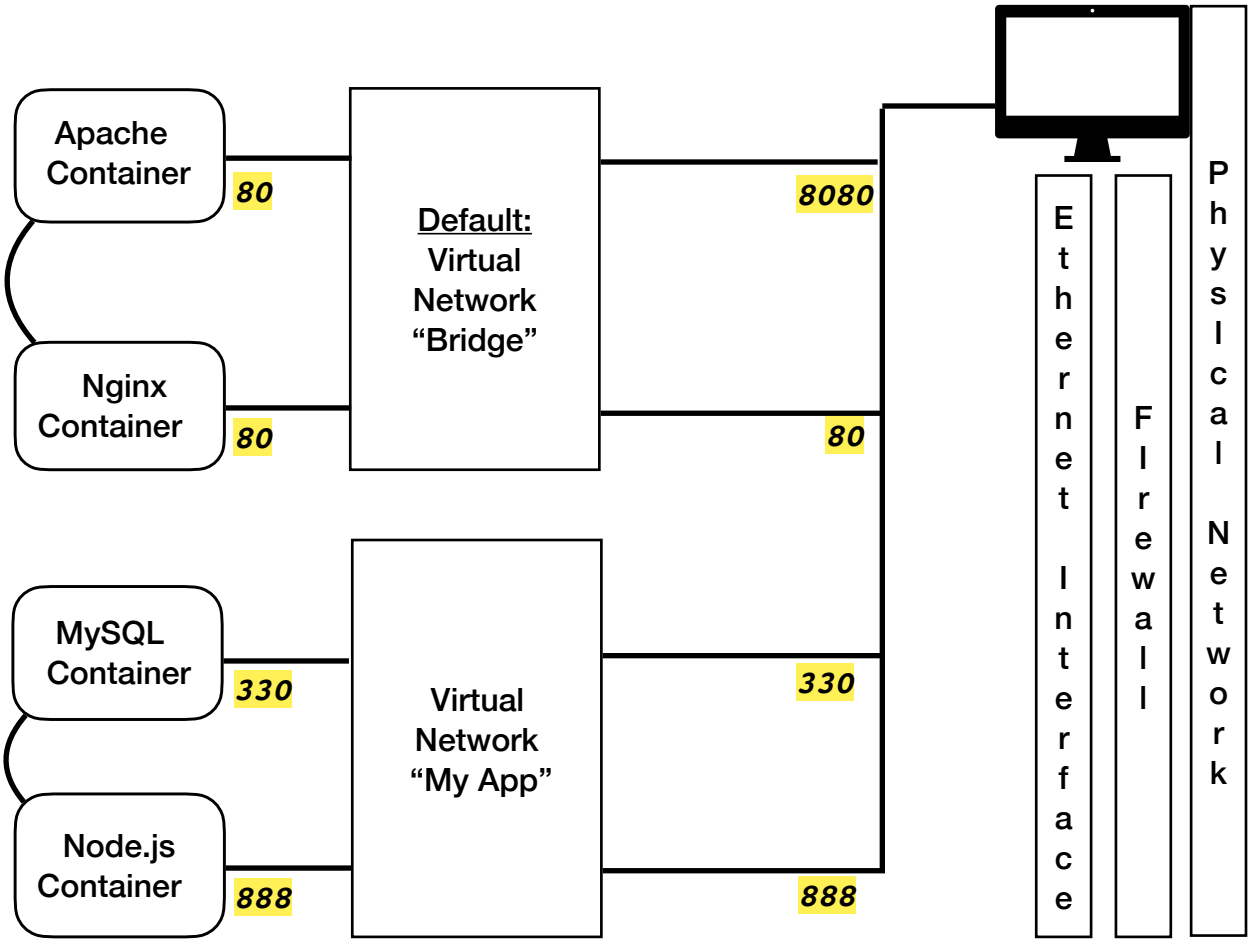
The host operating system is connected to a physical network. There is an Ethernet interface on the host with a firewall which blocks all incoming traffic. And any traffic coming from the containers is going to be NATed by default.

When we start a new container, it gets attached to a virtual network and that virtual network is automatically attached to our Ethernet Interface so that it can communicate.

In our case, we launched that Nginx container, we gave it a -p 80:80. This opens up port 80 on the Ethernet Interface on the host and forward anything coming into port 80 through the virtual network "bridge" to port 80 in that container.

By default, when we create a second container, it's put on that same bridge network (e.g. Apache container). Those two containers can talk freely back and forth on their exposed ports.

We can also create more virtual networks and call them whatever we want. And let's say we connect two containers - MySQL and Node.js. The containers can communicate with the host on the specified ports. But, they are also free to talk amongst themselves over their listening ports.



When we think about virtual networks in Docker, and where containers belong, think about how we would put different containers in proximity to each other because they're related in their application.

As a reminder, on the host level, we cannot have two containers listening to the same port.

8.2. CLI Management

A few command line options to manage how all the virtual networking and IP stuff works.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker network --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

- ***`docker network ls`***

Lists all the networks. As we can create multiple networks, we surely have an ls command to list them.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
82b59b7e716c        bridge              bridge              local
ad23476a3b38        host                host                local
dac596d9c472        none                null                local
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

–`network bridge:` is the default Docker virtual network that bridges through the NAT firewall to the physical network that the host is connected to.

–`network host`: It gains performance by skipping virtual networks but sacrifices security of container model.

`—network none`: removes eth0 and only leaves you with localhost interface in container

- ***`docker network inspect [OPTIONS] NETWORK [NETWORK...]`***

Shows us the details about a specific network.

As the default network is bridge and all our containers until now were attached to it, we can quickly run the inspect command on it. As shown below, the Nginx container is attached to the bridge network.

- ***docker network create [OPTIONS] NETWORK***

Creates a new network. It has an optional driver that we can specify for using built-in and third-party drivers to create a new virtual network.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker network create --help

Usage:  docker network create [OPTIONS] NETWORK

Create a network

Options:
      --attachable           Enable manual container attachment
      --aux-address map      Auxiliary IPv4 or IPv6 addresses used by Network driver (default map[])
      --config-from string   The network from which copying the configuration
      --config-only          Create a configuration only network
  -d, --driver string        Driver to manage the Network (default "bridge")
      --gateway strings      IPv4 or IPv6 Gateway for the master subnet
      --ingress              Create swarm routing-mesh network
      --internal             Restrict external access to the network
      --ip-range strings     Allocate container ip from a sub-range
      --ipam-driver string   IP Address Management Driver (default "default")
      --ipam-opt map         Set IPAM driver specific options (default map[])
      --ipv6                 Enable IPv6 networking
      --label list           Set metadata on a network
  -o, --opt map              Set driver specific options (default map[])
      --scope string         Control the network's scope
      --subnet strings       Subnet in CIDR format that represents a network segment
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

Let's create a new network:

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker network create my_app_net
5a530a6cf2e0d687d673a59a8eee42341963fb9df319155a1e94452e924cc3e3
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

The new network has been created with the default bridge driver. We can list all networks and view it:

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
82b59b7e716c        bridge              bridge              local
ad23476a3b38        host                host                local
5a530a6cf2e0        my_app_net          bridge              local
dac596d9c472        none                null                local
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

We can also connect a container to a network during initialization, by specifying the network name. As an example, let's create a new NginX container and connect it to my_app_net network created above.

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

When we inspect the network my_app_net, we clearly see the newly created container connected to the network.

```
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ % docker network inspect my_app_net
[
    {
        "Name": "my_app_net",
        "Id": "5a530a6cf2e0d687d673a59a8eee42341963fb9df319155a1e94452e924cc3e3",
        "Created": "2020-06-18T04:38:24.043157984Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "b52096d35512d84524baede2732302a5b121086894c5b9dd3fa1f2c8e58d486d": {
                "Name": "new_nginx",
                "EndpointID": "555d8ed2e96c2db4e80bcfae76d8946a080accb8290127cf521c0622d775055a",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
(base) hetanshmehta@Hetanshs-MacBook-Pro ~ %
```

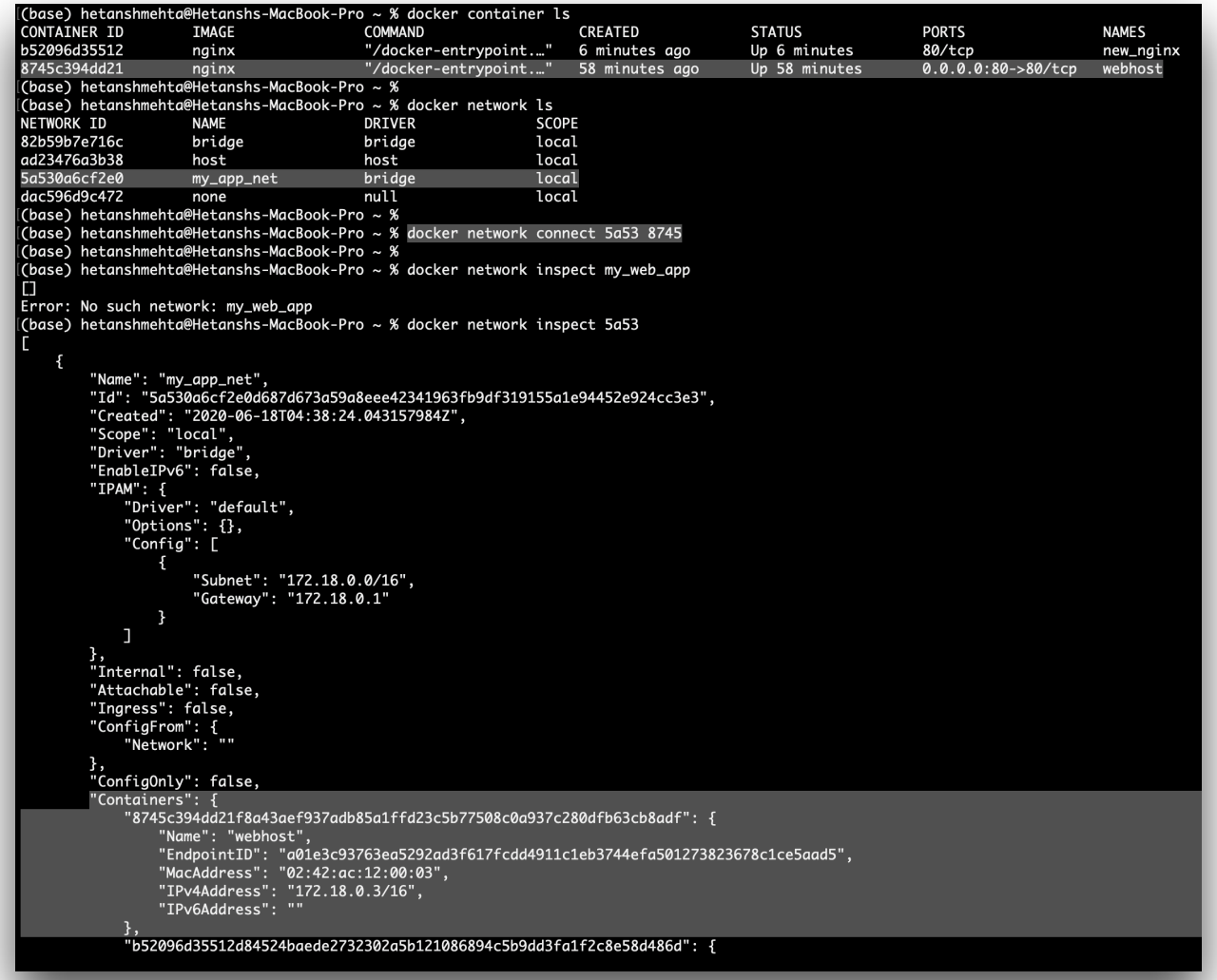
It also has a new IP address.

But we can also do the same with existing networks and containers.

- ***docker network connect [OPTIONS] NETWORK CONTAINER***
- ***docker network disconnect [OPTIONS] NETWORK CONTAINER***

Connect/Disconnect commands for changing a live running container so that a new NIC is created on a virtual network for that container.

We first pick the NETWORK and then pick the CONTAINER we want to connect/ disconnect to.



8.3. Docker Networks: Default Security

- Create your apps so frontend/backend sit on same Docker network
- Their inter-communication never leaves host
- All externally exposed ports closed by default
- You must manually expose via -p, which is better default security!