Fundamentals of DOCKER

By Vignesh S

HOW APPLICATION EVOLVE ?

IDEA

CODE

COLLABORATION

TESTING

DEPLOYMENT

What can you containerize?

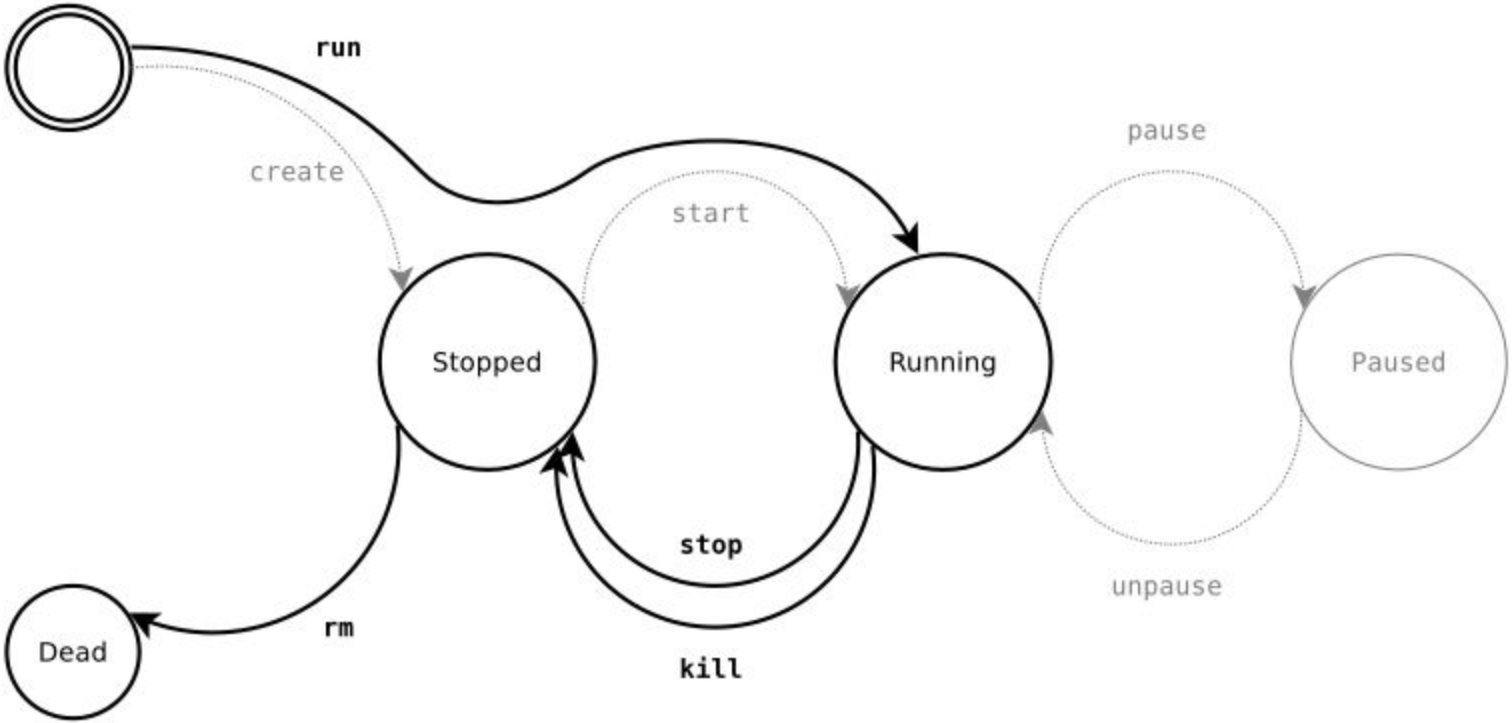
Containerize Everything!!!

A **docker image** is an immutable snapshot of the filesystem

A **docker container** is
- a temporary file system
- layered over an immutable fs (docker image)
- fully writable

A network stack
- with its own private address

A process group
- one main process launched inside the container
- all sub-processes SIGKILLed when the main process exits


run
create
start
pause
Stopped
Running
Paused
stop
unpause
Dead
rm
kill

Basic Docker Commands

Command: **docker --version**

Use: Check the installed Docker version on your system.

Command: **docker info**

Use: Information about your Docker installation.

Command: **docker --help**

Use: Access help and usage information for Docker commands.

Command: **docker images**

Use: List all Docker images on your system.

Basic Docker commands

DOCKER RUN

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

The docker run command is used to create and start a Docker container from a Docker image.

It allows you to configure various aspects of the container's runtime environment using flags

FLAGS and ARGUMENTS

-it:

This combination of flags is commonly used to run the container interactively.

--name <container_name>:

This flag allows you to specify a custom name for the container.

-d:

Run the container in detached mode, which means it runs in the background and doesn't hold your terminal.

- Foreground mode is the default
  - *stdout* and *stderr* are redirected to the terminal
  - `docker run` propagates the exit code of the main process

- With `-d`, the container is run in detached mode:
  - displays the ID of the container
  - returns immediately

```
$ docker run debian date
Tue Jan 20 17:32:07 UTC 2015
$ docker run -d debian date
4cbdefb3d3e1331ccf7783b32b47774fefca426e03a2005d69549f3ff06b9306
$ docker logs 4cbdef
Tue Jan 20 17:32:16 UTC 2015
```

`docker run` — TTY allocation

Use `-t` to allocate a pseudo-terminal for the container

→ without a tty

```
$ docker run debian ls
bin
boot
dev
...
$ docker run debian bash
$
```

→ with a tty (`-t`)

```
$ docker run -t debian ls
bin   dev  home  lib64   mnt  proc  run   selinux  sys  usr
boot  etc  lib   media   opt  root  sbin  srv      tmp  var
$ docker run -t debian bash
root@10d90c09d9ac:/#
```

`docker run` — interactive mode

- By default containers are non-interactive
  - *stdin* is closed immediately
  - terminal signals are not forwarded[4]

```
$ docker run -t debian bash
root@6fecc2e8ab22:/# date
^C
$
```

- With `-i` the container runs interactively
  - *stdin* is usable
  - terminal signals are forwarded to the container

```
$ docker run -t -i debian bash
root@78ff08f46cdb:/# date
Tue Jan 20 17:52:01 UTC 2015
root@78ff08f46cdb:/# ^C
root@78ff08f46cdb:/#
```

`docker run` — override defaults (1/2)

user (`-u`)

```
$ docker run debian whoami
root
$ docker run -u nobody debian whoami
nobody
```

working directory (`-w`)

```
$ docker run debian pwd
/
$ docker run -w /opt debian pwd
/opt
```

FLAGS and ARGUMENTS

--env <KEY=VALUE>:

Set environment variables within the container.

--volume <host_path:container_path> :

Mount a volume from the host into the container.

-network <network_name>:

Connect the container to a specific Docker network.

FLAGS and ARGUMENTS

--entrypoint:

Override the default entry point specified in the Dockerfile.

--user:

Set the username or UID (User Identifier) to use when running the container.

-p <host_port:container_port>:

Use this flag to publish ports from the container to the host. For example, -p 8080:80 maps port 80 in the container to port 8080 on the host.

We’ve explored the IMAGES on the DOCKER HUB, Let's create our own image from a DOCKERFILE.

Choose a Base Image: Start by selecting a base image that suits your application's requirements. Common choices include images for specific programming languages (e.g., Python, Node.js) or minimalistic Linux distributions like Alpine.

Create a Dockerfile: This is where you define the instructions for building your custom image. You specify things like which base image to use, what files to copy into the image, which packages to install, and how to configure the environment.

Build the Image: ***docker build -t my-custom-image:tag .***

```
FROM ubuntu:latest

RUN apt-get update && \
    apt-get install -y nginx

COPY index.html /var/www/html/index.html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```