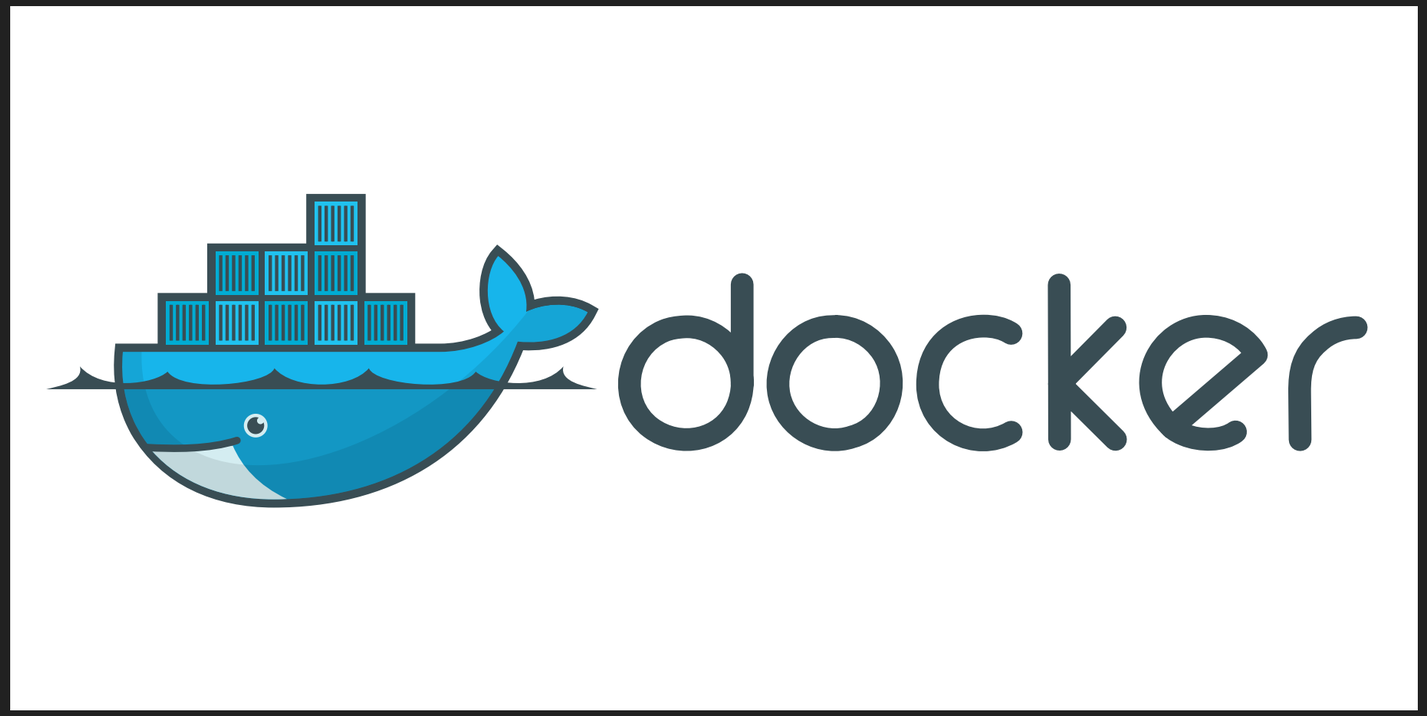
INFINITYWORKS
ACADEMY

Generation

Docker


docker

Overview

- A Brief History of Software Delivery
- An Introduction to Docker
- Setting up Docker
- Running Docker containers
- Creating your own Docker containers
- Sharing your containers with the world

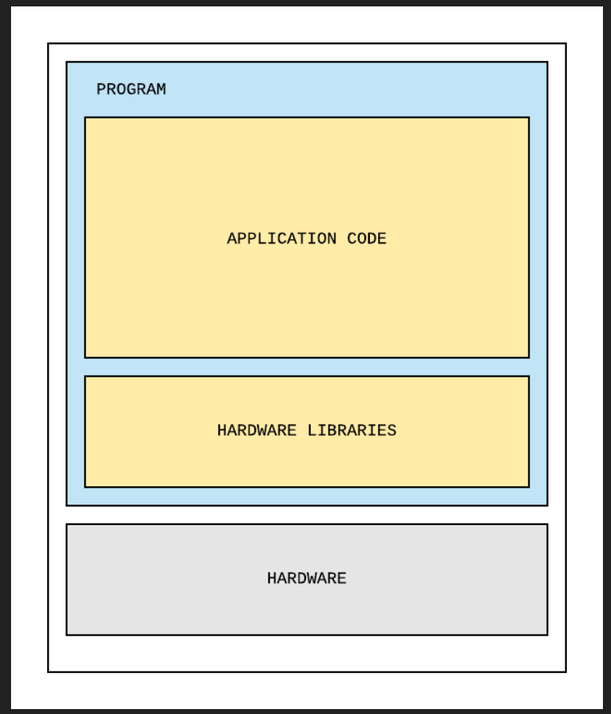
Learning Objectives

- Explain the benefits of images and containers
- Identify commands to start, stop and log into containers
- Define the structure of a Dockerfile and explain how to write them
- Demonstrate how Docker Compose can create and run multi-container applications

The History of Software Delivery

Early Days (1940 - 1960)

Can only run one program

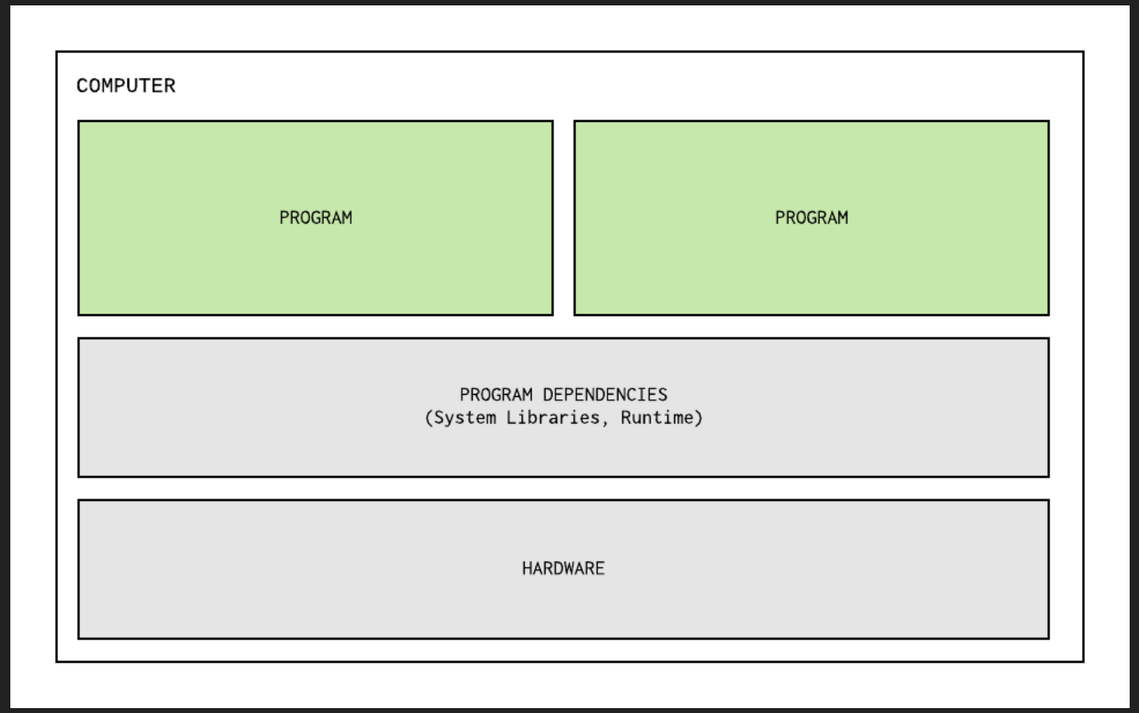


Early Days (1940 - 1960)

- Computers had to be painstakingly rewired or different punch cards fed into them to run different programs
- Very slow, can only do one thing at a time

Multitasking operating systems (1960 - 1990)

Install many programs to disk and run them all at once

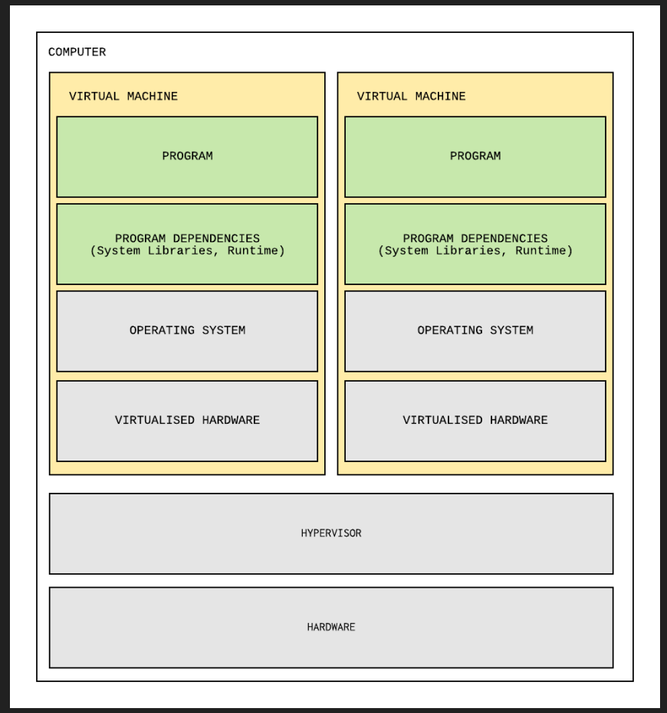


Multitasking operating systems (1960 - 1990)

- Dependency hell
- Processes are not isolated, which means they can interfere with each other
- Security risk

Virtualisation (1990 - 2010s)

Can isolate dependencies in individual virtual machines



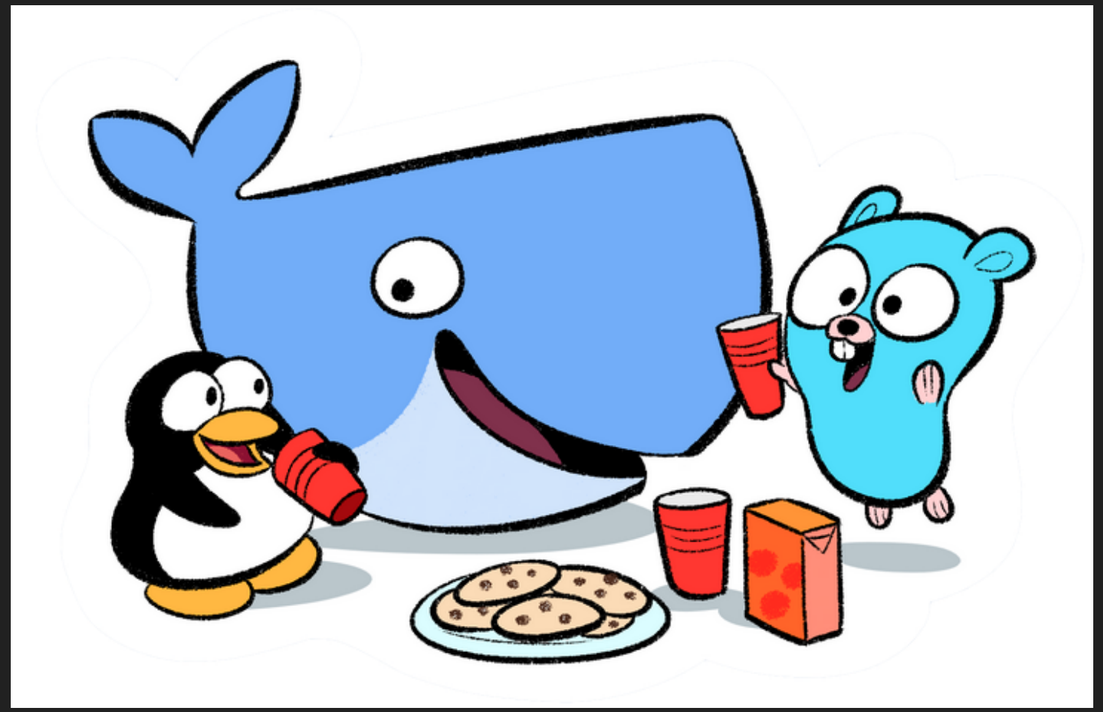
Virtualisation (1990 - 2010s)

- A virtual machine is an emulation of a computer system
- Put simply, it makes it possible to run what appears to be many separate computers on hardware that is actually one computer
- Each VM requires its own underlying OS, and the hardware is virtualized
- A hypervisor, or a virtual machine monitor, is software, firmware, or hardware that creates and runs VMs
- It sits between the hardware and the virtual machine and is necessary to virtualize the server

Virtualisation (1990 - 2010s)

- VMs can take up a lot of system resources
- Each VM runs not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run
- This quickly adds up to a lot of RAM and CPU cycles
- Still economical compared to running separate actual computers, but for some applications it can be overkill, which led to the development of containers

Enter Docker (2013 - Now)



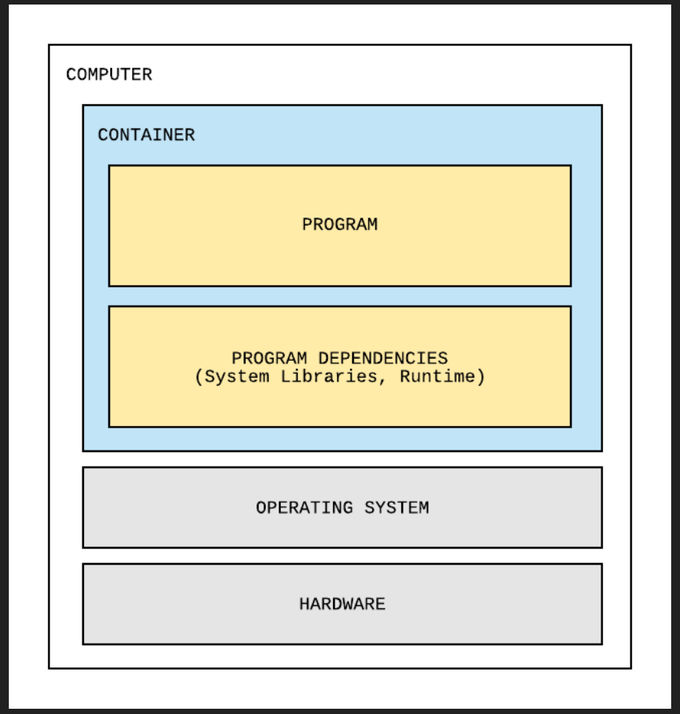
What is Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

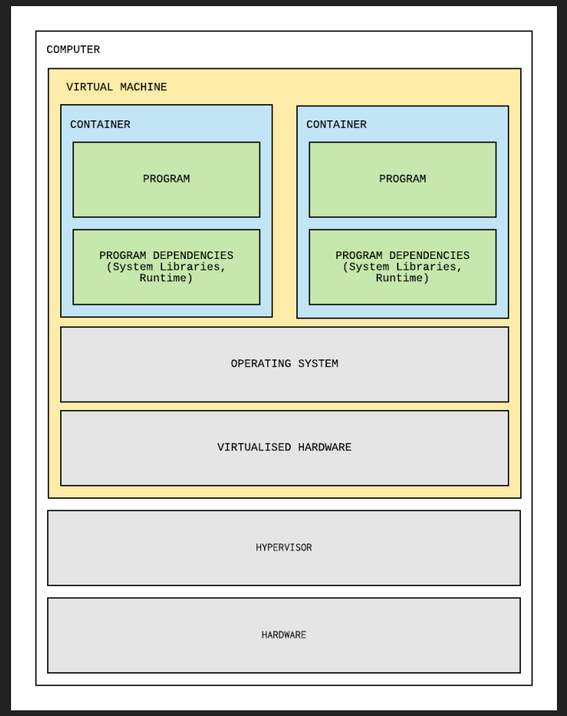
Containers

Isolate applications and all of their dependencies with none of the overhead of VMs.



Why containers?

- Provide a consistent, self-contained environment to develop and deploy.
- Allow us to launch and keep around multiple instances of our application cheaply.
- Can be run in VMs to maximise application density in our environments.


COMPUTER
VIRTUAL MACHINE
CONTAINER
PROGRAM
PROGRAM DEPENDENCIES (System Libraries, Runtime)
CONTAINER
PROGRAM
PROGRAM DEPENDENCIES (System Libraries, Runtime)
OPERATING SYSTEM
VIRTUALISED HARDWARE
HYPERVISOR
HARDWARE

Running Docker

Make sure Docker Desktop is running.

You can use your terminal to check if Docker is running:

```
# Get Docker version
$ docker version

# See running containers
$ docker ps
```

Launching Containers

```
# Example usage
$ docker run [flags] <image> [args]

# Actual usage
$ docker run docker/whalesay cowsay Hello!
```

```
< Hello! >
 --------
    \
     \
      \
                    ##        .
              ## ## ##       ==
           ## ## ## ##      ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
```

Container IDs

Every container has a unique ID we can use to refer to it instead of its name:

```
$ docker ps

CONTAINER ID        IMAGE     ...
31d16fa3edb2        ubuntu    ...
949316a32b9f        mysql     ...
```

Stopping containers

```
# Stop a container
$ docker stop <container_id>

# Remove a container
$ docker rm <container_id>

# Restart a container
$ docker restart <container_id>
```

Logging into our containers

Just pass in the `-it` flags (`--rm` deletes the container automatically on exit)

```
$ docker run --rm -it debian

root@dba9683049bd:/ # You're now in Debian!
```

exec

Run commands in running containers with the `exec` command

```
# Run a command in an already running container
$ docker exec -it <container_id> <command>

# Launch a bash shell in a container
$ docker exec -it <container_id> bash
```

Running Background Containers

You can run a container as a background task called a `daemon`.

All output is kept inside the container, and is non-interactive.

```
# Run container as foreground task
$ docker run docker/whalesay cowsay Hello!

# Run container as daemon (background task) - no output
$ docker run -d docker/whalesay cowsay Hello!
```

Running a Database Container

Pass environment variables to your container with the `-e` flag.

Choose which ports your container will use with the `-p` flag.

```
$ docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=pass mysql
```

```
# Connect to your containerised DB from your computer
$ docker exec -it <mysql_container_id> mysql -u root -p
```

Quiz Time! 🤓

True or false: You can create multiple containers from the same image.

Answer: `True`

What command would you run to get a list of running containers?

1. `docker exec`
2. `docker inspect`
3. `docker ps`
4. `docker service`

Answer: `3`

How would you permanently remove a running container?

1. `docker rm <c_id>`
2. `docker kill <c_id>`
3. `docker pause <c_id> && docker rm <c_id>`
4. `docker stop <c_id> && docker rm <c_id>`

`<c_id>` refers to a container ID.

Answer: `4`

Docker Images

- The basis of containers
- Consists of the system libraries, system tools and platform settings
- They are built-in layers which represent the commands of a Dockerfile
- When an image is run, it becomes a container

```
# List images
$ docker image ls
```

Dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image

Structure of a Dockerfile

`FROM` - what base image we are using

`COPY` - what we want in the image

`RUN` - any commands we want to run (apt-get install, update etc.)

`EXPOSE` - what ports we want accessible by other containers

`ENTRYPOINT` - configure what gets executed when starting a container

`CMD` - used to define the arguments to ENTRYPOINT

Image example - Ubuntu

Instructor to provide example `Dockerfile`.

Building a Dockerfile

This command will look for a file called Dockerfile in the directory you're in:

```
$ docker build .
```

Docker Pull

- Docker Hub is a registry of Docker images
- We can search for images that might come pre-installed with packages, modules etc.
- By default the pull will chose the image tagged with latest

```
# default
docker pull python

# specific version
docker pull python:3.7.4
```

Docker Push

- Pushes an image to a docker registry
- Creates a repository in your registry
- Tags the image with a version or name

Anyone can pull your docker image from your public docker hub registry

```
# Tag the image
docker tag \
  <name of local image>:<version> \
  <repository name>:<version tag>

# Push the image to your repository
docker push <repository name>:<version tag>
```

Port Binding

Allows you to connect ports in your containers to your host machine so you can talk to the services running in your containers as though they were running in your machine directly

With port binding, we can run services like web servers and databases in isolated containerised environments and access them as if they were local

```
$ docker run -p host_port:container_port <image>

$ docker run -d -p 8080:80 nginx
```

Volume Mounting

Lets us designate a directory in our local filesystem to be shared with a container

```
$ docker run -v <host_dir>:<container_dir> <container_name>

$ docker run -d -v \
  /home/user/projects/html-site:/usr/share/nginx/html nginx
```

Docker Compose

A tool that allows you to create and run multi-container Docker applications.

Using Docker Compose is a three-step process:

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment
3. Run `docker-compose` up and Compose starts and runs your entire app

Compose Example

Instructor to distribute `docker-compose.yml`

Quiz Time! 🤓

What is a Docker image?

1. A lightweight, standalone, executable package of software that includes everything needed to run an application.
2. A virtual machine running the host operating system.
3. A file that, when executed, creates a Docker container.
4. An alias for a container.

Answer: 3

What is a Dockerfile?

1. A lightweight, standalone, executable package of software that includes everything needed to run an application.
2. A text document that contains all the commands a user could call on the command line to assemble an image.
3. A virtual machine running the host operating system.
4. An alias for a container.

Answer: 2

Learning Objectives Revisited

- Explain the benefits of images and containers
- Identify commands to start, stop and log into containers
- Define the structure of a Dockerfile and explain how to write them
- Demonstrate how Docker Compose can create and run multi-container applications

Terms and Definitions Recap

Docker: A set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.

Dockerfile: A text document that contains all the commands a user could call on the command line to assemble an image.

Image: A serialized copy of the entire state of a computer system stored in some non-volatile form such as a file.

Docker Image: A file, comprised of multiple layers, that is used to execute code in a Docker container.

Terms and Definitions Recap

Virtualisation: An operating system paradigm in which the kernel allows the existence of multiple isolated user space instances.

Container: A standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Daemon: A computer program that runs as a background process, rather than being under the direct control of an interactive user.

Further Reading and Credits

- Docker Documentation
- What even is a container?
- First Steps with Docker (pic)
- Docker (pic)
- opensource.com (citation)
- Docker layers