School of Tech
part of accenture

Generation

Docker


docker

Overview

- A Brief History of Software Delivery
- An Introduction to Docker

- Commands to work with Docker containers
- Creating your own Docker containers
- Sharing your containers with the world

Learning Objectives

- Explain the benefits of images and containers
- Use commands to start, stop and log into containers

- Understand the structure of a Dockerfile and be able to write one
- Demonstrate how Docker Compose can create and run multi-container applications

Lots of follow-along

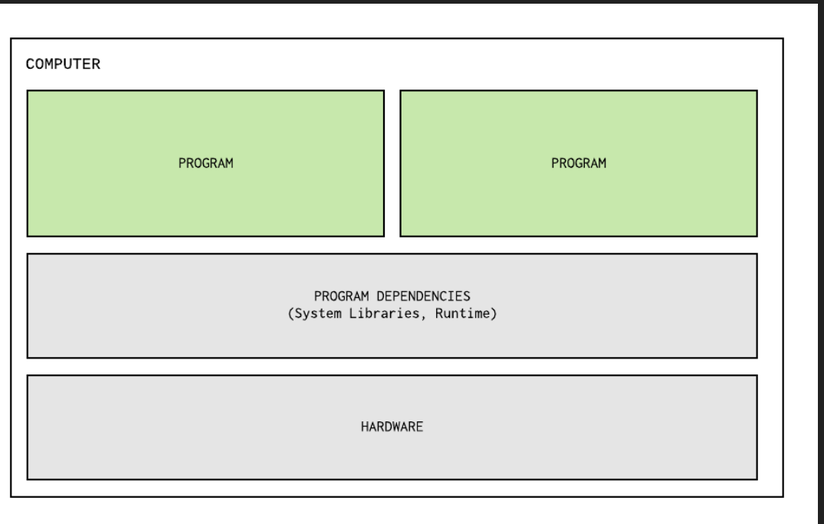
> In this session there will be lots of chance to follow-along with what the instructor does so you can practice too.

The History of Software Delivery

Or, how we ended up with Docker.

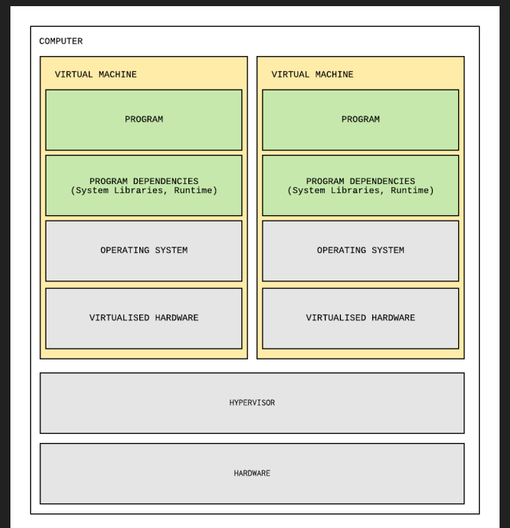
Multitasking operating systems (1960 - 1990)

Install many programs to disk and run them on demand



Virtualisation (1990 - 2010s)

Can isolate dependencies in individual VMs (virtual machines)

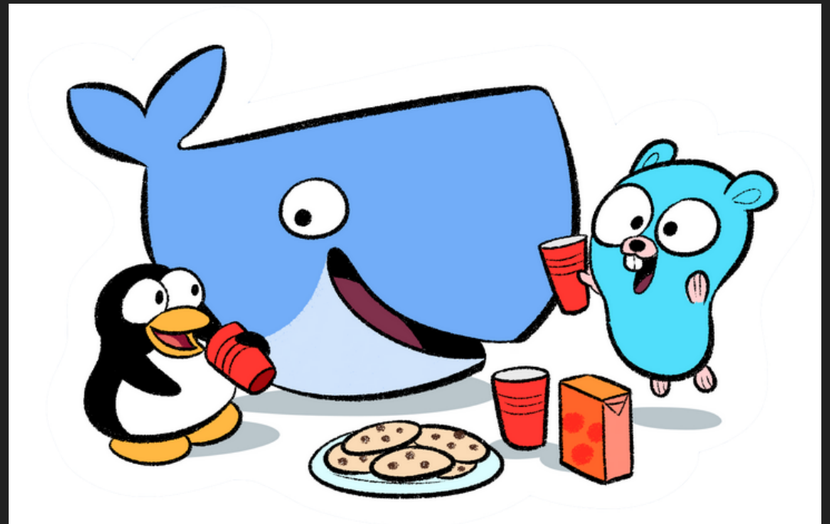


Virtualisation (1990 - 2010s) Pro's and Con's

- Each VM runs its own OS, using a lot of system resources

- Each VM emulates the underlying hardware, using yet more resources
- Still in many cases can save money on computer hardware
- A decent trade off when catering to a large number of users

Enter Docker (2013 - Now)



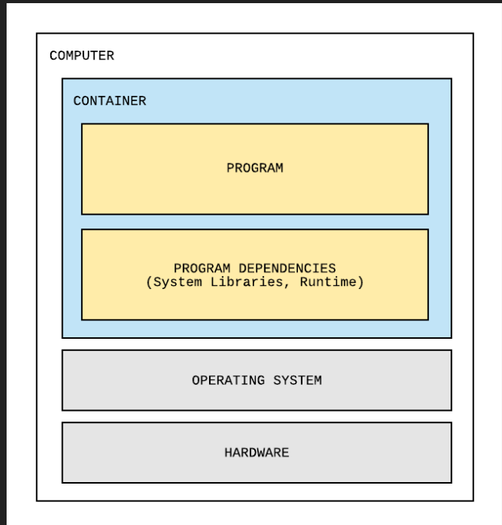
What is Docker?

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

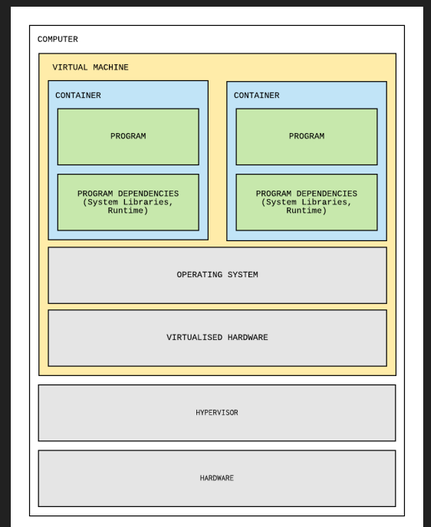
Isolate applications and all of their dependencies with none of the overhead of VMs.



Why containers?

- Provide a self-contained environment to develop and deploy applications

- Isolates application dependencies
- Lightweight in comparison to VMs
- Can easily 'transport' containers and run them on any machine


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

Quiz Time! 🤓

> We're about to have a go with Docker, but before that, a quiz!

True or False: A Container has an entire copy of an Operating System (OS) inside it.

Answer:

True or False: A Container has an entire copy of an Operating System (OS) inside it.

Answer: `False`

Which of the following statements is FALSE?

1. Containers can improve security if a running application is compromised or hacked
2. Containers often use more memory and CPU than Virtual Machines
3. Containers help prevent "Dependency Hell"

Answer:

Which of the following statements is FALSE?

1. Containers can improve security if a running application is compromised or hacked
2. Containers often use more memory and CPU than Virtual Machines
3. Containers help prevent "Dependency Hell"

Answer: 2

Running Docker

Make sure Docker Desktop is running.

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

```
# Get Docker version
$ docker version

# ps = 'process status', i.e. see running containers
$ docker ps
```

Container Images

Container images describe the type of container to run.

An image could contain software such as a web-server or database, or our own applications we have written.

Just like with Python libraries, there are thousands of images already available online which we can download and use.

An image is referenced by its name.

School of Tech
part of accenture

Launching Containers

Generation

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

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

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

Flags, Images and Arguments

- Flags - control options for docker itself and how it behaves when running the container
- Image name - tells Docker what to run
- Args (Arguments) - options passed into the program that you're running inside the container, like a webserver or database

Container vs Image

- The Docker Image is the binary definition of what we want to run
- The Docker Container is the running software

For example, on your laptop will be a file called `chrome.exe` or `word.exe`. These binary files are the definition of the program we want to run. This is like the static image.

When we open it (i.e. run it), we get a web browser or word processor. This is like the running container.

Container IDs

Every container has a unique ID we can use to refer to it instead of its name. We can see these with the command `docker ps`:

```
$ docker ps


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

We can also see stopped containers with `docker ps -all`

Stopping containers

When a container is running, we can stop it. We can also remove and restart them:

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

# Remove a stopped container
$ docker rm <container_id | name>

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

Exercise prep

> Instructor to give out zip file of `handouts`

Everyone please extract this, it has a folder `handouts` in it with files we will use.

Exercise - running vs stopped

In a second terminal run `docker ps -a` between these commands and note what it reports.

In your first terminal:

- Run `docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=pass --name my-sql mysql`
- Run `docker exec -it my-sql sh`
- Run `echo "hello I am inside my db"`
- Run `exit`
- Run `docker stop my-sql`
- Run `docker rm my-sql`

On the following slides we'll look at what all the commands mean.

Emoji Check:

Did you all mange to start and stop a container and see its state change. Could you do it again?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Logging into our containers

We need to pass in the `-it` flags (`--rm` deletes the container automatically on exit):

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

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

$ echo "hello I am in debian"
$ exit
```

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 654321 bash
```

Running Background Containers

When we run a container, by default it is `attached` to the current terminal - and we see the output from it.

You can run a container as a background task in `detached` mode.

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

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

# Run container as a detached - no output
$ docker run -d docker/whalesay cowsay Hello!
```

Seeing container logs

When a container is running in the background, we can still see its logs as follows:

```
# Print the logs from the container to your terminal:
$ docker logs [-f] <container_id>

# E.g.
$ docker logs 31d16fa3edb2

# And to "watch" the logs with the "follow" flag:
$ docker logs -f 31d16fa3edb2
```

Emoji Check:

Did you all manage to see some logs?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Port Binding

In order to connect to apps like your web browser (Chrome, Edge, etc), those apps have open ports to talk on. Our Containers do this too, so they can talk to each other. We can bind our machine's ports to those in the containers, so we can "talk" to them too.

This lets us 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>

# the port binding here is "-p 3306:3306"
$ docker run -d -p 3306:3306 \
    -e MYSQL_ROOT_PASSWORD=pass --name my-sql mysql
```

Using names not Ids

Using the Container ID is one way to control it and start and stop it.

More common is to set the name of your container, with the `--name` flag:

```
$ docker run --name <custom_name> <image>

# Here the name is "--name my-sql"
$ docker run -d \
    -e MYSQL_ROOT_PASSWORD=pass \
    --name my-sql \
    mysql
```

If we don't set the name, Docker will generate a random one - which is fine for temporary use, but not predictable for tests or scripting!

Environment variables

We can pass environment variables to our container with the `-e` flag, for example `-e MYSQL_ROOT_PASSWORD=pass`:

```
$ docker run -d -p 3306:3306 \
    -e MYSQL_ROOT_PASSWORD=pass \
    --name my-sql mysql
```

Exercise: Run a Database

- Start a database instance with the following command:

```
$ docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=pass \
    --name my-sql mysql
```

- Confirm the container is running using `docker ps`
- Try and start another instance with the exact same command.
- It won't work! Identify the problem from the error message.
- Change the startup command to fix the problem and allow another database to run at the same time
- Confirm both containers are running using `docker ps`
- Stop both containers

Emoji Check:

Did you mange to start and stop a database container?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Volume Mounting

This lets us designate a directory in our local filesystem to be shared with a container. For example, we can run some sql files when the database starts up:

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

$ docker run -d -v \
  /home/user/some-sql-files:/var/lib/mysql mysql

$ docker run -d -v \
  "$PWD/some-sql-files:/var/lib/mysql" mysql
```

Quiz Time! 🤓

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

Answer:

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:

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>`

(Here `<c_id>` refers to a container ID).

Answer:

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>`

(Here `<c_id>` refers to a container ID).

Answer: `4`

Components of a Docker Image

- 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 to assemble an image

Structure of a Dockerfile

`FROM` - what base image we are using

`WORKDIR` - the directory in the image that will contain our files

`COPY` - what files we want copied into the image

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

`EXPOSE` - what ports we want to make available outside our container

`ENTRYPOINT` - what gets executed when starting a container

`CMD` - a command to run when the container is started

Full Dockerfile reference information is here:
https://docs.docker.com/engine/reference/builder/

Dockerfile example

```
FROM python:3
WORKDIR /usr/src/app
COPY /app .
RUN pip install -r requirements.txt
CMD python ./app.py
```

Building a Dockerfile

A Dockerfile can be built into an image that we can then run.

This command will look for a file called `Dockerfile` in the current directory, and build it:

```
$ docker build .
```

To tag the image (give it a meaningful name) so we can use it later, use `-t`:

```
$ docker build -t my-application .
```

Exercise - build and run an image

> Build and run the sample python dockerfile at `handouts/Dockerfile`

- Open a terminal in the `handouts` folder
- Build it using `docker build -t my-app-image .`

- Run it using `docker run -d --name my-app-container my-app-image`
- Check its logs with `docker logs my-app-container`
- Stop it with `docker stop my-app-container`
- Remove it with `docker rm my-app-container`

Emoji Check:

Do you manage to build and run the container?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

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 choose the image tagged with latest

```
# default
docker pull python
# which the same as
docker pull python:latest

# specific version
docker pull python:3
```

Docker Tag

This "tags" or re-labels an image with a version or name:

```
# Tag the image
docker tag \

  <name-of-local-image>:<version> \
  <repository-name>:<version-tag>
```

This can be done at any time to "rename" an image. By default we always to this implicitly with the `-t` flag when we build an image.

Docker registry vs. repository

A registry is a docker system that stores images. There is one in your machine, and Docker Hub is one.

With a registry, images can be separated into repositories for different organisations

For example, let's look at The Python Repository on Docker Hub.

This is a repository of different images, all with different python versions.

Docker Push

- Pushes an image to a docker registry
- Creates a repository in your registry

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

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

Quiz Time!

> More docker questions!

What does docker push do?

1. Push image files from one folder to another in your machine
2. Push image files from Docker Hub to your machine
3. Push image files from your machine to Docker Hub

Answer:

What does docker push do?

1. Push image files from one folder to another in your machine
2. Push image files from Docker Hub to your machine
3. Push image files from your machine to Docker Hub

Answer: 3

What is a docker registry?

1. An organisation's list of images
2. Where dockerfiles go to get married
3. Internal storage where docker puts images
4. A folder in your machine

Answer:

What is a docker registry?

1. An organisation's list of images
2. Where dockerfiles go to get married
3. Internal storage where docker puts images
4. A folder in your machine

Answer: 3

Which command will name your image while building?

1. `docker tag my-name .`
2. `docker build -t my-name .`
3. `docker build --name my-name .`

Answer:

Which command will name your image while building?

1. `docker tag my-name .`
2. `docker build -t my-name .`
3. `docker build --name my-name .`

Answer: `2`

Emoji Check:

Do you feel, on a high level, you understand the key Docker concepts of images, and building, running and stopping containers?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Docker Compose

Compose is a tool that allows you to create and run Docker applications that use multiple images.

We can set up multiple containers side by side, and pass them environment variables to configure them.

Using Compose

Using Docker Compose is a three-step process:

1. Define your application with a `Dockerfile` so it can be reproduced anywhere

2. Define the services that make up your system in `docker-compose.yml` so they can be run together in an isolated environment
3. Run `docker compose up -d` to start docker and run your entire system

Compose commands

Compose can operate on your whole system (set of services) like `docker` can. It has the same set of commands:

- `build` - Build or rebuild services
- `up` - Create and start containers
  - Also `up -d --build` to build first, then `up`, in `detached` mode

- `stop` - Stop services
- `start` - Start services
- `rm` - Remove stopped containers
- `down` - Stop and remove containers & networks
- `pull` - Pull service images

...and a few more!

Compose Example

> See example file `handouts/docker-compose.yml`

In particular we have:

- Two services
  - Each with an image and container name specified
  - Each with port mappings defined
- Environment variables from the `.env` file

Exercise - Using Compose

- Open a terminal in the `handouts` folder
- Start all services with `docker compose up -d`
- Use `docker ps -a` to see what is running

- Check the logs of both containers
- Stop everything with `docker compose stop`
- Remove everything with `docker compose rm`

Emoji Check:

Did you manage to run both services using the compose file?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Quiz Time! 🤓

What is a Docker image?

1. An isolated environment to contain a running application and its dependencies.
2. A virtual machine running the host operating system.
3. A template from which a Docker container can be created.
4. An alias for a container.

Answer:

What is a Docker image?

1. An isolated environment to contain a running application and its dependencies.
2. A virtual machine running the host operating system.
3. A template from which a Docker container can be created.
4. An alias for a container.

Answer: 3

What is a Dockerfile?

1. An isolated environment to contain a running application and its dependencies.
2. A text document that contains all the commands needed to assemble a Docker image

3. A virtual machine running the host operating system.
4. Another word for a container

Answer:

What is a Dockerfile?

1. An isolated environment to contain a running application and its dependencies.
2. A text document that contains all the commands needed to assemble a Docker image

3. A virtual machine running the host operating system.
4. Another word for a container

Answer: 2

Offline Exercise: Dockerise your Mini Project!

- Create a Dockerfile to build a container image for your mini project
- Name the image `<yourname>-miniproject`
- Run the Dockerfile as `<yourname>-miniproject`

- Verify the image is created using the `docker image ls` command
- Run your mini project from the image
- Add the dockerfile to source control as part of your Mini Project and commit it to GitHub
- Add the build and run commands to your README file

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.

Overview - recap

- A Brief History of Software Delivery
- An Introduction to Docker

- Commands to work with Docker containers
- Creating your own Docker containers
- Sharing your containers with the world

Learning Objectives - recap

- Explain the benefits of images and containers
- Use commands to start, stop and log into containers

- Understand the structure of a Dockerfile and be able to write one
- Demonstrate how Docker Compose can create and run multi-container applications

Further Reading and Credits

- Docker Documentation
- What even is a container?
- First Steps with Docker (pic)

- Docker (pic)
- opensource.com (citation)
- Docker layers

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively