

# **CAB432 Cloud Computing**

## **Lecture 2: Container as a Service**

Faculty of Science



Queensland University of Technology

CRICOS No. 00213J



How to actually use a container

# DOCKER IN DEPTH



a university for the **real** world<sup>®</sup>

# What is Docker?

- A popular CaaS technology on Linux
  - Backed with commercial offering for container image hosting, versioning and sharing: <https://docker.com>
  - Far from the only game in town – see OCI later in the lecture.
  - Containers not the only abstraction used (see Kubernetes later).
- More technically:
  - Yet another “daemon” (OS-level software service) on 64 bit Linux
  - Command-line tools to run containers, create images, etc.
  - Can be run inside a Linux OS that is itself running on a VM
  - We do this all the time:
    - VirtualBox (Oracle), VMWare Player, Parallels, etc.

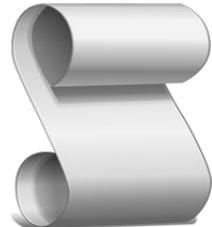
# Docker Terminology

- **A container:**
  - a running (stopped, finished, ...) instance of (an) application(s)
  - installed on top of some “base image”.
  - Isolated from other containers and from the guest OS (but may interact with these through well-defined interfaces)
- **An image:**
  - a “pre-canned” software stack loaded from a repository
  - <https://dockerhub.com> is the usual choice
  - An image is like a “blueprint” for containers
  - The cookie-cutter model - deploy as many as we want

# Docker Terminology

- **Dockerfile:**
  - Script to build a Docker image
  - Usually start with a base OS image
  - Execute a sequence of commands
  - Typically installation and configuration tasks for your stack
  - [Examples to come]
- See also Docker compose
  - Specify services and configuration for multi-container apps
  - YAML file for the configuration
  - You are free to use it but we won't teach it (yet)
  - <https://docs.docker.com/compose/>

# Containers and Images



Dockerfile

`docker build -t image`



User manually entering  
Docker commands

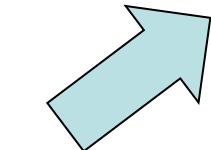
`docker create image`



Image repository  
(e.g., dockerhub.com)

`docker pull image`

Docker image



`docker run image`



Container(s)



Getting started

# RUNNING DOCKER



a university for the **real** world<sup>®</sup>

# Ubuntu Basics

- In this lecture, we use Ubuntu 18.04 to work with Docker
  - Any modern Linux distribution will also work
  - Some commands may be different (different software package manager, different launch daemons, ...)
- Ubuntu
  - Images available on Azure and AWS
  - Can be installed inside Oracle VirtualBox: <https://virtualbox.org>
  - Get the ISOs at: <http://www.ubuntu.com/download/desktop>
  - [points to the latest LTS release, presently 20.04. 18.04 also easily found at the site.]

# Ubuntu Basics

- Ubuntu uses the “Advanced Packaging Tool” software package manager:

`apt-get install somepackage`

`apt-get uninstall someinstalledpackage`

`apt-cache search somepackage`

...

- Sometimes we will use elevated privileges

– `sudo apt-get install somepackage`

...

# Ubuntu Basics (contd.)

- Installing Docker (use the update if it doesn't work):

```
sudo apt update
```

```
sudo apt install docker.io
```

(then confirm all questions with “Y”)

- Launching the Docker daemon
  - Ubuntu 18.04 uses the “systemd” launch daemon
  - Some older Linux distributions use “SysV init” or “upstart”

```
sudo systemctl start docker
```



# Ubuntu Basics (contd.)

- Launching the Docker daemon
  - Start on boot: <https://dockr.ly/2LryNH3>

```
sudo systemctl enable docker
```

```
sudo systemctl disable docker
```

# Ubuntu Basics (contd.)

- Organising the Docker group
  - Usually the docker group will be created on installation
  - If you want to avoid using the sudo prefix all the time, add your username to the group.

```
sudo usermod -aG docker $USER
```

- For subsequent logins you can then just use the docker commands without sudo

```
docker ps
```



# Principles of Docker

- Core idea: manages and runs a software stack in a reproducible and robust manner
- Software stack: a Docker image
  - Typical setup: create your own image on top of another image
  - Also possible: simply use a 3<sup>rd</sup> party image “as is”
- Creating an image:
  - Create a **Dockerfile**: script that configures an image and installs software inside it
  - Also possible: build up image by creating a “blank” image, then launching a container and “committing” versions of that image

# Principles of Docker

- **Image versioning:**
  - Versioning works the same as for source code
  - (including reverts to earlier states)
- **Running a container: instantiate an image**
  - Cookie cutter: instantiate into arbitrarily many containers
  - Containers for the same image can run on different servers

# Principles of Docker

- Container lifecycle operations:
  - Pause/resume container
  - Attach to a container
  - List all running containers
  - Kill (stop) a container
  - Restart a stopped container
  - ...



Standard commands

# BASIC DOCKER



a university for the **real** world<sup>®</sup>

CRICOS No. 00213J

# Basic Docker Commands

- Retrieve logs from container

```
sudo docker logs [OPTIONS] container
```

- “-f”: Follow log output
- “-tail=...”: Number of lines to show (from end of log)
- ...
- Examples:

```
sudo docker logs -tail=10 foo  
sudo docker logs -f foo
```

# Basic Docker Commands

- List all containers on the local system:

```
sudo docker ps [OPTIONS]
```

- “-a” show all containers (also stopped ones)
- “-l” show only the latest created container
- ...
- Example:

```
sudo docker ps -a
```



# Basic Docker Commands

- Show processes inside container:

```
sudo docker top container [ps command options]
```

- Example:

```
sudo docker top foo aux
```



# Basic Docker Commands

- Retrieve metadata about a container or image

```
sudo docker inspect container|image
```

- Examples:

```
sudo docker inspect foo  
sudo docker inspect ubuntu:16.04
```



# Basic Docker Commands

- Start a previously stopped container:

```
sudo docker start [OPTIONS] container
```

- “-a” attach standard output/error
- “-i” attach standard input

- Example:

```
sudo docker start -it cab432
```



# Basic Docker Commands

- Remove a container:

```
sudo docker rm [OPTIONS] container
```

- “-f” forces removing a running container
  - ...

- Example:

```
sudo docker rm cab432
```



# Basic Docker Commands

- Stopping a running container:

```
sudo docker stop [OPTIONS] container
```

- “-t ...” defines a “grace period” in seconds to wait until SIGTERM follows SIGKILL

...

- Example:

```
sudo docker stop cab432
```



# Basic Docker Commands

- Attaching to a running container:

```
sudo docker attach [OPTIONS] container
```

[attach local I/O streams to a container already running]

- Example:

```
sudo docker attach cab432
```





Building an Image

# THE DOCKERFILE



a university for the **real** world<sup>®</sup>

# Building an image

- Images can be built manually:
  1. Create a container (`docker run` or `docker create`) then log into it (i.e., launch a shell)
  2. Install software on the container's file system (e.g., using `apt-get` on Ubuntu)
  3. Commit the changes to create a new image (`docker commit`)
- Better: build image by running the `docker build` command with a script (Dockerfile)

```
sudo docker build [OPTIONS] PATH
```

- PATH is the directory where the Dockerfile is located
- “-t” name of the repository for the new image

# Building an Image

- docker build command uses all files inside the specified directory as the “build context” for the image
  - Add a .dockerignore file to specify exceptions (i.e., files you do NOT want to be part of the build context for the new image)
  - The files of the build context are copied to the Docker daemon, which can take a lot of time
  - Best practice is to put the Dockerfile into an empty directory
  - Then add only those files there that are really needed
- A repository/tag can be specified at which to save the new image:

```
sudo docker build -t qut/cab432demo .
```

# Complete Dockerfile example

```
FROM ubuntu:16.04
```

```
MAINTAINER Soeren.Balko@qut.edu.au
```

```
RUN apt-get update && apt-get -y install curl
```

```
RUN curl -sL https://deb.nodesource.com/setup |  
bash -
```

```
RUN apt-get -y install nodejs
```

```
CMD ["console.log('Hello nodejs world!');"]
```

```
ENTRYPOINT ["/usr/bin/nodejs", "-e"]
```

# Dockerfile syntax

- General syntax:

```
# comment  
INSTRUCTION arguments
```

- First instruction must be FROM base\_image, e.g.:

```
FROM ubuntu:16.04
```

# Dockerfile syntax

- Instruction to run a command (e.g., for installation) when the image is created (NOT when a container for that image is run):

```
RUN <command>
```

```
RUN ["executable", "param1", "param2"]
```

→ RUN apt-get -y install curl

# Dockerfile syntax

- In order to expose a network port, use the EXPOSE command:

```
EXPOSE 3000
```

When running a container with this image with the “-P” parameter, Docker will expose port 3000 on some port on the guest OS

Use port mappings to control access – see later

# Dockerfile syntax

- In order to copy files into the file system of the new image, use the ADD command:

```
ADD <src>... <dest>  
ADD [<src>, ... <dest>]
```

Source files are relative to the location of the Dockerfile and may contain “wildcards”

Surprisingly common gotcha for novices – follow the examples carefully.

# Dockerfile syntax

- When running a Docker container for the image, an ENTRYPOINT command can be specified that is run inside the container:

```
ENTRYPOINT ["executable", "param1", "param2", ...]  
ENTRYPOINT command param1 param2 ...
```

Additional command line parameters can be specified when doing a “docker run”

Alternatively, default parameters can be specified using the CMD command



# Complete Dockerfile example

```
FROM ubuntu:16.04
```

```
MAINTAINER Soeren.Balko@qut.edu.au
```

```
RUN apt-get update && apt-get -y install curl
```

```
RUN curl -sL https://deb.nodesource.com/setup |  
bash -
```

```
RUN apt-get -y install nodejs
```

```
CMD ["console.log('Hello nodejs world!');"]
```

```
ENTRYPOINT ["/usr/bin/nodejs", "-e"]
```

# Building an image from a Dockerfile

1. Build the image:
  - Go into the directory where the Dockerfile is located
  - Run: `sudo docker build -t qtech/test .`
2. Run a container off the new image “qtech/test”:
  - Without arguments: `sudo docker run -it qtech/test`
  - With arguments: `sudo docker run -it qtech/test "console.log('blah');"`

# Final Comments on Docker

- Our focus in the early stages of this course will be on the creation and deployment of individual containers
- All of the majors support deployment of Docker containers and equivalents under OCI (see later).
- So the value add from the cloud vendors is then to increase the convenience of deployment by integrating the process with the creation and maintenance of a cluster of VMs.
- We won't require that you use these in CAB432, but you are free to use them as part of Assignment 2 if you wish