Birkbeck
UNIVERSITY OF LONDON

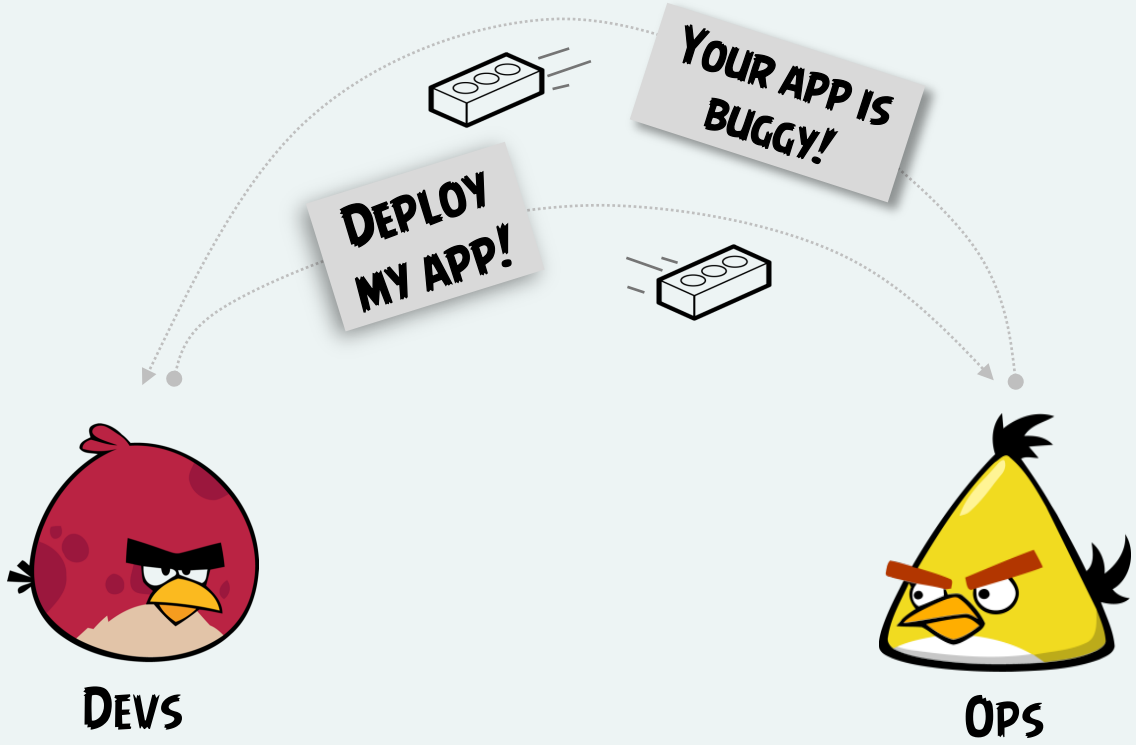
Cloud Computing Workshop
Session 2: Containers

Dimitris Kargatzis

Stelios Sotiriadis

Plan for today

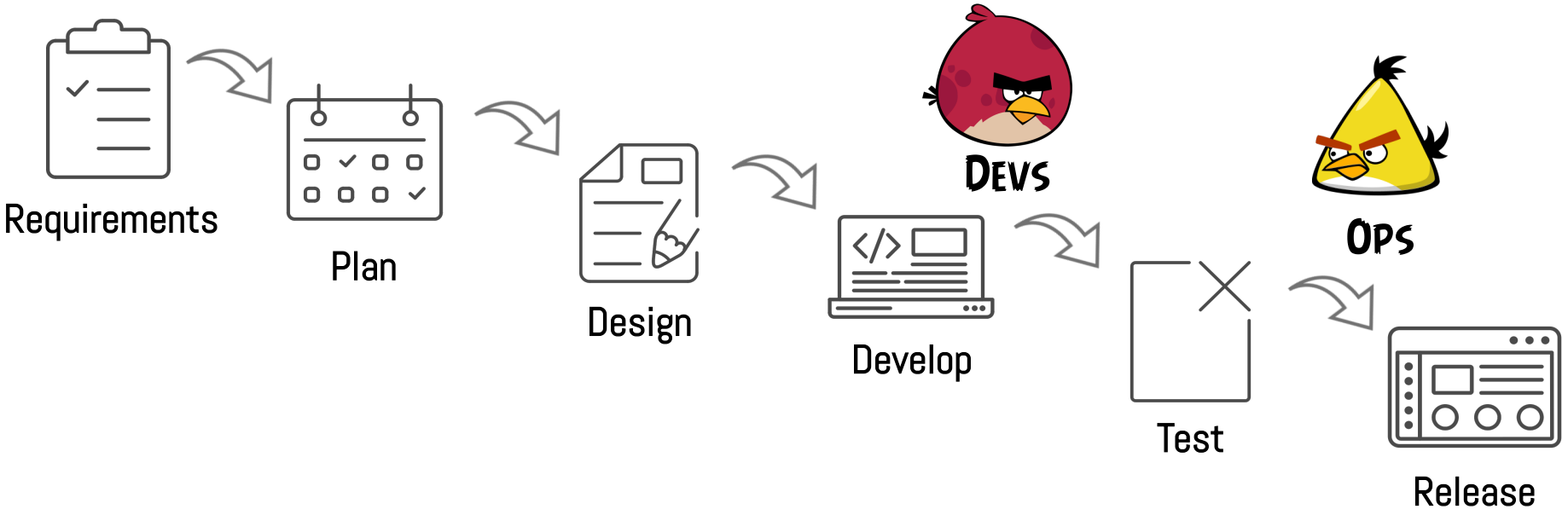
- Lecture part 1:
  - What is DevOps?
  - What are the best practices for software development?
- Lecture part 2:
  - What is a container?
  - What are the difference between VMs and Containers?
  - What is Docker?
- Lab session:
  - Learn Docker
  - Using Docker to deploy containers



Development and Operations

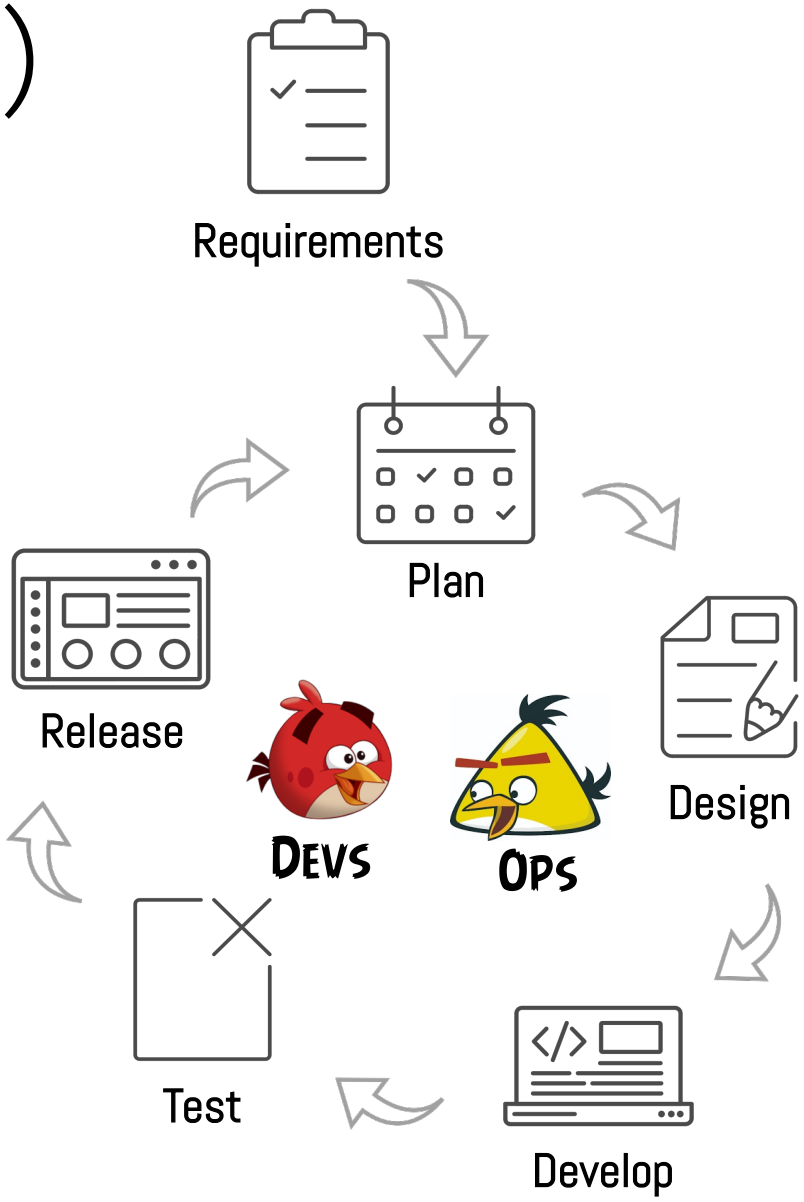
Software Development Models (1)

- In waterfall, we develop systematically, from phase to phase in a downward fashion
- Suited for smaller projects where requirements are well defined



Software Development Models (2)

- Agile adopts iterative development
- Each incremental part is developed over an iteration
  - Each iteration is intended to be small and easily manageable and that can be completed within a couple of weeks only
- Each iteration software is developed and deployed to the customers



What is DevOps?

- **Development** and **operations** are the key two teams that every software development company needs!
- Both Devs and Ops teams are composed by highly technical people
- Historically communication have been manual and slow
- **DevOps** refers to the merging of roles between Devs and Ops

Devs and Ops roles

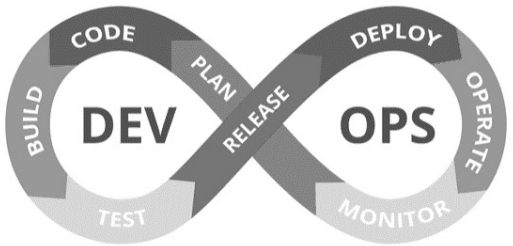
- Devs develop
  - Application performance e.g. using cache, databases etc.
  - End use analytics e.g. user latency
  - Code quality
  - Code level errors e.g. fixing bugs
- Ops deploy and maintain
  - Application availability e.g. up and running
  - Performance, e.g. monitor application CPU usage
  - End user complaints e.g. fix problems
  - Performance analytics e.g. alerts and troubleshooting

What is DevOps?

- **Development** and **operations** are are no longer “siloed.”
  - These two teams are merged into a single team
  - The **engineers** work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.
    - **The skills we develop in this class**
- They use a technology stack and tooling which help them operate and evolve applications quickly and reliably.

DevOps

- DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity.



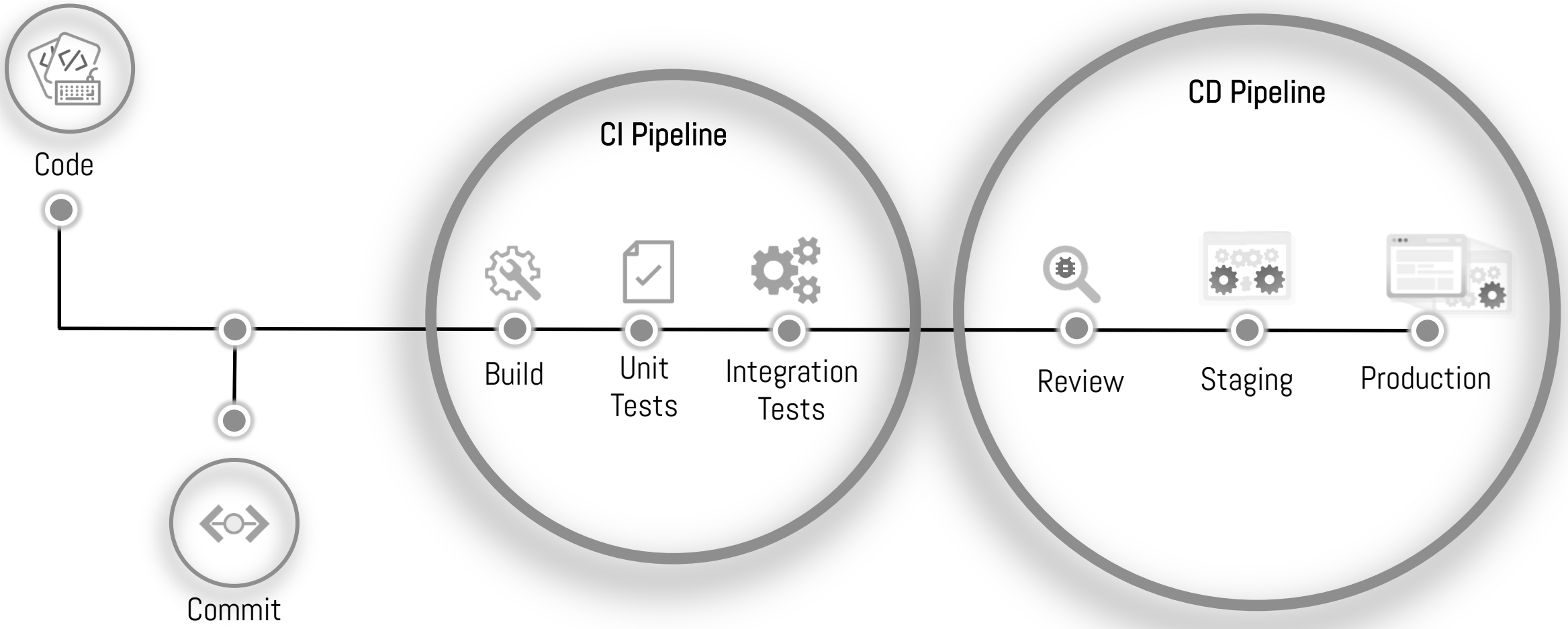
Benefits of DevOps

- Speed of Development:
  - Faster software development, teams take ownership of software
- Rapid Delivery of Products:
  - Increase the frequency and pace of releases so you can innovate and improve your product faster.
- Reliability of Products:
  - Use practices like continuous integration and continuous delivery to test functionalities
- Scaling and use of Cloud Computing:
  - Automation and consistency help you manage complex or changing systems efficiently.
- Improved Collaboration
  - Build more effective teams emphasising values such as ownership and accountability.

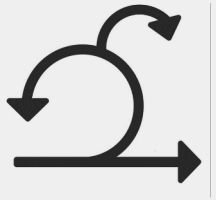
DevOps key practices

- **Continuous Integration (CI)**
  - Merging all developers' working code copies to a shared repo several times a day.
- **Continuous Delivery (CD)**
  - Teams producing software in short cycles and software can be reliably released at any time

DevOps: from your laptop to production



Agile vs CI/CD vs DevOps



Agile

focuses on
processes

highlighting
change

while accelerating
delivery

CI/CD

focuses on
software-defines life cycles

highlighting
tools

while accelerating
automation

DevOps

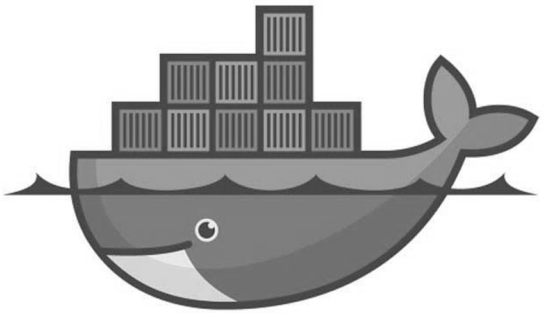
focuses on
culture

highlighting
roles

while accelerating
responsiveness

DevOps summary

- A way to evolve and improve products taking advantage of modern:
  - Project management approaches
    - Agile methods
  - Software engineering methods
    - Microservices etc.
  - Modern systems for CI/CD
    - More to come soon (Infrastructure as a Code)
- What is the enabling technology?
  - **Containers**



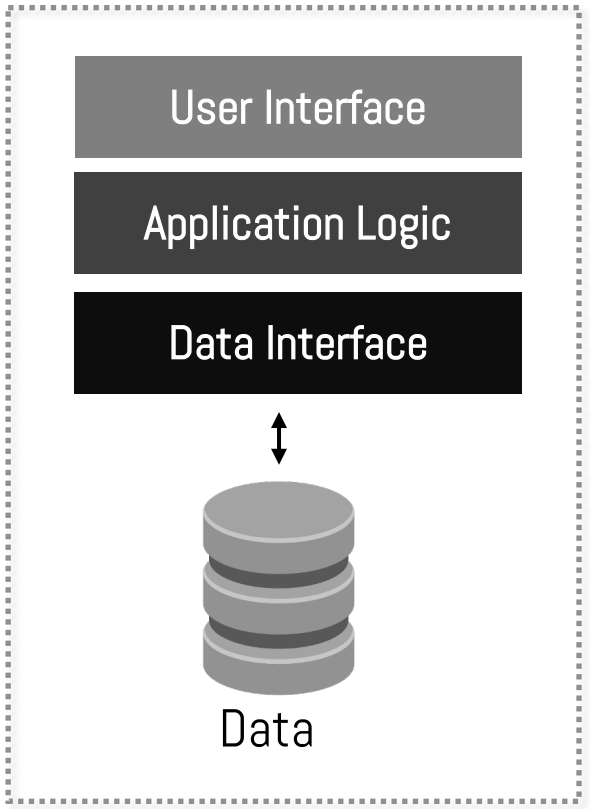
Application Containerisation

Is there a technology to…

- …develop and deploy code and move from your computer to stagging and then to production system, in a fast and efficient manner?
  - Yes! Let's use containers!

Containers...

- The bad old days...
  - Applications are the heart of a system
  - If an app breaks, businesses break
  - One app in one server
- Do we need a new app?
  - We need to buy a new server...
  - Tragic waste of company capital and environmental resources



Hello VMWare!

- VMWare Inc. gift to the world
  - VMware presents its first product, Workstation 1.0, at DEMO 1999.
  - Release of VMware ESXi type-1 hypervisor to deploy virtual computers (March 23, 2001)
    - Read this → VMWare history
- Overnight world changed to a better place!
- We could squeeze multiple applications in one server
  - **...the virtual machine ☺**

VMs changed IT industry, but …

- VMs are far from perfect ☹
  - Slow to boot, need installation of OS, libraries, too manually tasks…
  - Migration is a headache
  - Portability is an issue

*Portability: The ability of software to be transferred from one machine or system to another.

Hello Containers!

- Google used containers for years to address the shortcomings of VMs
  - Containers = VMs – OS
  - Fast to boot, portable and good for DevOps

- Linux containers (LXC):
  - A Linux container is a set of processes isolated from the system, running from a distinct image that provides all the files necessary to support the processes
- Docker:
  - Magic!
  - Software that creates, manages and orchestrates containers
  - Docker Inc. is the company created the technology and solutions

Container?

- What is a container?
  - It is an Operating System virtualisation method to deliver software in packages
- Applications are self-contained and easy to move around
- Containers are not replacing virtualization!
- But, it is an elegant way to deploy applications and services
  - Easy to distribute applications

So how does this work?

- Why do we need containers?
- Why containers are ideal for DevOps?
- What is Docker?

Nick is a software developer...

- Nick developed the a MiniPost REST API using Node.js
- Nick developed the app in his MacBook using a MongoDB service
- Nick used different libraries:
  - Node.js (v16.13.1)
  - NPM (v8.1.2)
  - Mongoose (v6.2)
  - ...
- These are called dependecies

```
"dependencies": {
    "body-parser": "^1.19.1",
    "express": "^4.17.2",
    "mongoose": "^6.2.0",
    "nodemon": "^2.0.15"
  }
```

- The app is ready! It works perfectly!
- Nick felt amazing for just a minute, then he felt desperate!
  - The app is still in his laptop...

Why Nick is in trouble?

- Nick developed the app in his laptop, now he has to deploy it in a Cloud server e.g. Google Cloud Platform.
- What are the possible problems?
  - Setup the application again & install software and libraries
  - What about incompatible libraries?
    - Nick developed the app in his MacBook…, does it work the same in Ubuntu?
      - **No!!!**
  - Incompatible dependencies
  - Can we be sure that everything runs?

Todays problem?

- How to get software to run reliably when moved from one computing environment to another.
  - From a developer's laptop to a test environment, from a staging environment into production?
  - From a physical machine in a datacentre to a virtual machine in a private or public cloud?

Docker

- Docker creator Solomon Hykes.
  - "You're going to test using Python 2.7, and then it's going to run on Python 3 in production and something weird will happen. You'll run your tests on Debian and production is on Red Hat and all sorts of weird things happen."
  - How to solve such problems?

- Use Docker!
  - 1200 Docker contributors
  - 100K Dockerized applications
  - 3-4 million devs using Docker
  - 70% of enterprises are using Docker

Containers

- What is a container?
  - A container is 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.
  - It is another way to deploy applications

- It is ideal for DevOps: CI/CD
- Key features of Containers?
  - **AGILITY, PORTABILITY & SCALABILITY**

What is a Linux Daemon?

- A daemon is a long-running background process that answers requests for services
- The term originated with Unix, but most operating systems use daemons in some form or another
  - In Unix, the names of daemons conventionally end in "d".
  - Some examples include httpd , nfsd , sshd and other
  - At a particular instant of time, it can be either running, sleeping, or zombie (a process that completed its task, but waiting for its parent process to accept the return value).

Docker Platform (terminology)

- **Docker Engine**: Open-source containerization technology for building and containerizing your applications.
  - **Docker daemon**: The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
  - **Docker CLI**: (command line interface): A command line tool that lets you talk to the Docker daemon
- **Docker Hub**: a service provided by Docker for finding and sharing container images with your teams

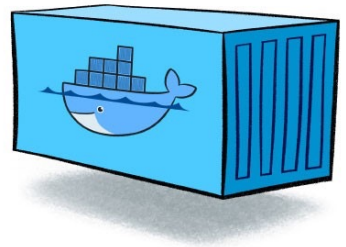
The Docker technology

- Three key parts:
  - The runtime: Lowest level, responsible for starting and stopping containers
    - **runc**
      - Interface with the OS to start/stop containers
    - **containerd**
      - Manager container lifecycle (pushing/pulling, networking etc.)
  - The daemon (engine): Runs and manages images, containers, provides a REST API
  - The orchestrator: Manages clusters of Docker nodes
    - These clusters are called swarms (native technology called Docker Swarm)

What about Kubernetes?

- Alternative docker orchestrator…
  - Most people choosing Kubernetes instead of Docker
- What is Kubernetes?
  - Open-source Google project
  - Today is considered the de facto orchestrator of containerised apps
  - We will talk about Kubernetes in Bonus class

Containers: Key features?



- Portability
  - Ability to move applications and data from one system to another with minimal disruption
- Scalability
  - Ability of a system resources to grow to accommodate demand
- Availability
  - Ability of a system or system component to be continuously operational

Why to use containers?

- Agile development
  - Deliver value to customers faster
- Scalable systems
  - Containers are easy to scale
- Cost optimization (more cheap)
  - Pack a lot of containers in the same environment (e.g. a VM)
- Portability:
  - Develop in one your laptop, deploy anywhere

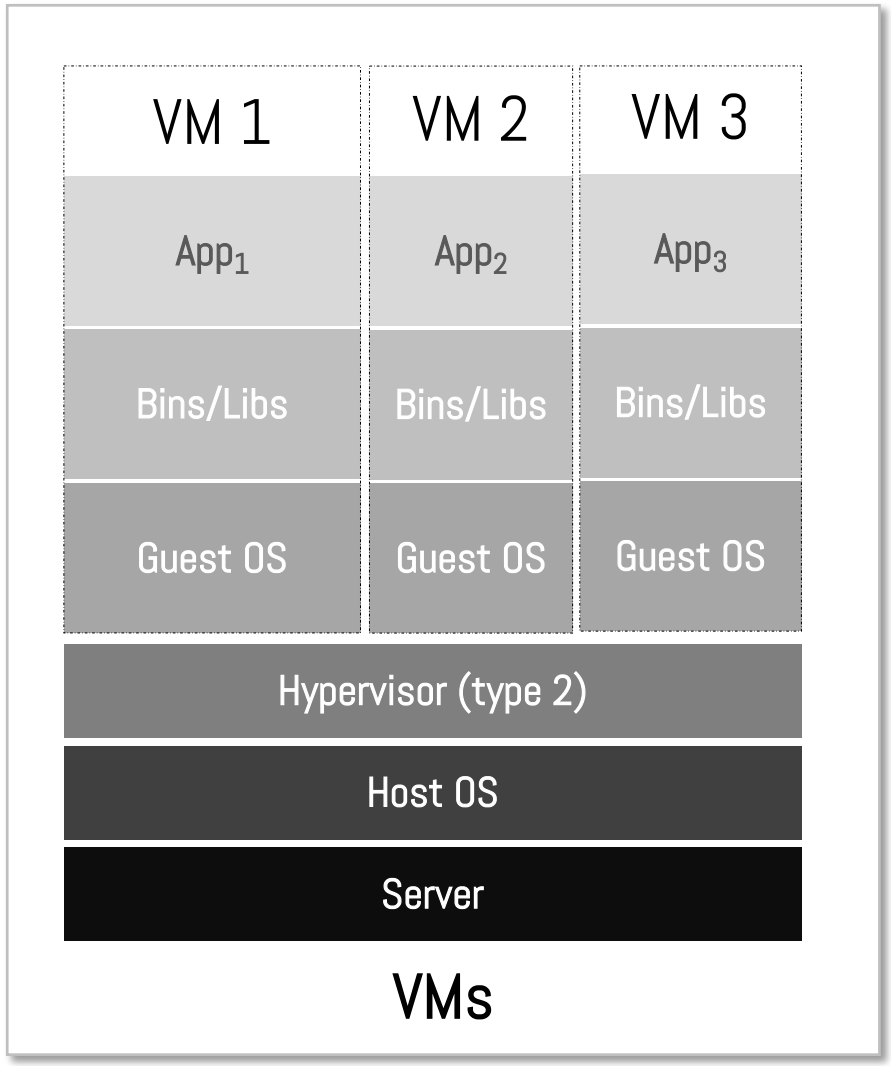
Container's cons

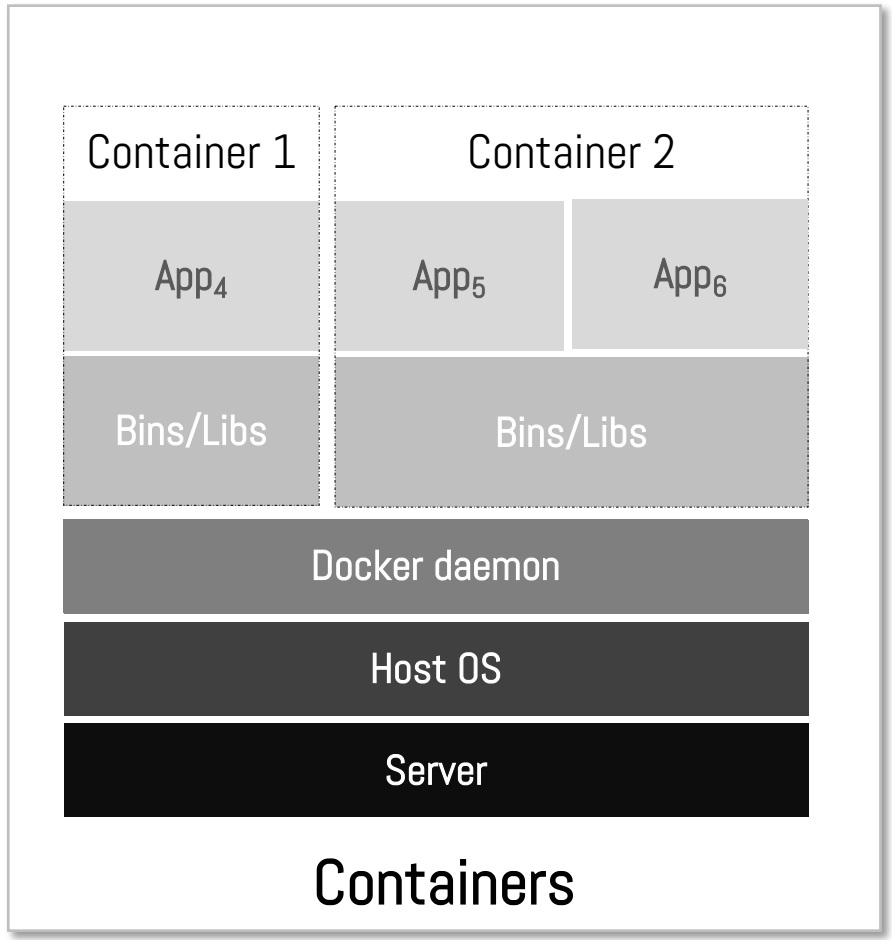
- Containers consume resources more efficiently than VMs
  - But containers are still subject to performance overhead due to overlay networking, interfacing between containers and the host system and so on.
- **Persistent data storage is complicated**.
  - By design, all the data inside a container disappears forever when the container shuts down, unless you save it somewhere else first.

VMs and Containers

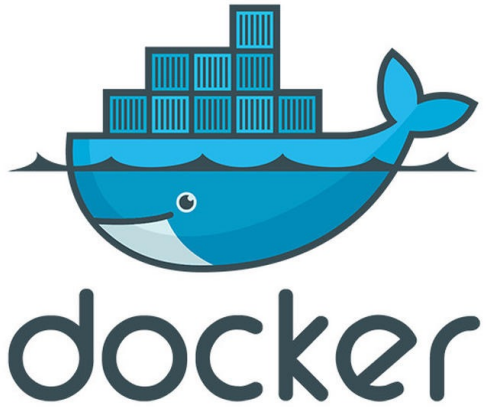
- Containers do not replace VMs
- They do different things
  - Use containers when you want to have data and application that is easy to move
- Container can run inside VMs
- Or, you can deploy directly on a server
- Usually, we run containers inside VMs

Virtualization vs Containerization





Container technology



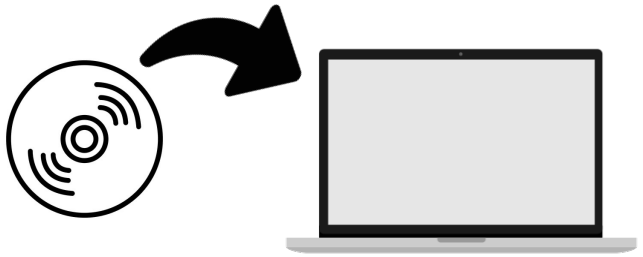
- Docker is the largest player
- Released as open source in 2013
- Leading container ecosystem
- Raised $190M in past years
- Currently most popular container technology…
- Orchestration frameworks:
  - Kubernetes and Mesos
  - Proprietary systems

Docker images vs Docker Container

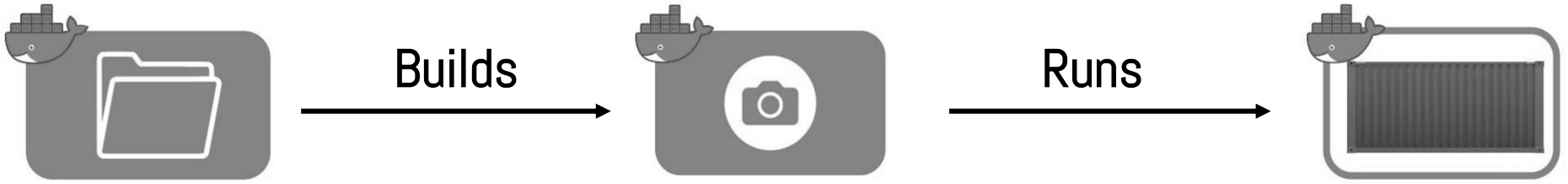
- Docker image

  - A Docker image is **a read-only template** that contains a set of instructions for creating a container.
  - It provides a convenient way to package up applications/configurations.

- Docker container
  - A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
  - Containers are running instances of an image



Docker: Building and running an app



Dockerfile | Docker image | Docker container

1

A **document** containing all the commands to create an image, e.g. install node.js, express, your code etc.

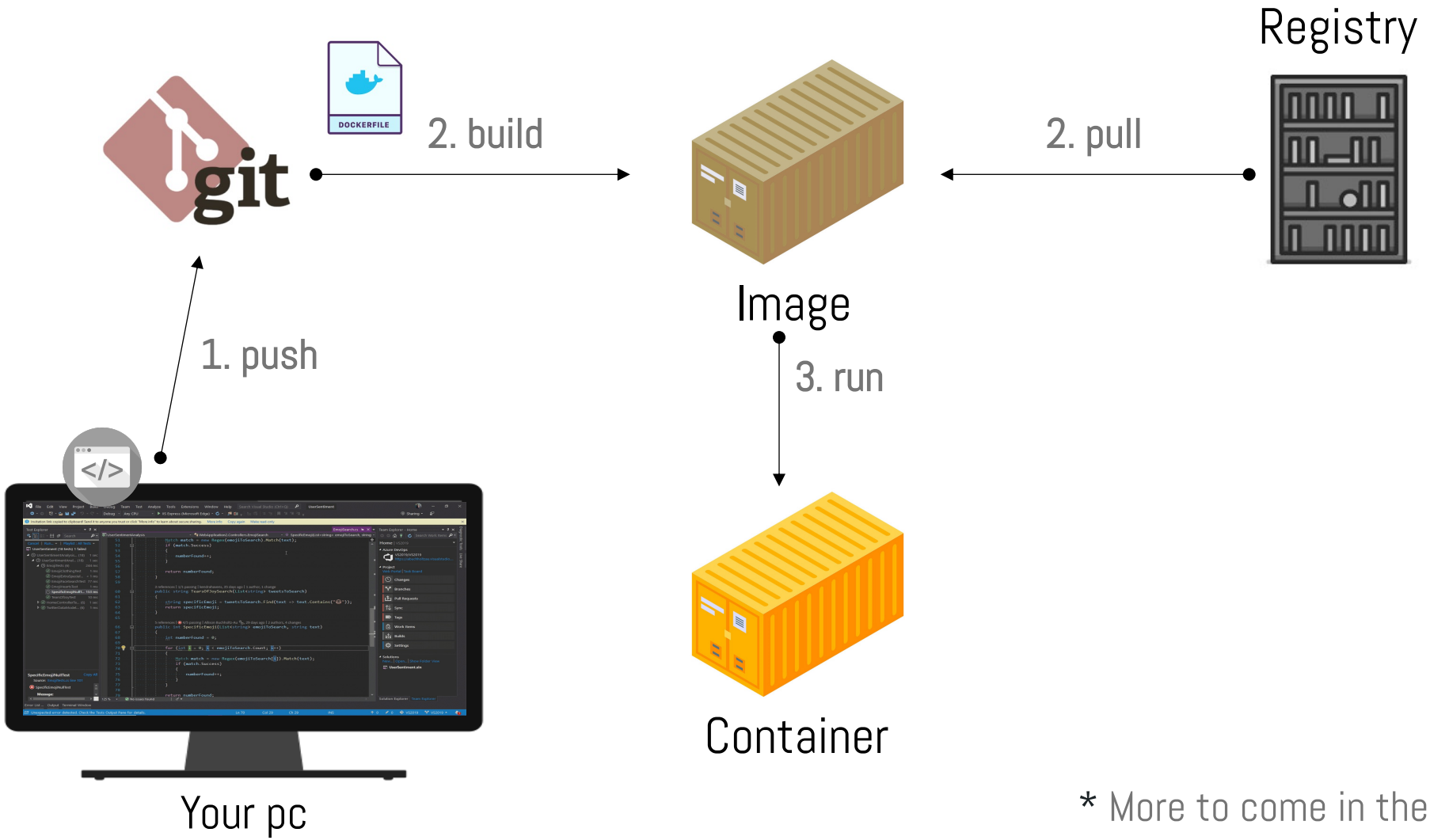
2

A read-only template that packages up your code and preconfigured environments e.g., node.js

3

A running instance of your image including: code, runtime, system tools, system libraries and settings.

Docker workflow followed in lab



* More to come in the next classes...

Comparison (1/2)

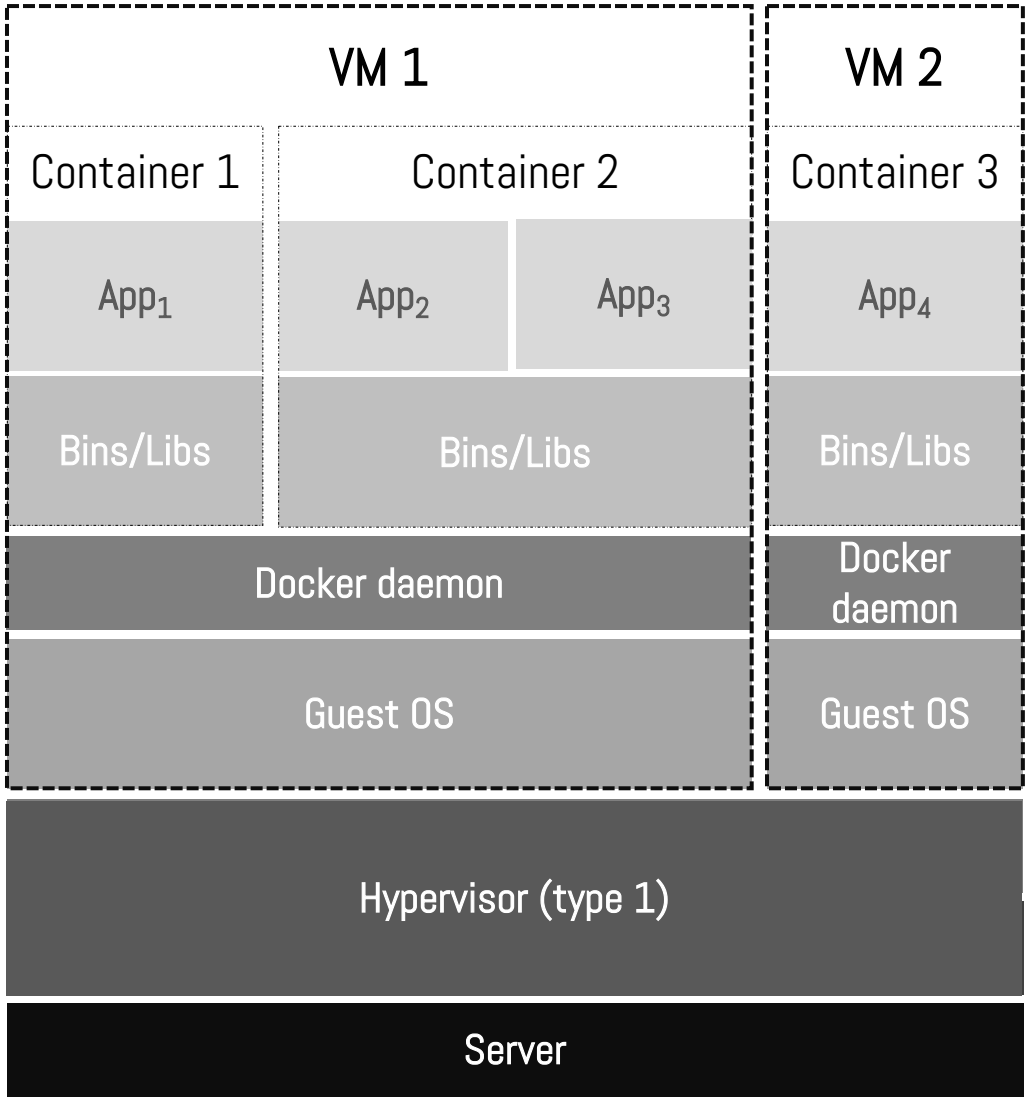
Feature	Virtual machine	Container
Isolation	▪ Provides complete isolation from the host operating system and other VMs. ▪ This is useful when a strong security boundary is critical. ☺	▪ Typically provides lightweight isolation from the host and other containers, but doesn't provide a strong security boundary as a VM ☹
Operating system	▪ Runs a complete operating system including the kernel, so requiring more system resources. ☹ • CPU, memory, and storage.	▪ Runs the user mode portion of an operating system, and can be tailored to contain just the needed services for your app. ☺ • Uses fewer system resources.
Guest compatibility	▪ Runs just about any operating system inside the virtual machine ☺	▪ Runs on the same operating system version as the host (but its usually fine). ☺

Comparison (2/2)

Feature	Virtual machine	Container
Deployment	▪ Uses a Hypervisor	▪ Uses an OS-level virtualisation method, e.g., Docker
Persistent storage	▪ Use a virtual hard disk for local storage or a network file share for storage shared by multiple servers ☺	▪ Use union file systems for local storage for a single node, or a network file share for storage shared by multiple nodes or servers ☺
Load balancing	▪ Virtual machine load balancing moves running VMs to other servers in a failover cluster. ☺	▪ Containers themselves don't move; instead an orchestrator can automatically start or stop containers on cluster nodes to manage changes in load and availability. ☺
Fault tolerance	▪ VMs can fail over to another server in a cluster, the VM's operating system restarts on the new server (takes time). ☺	▪ If a cluster node fails, any containers running on it are rapidly recreated by the orchestrator on another cluster node. ☺

Containers in VMs

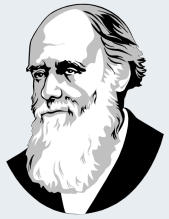
- The major concerns with running containers on bare metal servers is security:
  - Containers make use of the operating system kernel and libraries, so could potentially represent a better opportunity for an attack.
- The optimal deployment may be a combination of the two technologies.
- Deploying a container inside a VM improve isolation and portability
  - VMs can be moved between different platforms (but this is not a trivial task).



* Example of 2 VMs that host 3 containers in one bare metal server

Thank you!

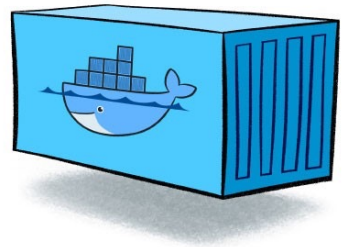
- Quote of the day:
  - "It is not the strongest of the species that survive, nor the most intelligent, but the one most responsive to change."
    - Charles Darwin



- Lab sessions starts soon!

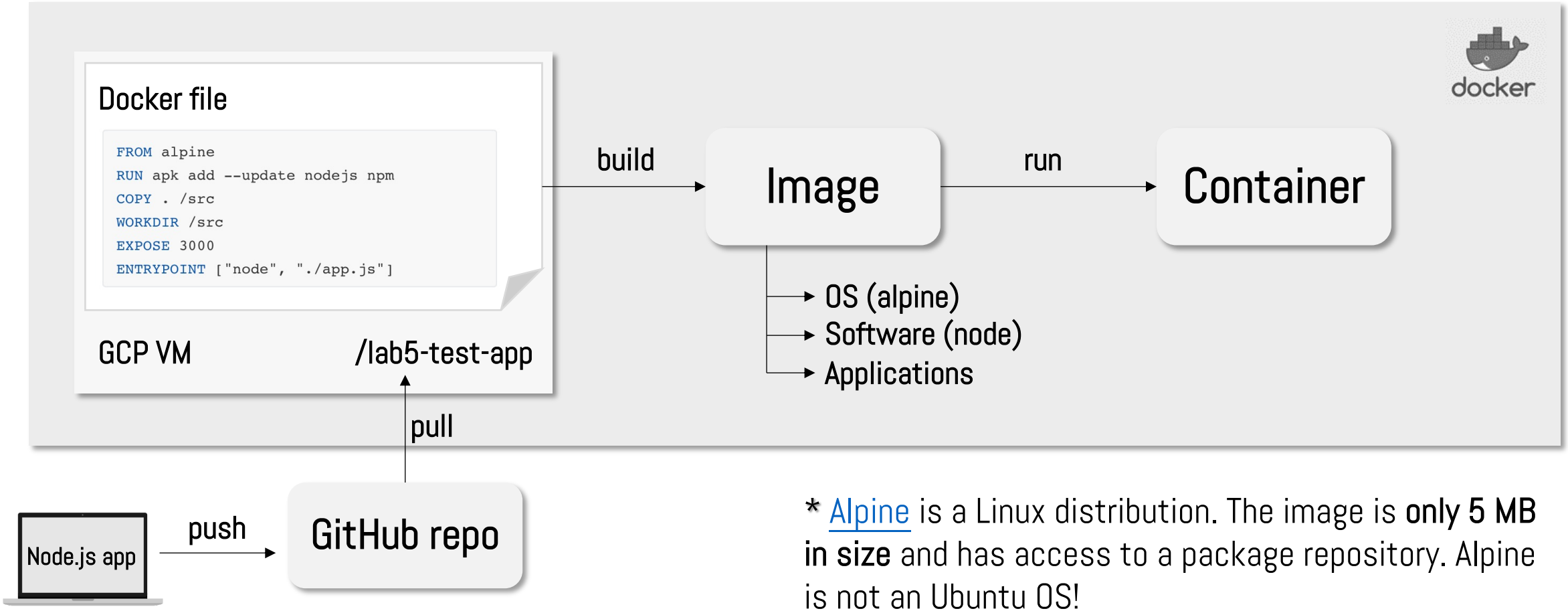
Containerising apps with Docker

What is Docker?



- Docker is a set of platform as a service products that uses OS-level virtualization to deliver software in packages called containers.
- A self-contained small space to run your code, including:
  - Your code (node.js)
  - Configuration files (.env, package.json, modules etc.)
  - Processes (running version of your node.js app)
  - Networking (allow remote connections on certain ports e.g. 3000)
  - Dependencies (express, nodemon, mongoose, body-parser, etc.)
  - Just enough of your OS to run your code ☺

Docker deployment in lab



* Alpine is a Linux distribution. The image is **only 5 MB in size** and has access to a package repository. Alpine is not an Ubuntu OS!

Docker CLI, basic commands

- `docker build` # Build an image from a Dockerfile
- `docker images` # List all images on a Docker host
- `docker run` # Run an image
- `docker ps` # List all running and stopped instances
- `docker stop` # Stop a running instance
- `docker rm` # Remove an instance
- `docker rmi` # Remove an image

Docker docs

Demo: Building a GKE cluster

Running Docker

- Intro to Docker
- Create a VM and install Docker
- Containerising an application
- Building Docker images and pushing code to GitHub

Practical Session

Deployment using Docker

- Run the tutorials:
  - Introduction to Docker
  - Docker Command Line tool
  - Using Docker to deploy and run containers
- Exercises:
  - Deploy your Docker containers using GitHub