Model experimentation & containerisation

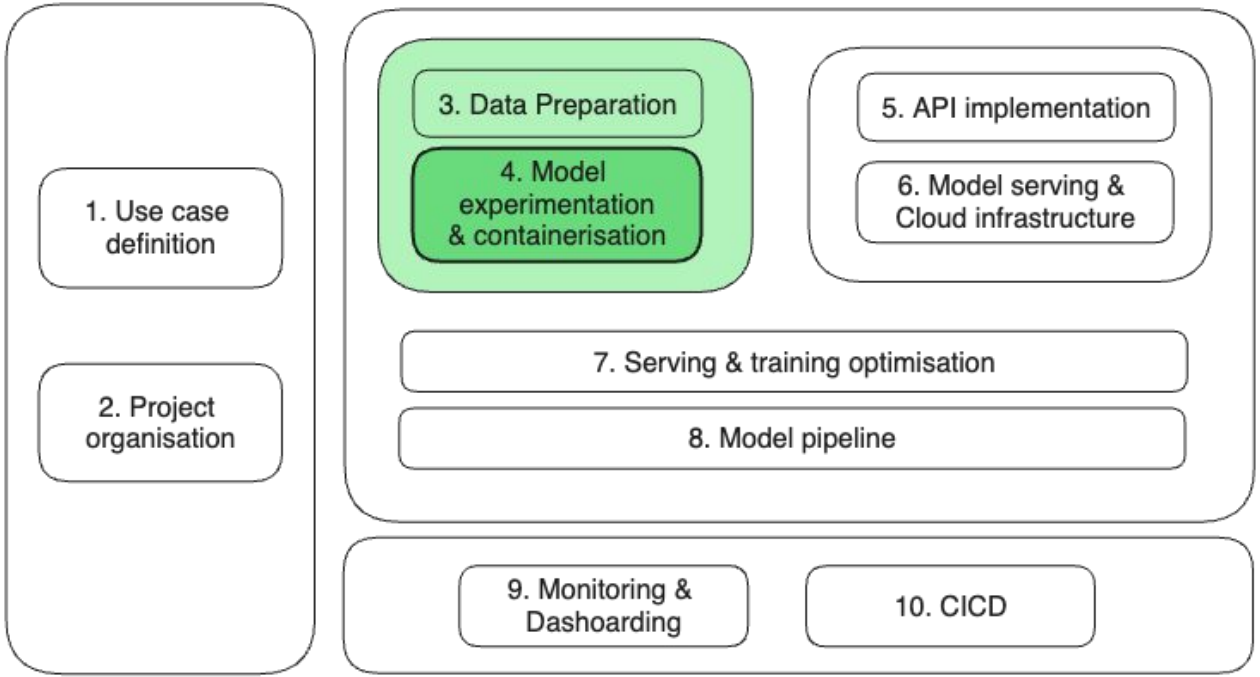
Sprint 2 - Week 4

INFO 9023 - Machine Learning Systems Design

2024 H1

Thomas Vrancken (t.vrancken@uliege.be)
Matthias Pirlet (matthias.pirlet@uliege.be)

Status on our overall course roadmap




SEPARATING
SOFTWARE
FROM HARDWARE
VIRTUAL ENVIRONMENTS
VIRTUAL MACHINES
CONTAINERS
imgflip.com

Agenda

What will we talk about today

Lecture (45 min)

1. Model experimentation
2. Virtual environments
3. Virtual machines
4. Containers

Lab (30 min)

5. Virtual environments & docker
   a. **Step 1 - 3 in lab**

Lecture (15 min)

6. Kubernetes

Lab (30 min)

7. Kubernetes
   a. **Step 4 in lab**

If we have enough time.

Project update

You now have **two options** for the Milestone 1 presentations:

A. **After class:** Between 11:30 and 12:30 on 11/03/2024 and 18/03/2024
   a. Other groups are still encouraged to use the time to work on their project, but from another location 🚀
   b. Support on the project will then be more ad-hoc those weeks. You can still raise questions by email, Discord or during the open office hours every Monday till 18:00 in office 77B in building B28.

B. **Online**: Between 13:00 and 17:00 on 11/03/2024 till 13/03/2024.
   a. On Google Meets

⚠️ Either way, make sure to **book a meeting slot** for your team with the link shared by email

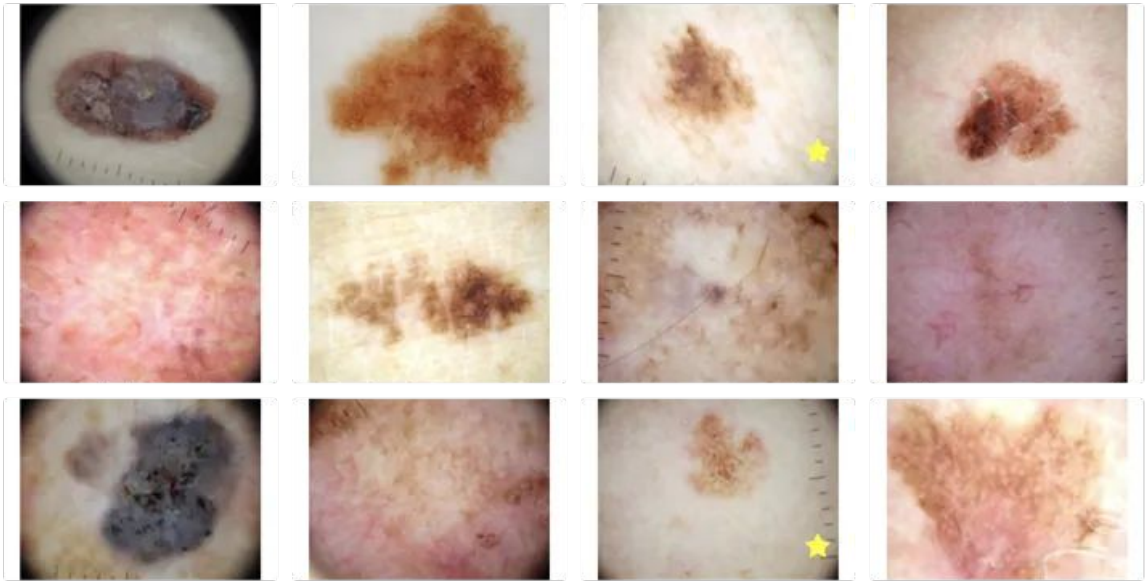
Project objective for sprint 2

You can still use W&B for your model experimentation, but not a hard requirement 🧪

#	Week	Work package	Requirement
2.1	W03	Prepare your data and run an Exploratory Data Analysis.	Required
2.2	W04	Train your ML model	Required
2.3	W04	Evaluate your ML model	Required
2.4	W04	Use W&B for step 2.2 - 2.3	Optional

Model experimentation

Test against bias

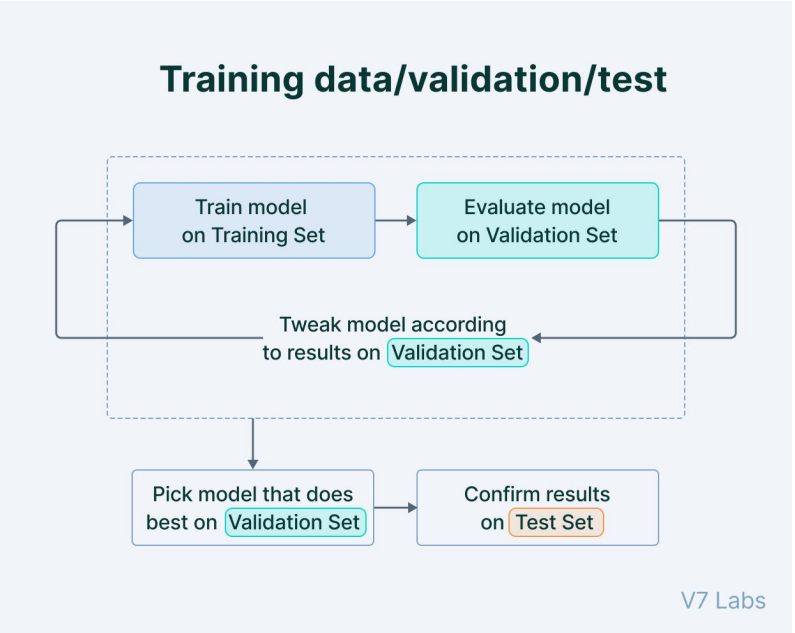


www.isic-archive.com

Splitting your dataset for experimentation

Split your raw data into:

- **Training set**
  - (60 - 80%)
  - Used to train your main model
- **Validation set**
  - (10 - 20%)
  - Validate each model epochs
  - Used for optimisation or early stopping
- **Test set**
  - (10 - 20%)
  - Confirm results of your final model

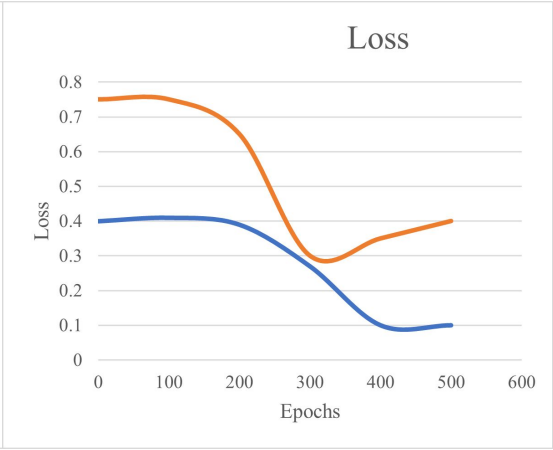


https://www.v7labs.com/blog/train-validation-test-set

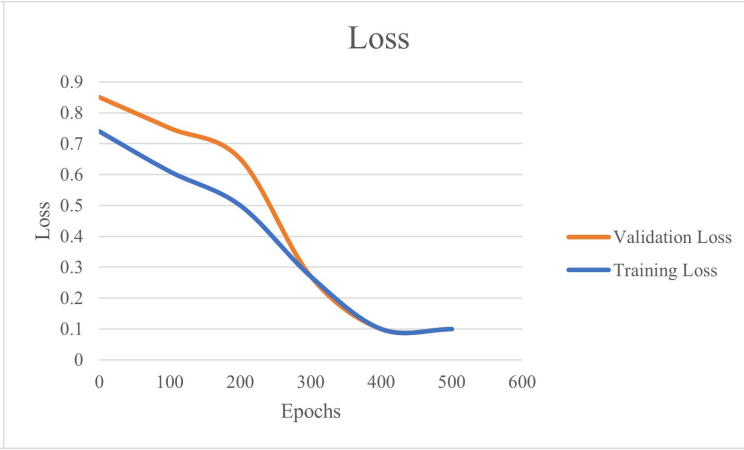
Tracking validation and training loss

Comparing loss on training and validation over each epoch can help detect fitting issues.

Underfitting



Overfitting

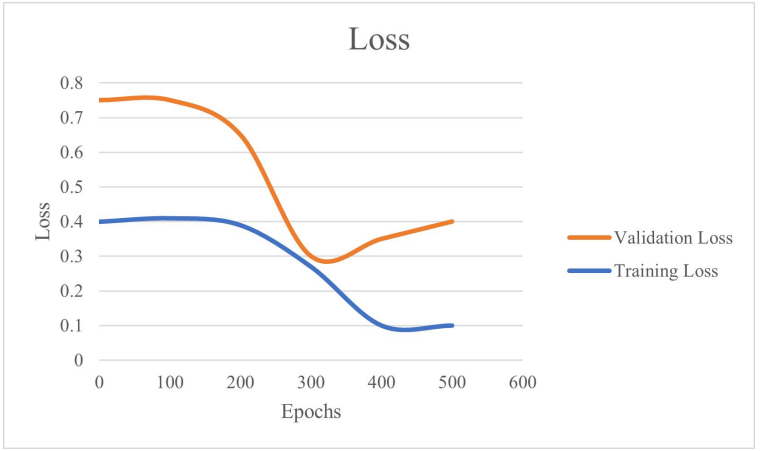


Optimal-fitting

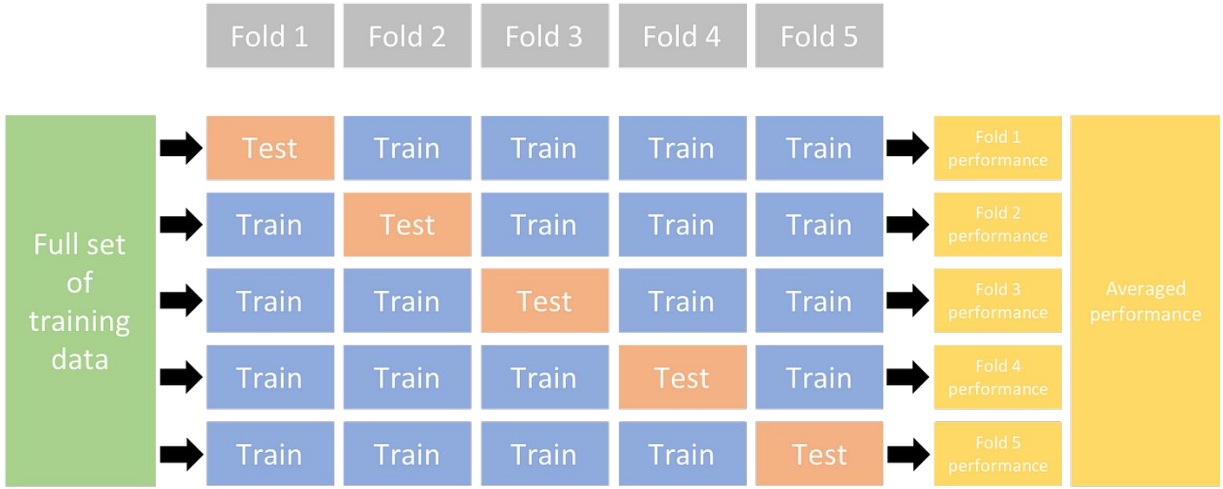
https://www.baeldung.com/cs/training-validation-loss-deep-learning

Overfitting prevention

- **Early stopping**: Pauses the training phase before the machine learning model learns the noise in the data.
- **Pruning**: Feature selection—or pruning—identifies the most important features within the training set and eliminates irrelevant ones.
- **Regularization**: Collection of training/optimization techniques that seek to reduce overfitting.
- **Ensembling**: Combines predictions from several separate machine learning algorithms. Some models are called weak learners because their results are often inaccurate. Ensemble methods combine all the weak learners to get more accurate results. The two main ensemble methods are bagging and boosting.
- **Data augmentation**: Changes the sample data slightly every time the model processes it. You can do this by changing the input data in small ways.



K-fold cross-validation



- Split the data in **K folds** and average the performance metric observed when sequentially taking each fold as test set.
- More statistically significant results
- Higher computational cost
- Works well for small data sets and lightweight models

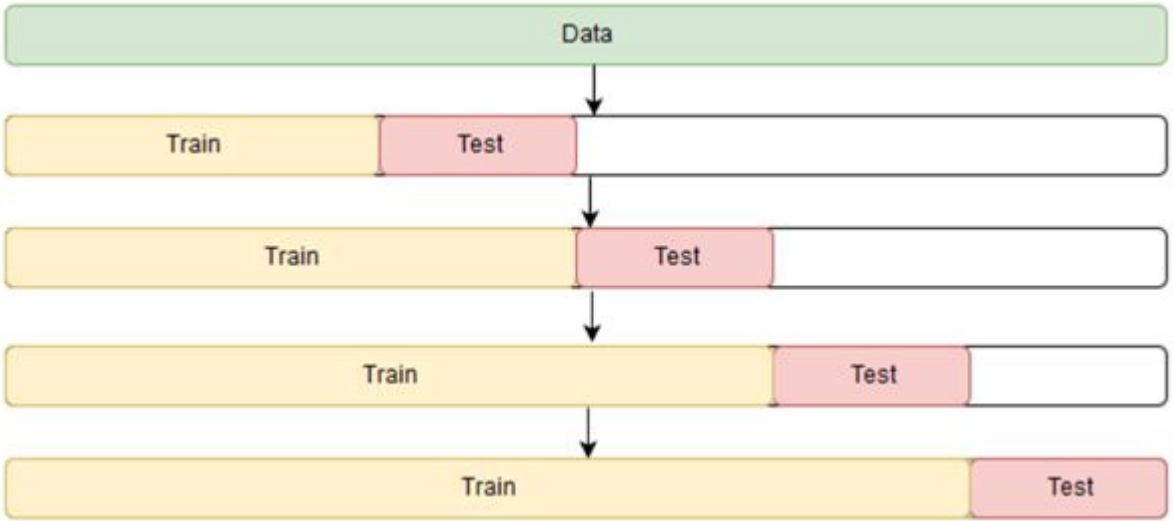
https://docs.ultralytics.com/guides/kfold-cross-validation/

Cross-validation on time series data

Often challenging to ensure that a time series model is consistently performing (e.g. can adapt to trend, seasonality, shocks, …).

Iteratively train-test your model on a rolling basis.

Requires having enough data and a short prediction horizon.



https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4

Scikit-learn Model Selection

You don’t have to implement it yourself.

```
from sklearn.model_selection import train_test_split, cross_val_score, KFold, TimeSeriesSplit
```


https://scikit-learn.org/stable/modules/cross_validation.html

Not all errors are equal

For example

- A system to detect when somebody is at the door that never works for people under 5ft (1.52m)
- A spam filter that deletes alerts from banks
- Customer churn prediction

Consider separate evaluations for important subpopulations and monitor mistakes in production.

Curate Validation Data for Specific Problems and Subpopulations:

- **Regression testing**: Validation dataset for important inputs ("call mom") -- expect very high accuracy -- closest equivalent to unit tests
- **Uniformness/fairness testing**: Separate validation dataset for different subpopulations (e.g., accents) -- expect comparable accuracy
- **Setting goals**: Validation datasets for challenging cases or stretch goals -- accept lower accuracy

Need to make sure that your model is robust against edge/dangerous cases.

https://ckaestne.github.io/seai/F2022/slides/04_modeltesting/modelquality2.pdf - CMU

Best practice: Avoid the SOTA trap

SOTA on research data != SOTA on your data

Take into consideration:

- Cost
- Latency
- Proven industry success
- Community support

Best practice: Start with simple models

- Important to **benchmark** - start simple then improve
- Fail fast - iterate towards the right modelling approach
- Lots of advantages with simpler models
  - Easier to train
  - Easier to deploy
  - Easier to maintain
  - Easier to explain
  - Easier to avoid data issues

Online testing

So far we discussed testing our model on a static dataset. There are also ways of testing your model on **live data**.

Either you somehow receive direct labels from your data (e.g. time series prediction, just wait a bit), or you can use **proxy signals**

- Manually label data periodically
- Ask users to report wrongly labeled cases and correct the label
- Soft signals such as clicks (e.g. for recommendation engines)

https://madewithml.com/courses/mlops/evaluation/

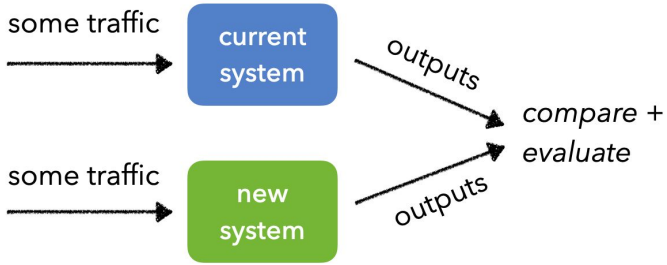
Online testing: A/B testing (1/3)

Deploy both version of your system.

Measure statistical difference of metric/KPI.

Might need to take into account **novelty effect** (users need time to adapt - think of a new chatbot).

Need enough data to draw a statistical significance.

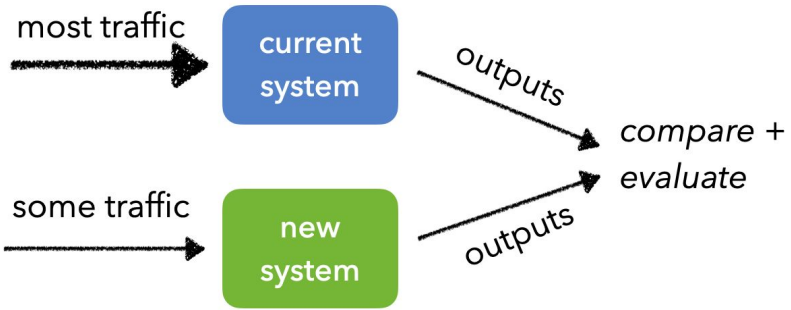


https://madewithml.com/courses/mlops/evaluation/

Online testing: Canary testing (2/3)

Similar to A/B but only redirects parts of the incoming traffic.

Useful to test out prototypes, to mitigate impact.



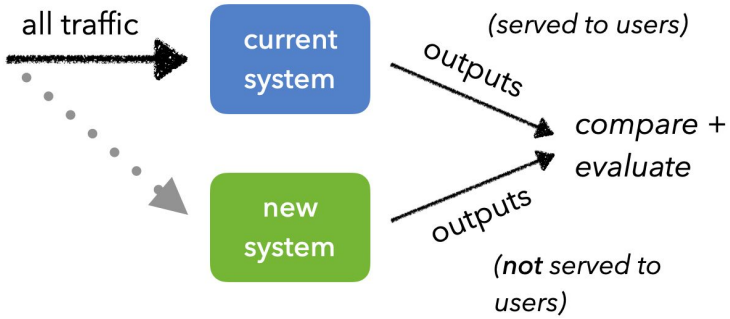
https://madewithml.com/courses/mlops/evaluation/

Online testing: Shadow testing (3/3)

Send incoming traffic to both systems but this time only send the outputs of the previous system to the users.

Very safe as you are not facing the new system to users.

But does not provide proxy data and feedback from users.



https://madewithml.com/courses/mlops/evaluation/

Virtual environments

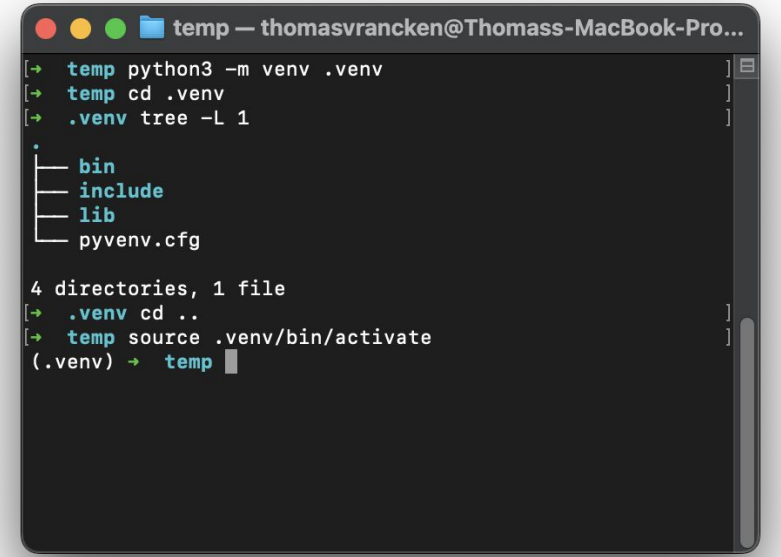
What is a virtual environment?

- Virtual environments keep **dependencies** (e.g. libraries…) in separate "**environments**" so you can switch between both applications easily and get them running.
- Given an operating system and hardware, you can set **different environments** using **different technologies**.
- Virtual environments help to make **development** and use of code more **streamlined on local machines**.
- You might use **different dependencies** (e.g. library versions) for **different projects**. Abstract away the dependencies by using a virtual environment.
- It will help maintaining dependencies (e.g. `requirements.txt`) but won't really help moving your application to production.

What is a virtual environment?

Concretely, a virtual environment is a directory with the following components:

- **Directory** where third-party libraries are installed
- **Links** to the executables on your system (python itself or pip)
- **Scripts** that ensure that the code uses the interpreter and site packages in the virtual environment



What is a virtual environment?

Here's what happens when using the virtual environment:

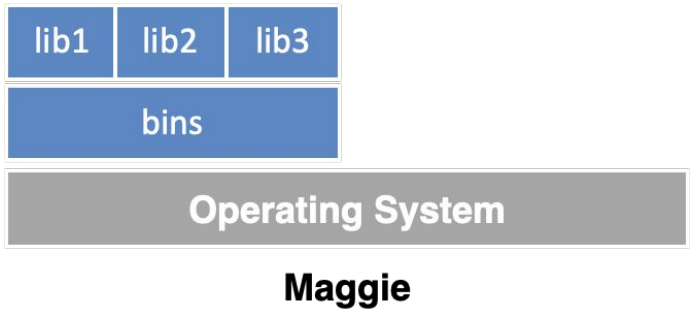
1. **Activation**: When you activate the virtual environment, your shell's PATH is updated to prioritize this bin (or Scripts) directory. This means that when you type python or pip, your shell will use the versions in the virtual environment instead of the system-wide versions.

2. **Execution**: Because of the updated PATH, when you execute Python or pip, your system uses the linked executables in the virtual environment directory. This ensures all Python operations are limited to the virtual environment.

3. **Isolation**: Since these executables are specific to the virtual environment, any Python packages you install or remove affect only this isolated environment, leaving other environments and the system-wide settings untouched.

Why should you use virtual environments?

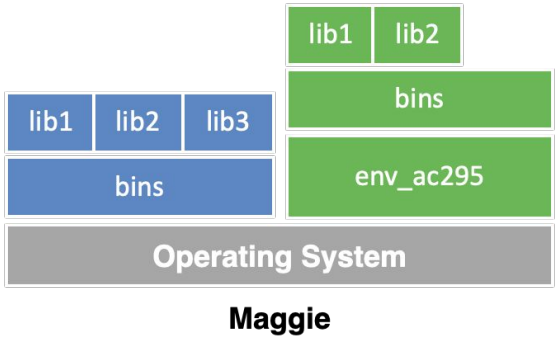
Maggie took “Intro to Machine Learning”. She used to run her Jupyter notebooks from anaconda prompt. Every time she installed a module it was placed in the either of `bin, lib, share, include`folders and she could import it in and used it without any issue.




```
$ which python
/c/Users/maggie/Anaconda3/python
```

Why should you use virtual environments?

Maggie starts taking “Machine Learning Systems Design”, and she thinks that it would be good to isolate the new environment from the previous environments avoiding any conflict with the installed packages. She adds a layer of abstraction called virtual environment that helps her keep the modules organized and avoid misbehaviors while developing a new project.

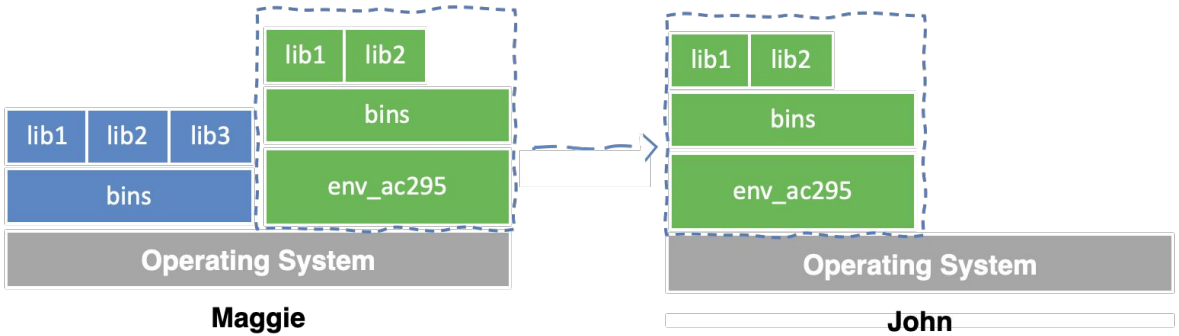


Maggie

```
$ which python
/c/Users/maggie/Anaconda3/envs/env_ac295/python
```

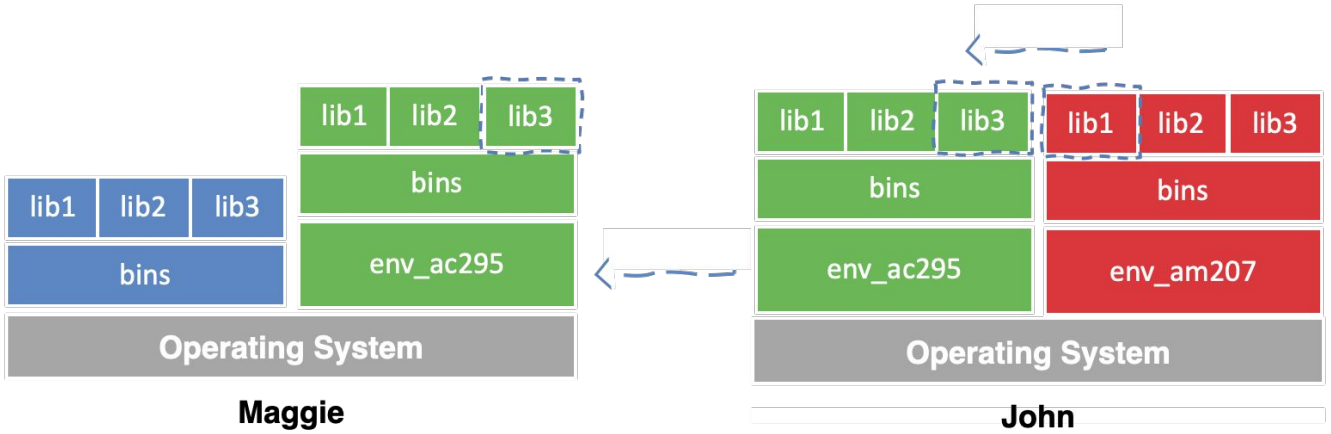
Why should you use virtual environments?

Maggie collaborates with John for the final project and shares the environment she is working on through .yml file.



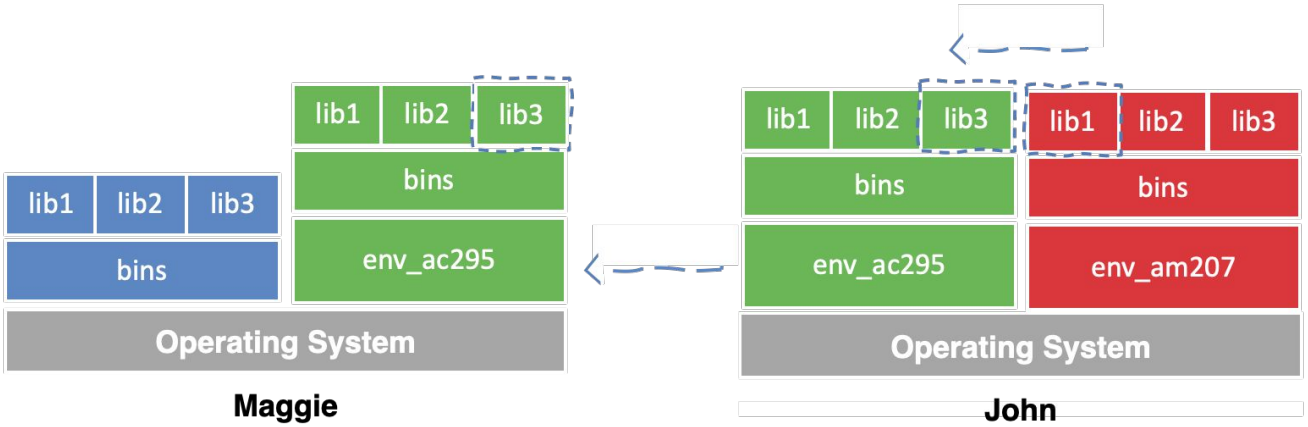
Why should you use virtual environments?

John experiments a new method he learned in another class and adds a new library to the working environment. After seeing tremendous improvements, he sends Maggie back his code and a new .yml file. She can now update her environment and replicate the experiment.



Why should you use virtual environments?

John experiments a new method he learned in another class and adds a new library to the working environment. After seeing tremendous improvements, he sends Maggie back his code and a new .yml file. She can now update her environment and replicate the experiment.



Virtual environments: virtualenv vs conda

`virtualenv`

- virtual environments manager embedded in Python
- incorporated into broader tools such as pipenv
- allow to install modules using pip package manager

How to use `virtualenv`?

- create an environment within your project folder: `python -m venv your_env_name` (often `.venv`)
- it will add a folder called `your_env_name` in your project directory
- activate environment: `source env/bin/activate`
- install requirements using: `pip install package_name=version`
- deactivate environment once done: `deactivate`

Virtual environments: virtualenv vs conda

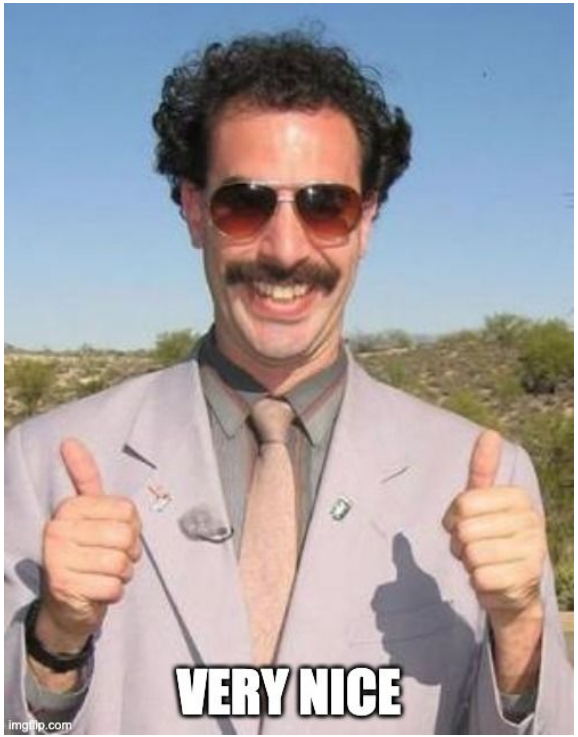
`conda` **environment**

- virtual environments manager embedded in Anaconda
- allow to use both conda and pip to manage and install packages

How to use `conda`**?**

- create an environment: `conda create --name your_env_name python=3.12`
- it will add a folder located within your anaconda installation: `/Users/your_username/anaconda3/envs/your_env_name`
- activate environment: `conda activate your_env_name` (should appear in your shell)
- install requirements using: `conda install package_name=version`
- deactivate environment once done: `conda deactivate`
- duplicate your environment using YAML file: `conda env export > my_environment.yml`
- to recreate the environment now use: `conda env create -f environment.yml`
- find which environment you are using : `conda env list`

When you installed all your dependencies only using your requirements.txt file



Why should you use virtual environments?

Pros

- Reproducible research
- Dependency management (forces you to maintain a requirements.txt with all dependencies)
  - Natural first step before containerisation
- Improved engineering collaboration
- Broader skill set

Cons

- Effort setting up your environment
- Storage space (duplicated binaries and libraries)
- Tool / OS compatibility (some IDEs and OS won't be able to share the same .venv directory)

The .gitignore file


```
# These are some examples of commonly
# ignored file patterns.

# Compiled Python bytecode
*.py[cod]

# Log files
*.log

# Environmen
*.env
venv/

# Data
data/

# Secrets
secrets/

# Jupyter Notebook
.ipynb_checkpoints
```

Virtual Machines (VM)

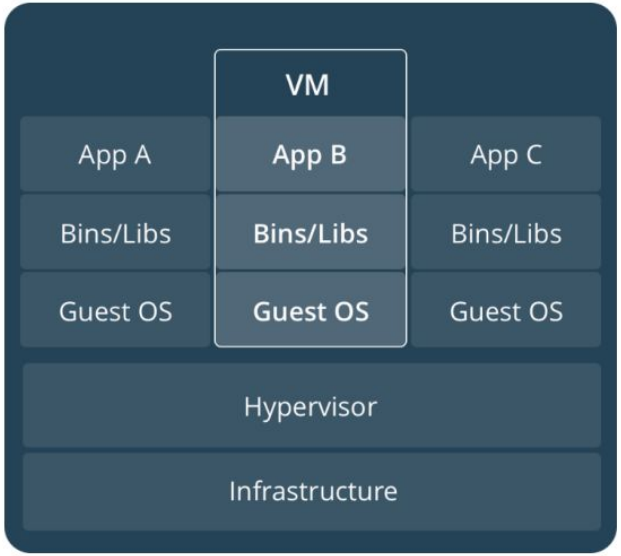
Virtual Machine (VM)

Provides a fully functional snapshot of an operating system (OS).

Connect to the VM in a similar way as you would connect to a specific computer.

Motivation

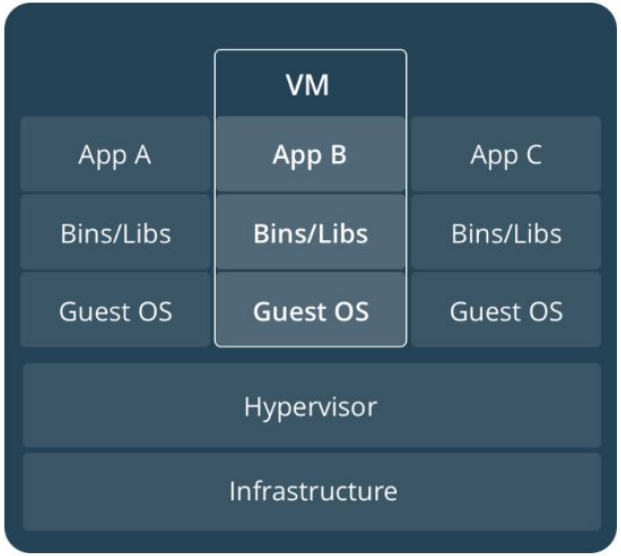
- We have our isolated systems, and after we set up the environment with our colleagues' machine, we expect to get identical results, right? Unfortunately, it is not always the case. Why? Most likely because we run on a different operating system.
- Even though using virtual environments, we isolate our computations, we might need to use the same operating system that requires running "like if" we are in different machines.
- How can we run the same experiment? **Virtual Machines**!



Virtual Machine (VM)

- Virtual machines have their **own virtual hardware**: CPUs, memory, hard drives, etc.
- You need a **hypervisor** that manages different virtual machines on server
- Operating system is called the "**host**" while those running in a virtual machine are called "**guest**"
- You can install a completely different operating system on this virtual machine

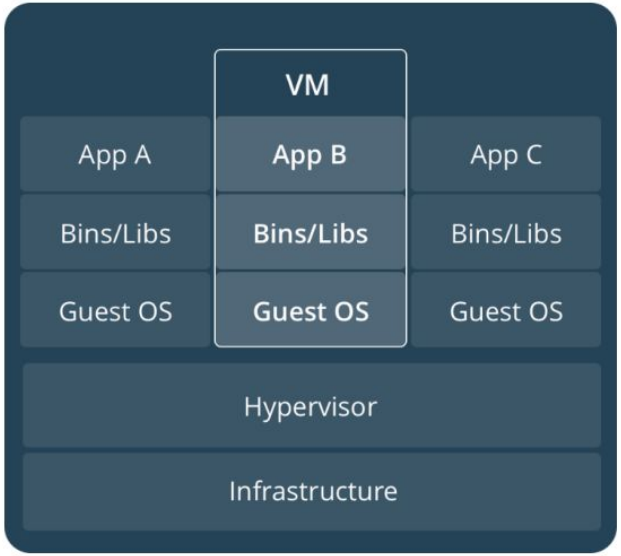
(Install a Windows VM on your mac here)



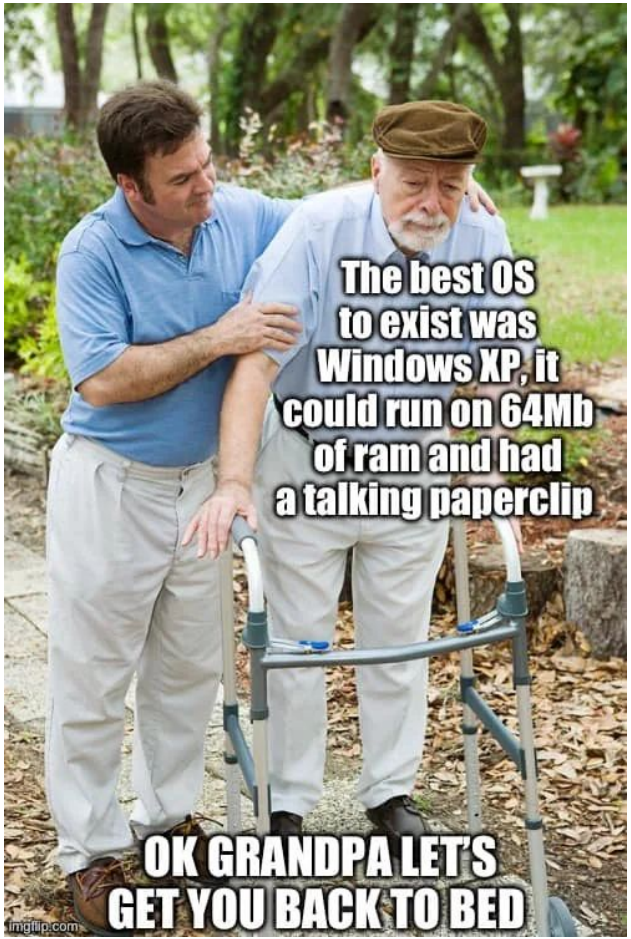
Virtual Machine (VM)

- Virtual machines have their **own virtual hardware**: CPUs, memory, hard drives, etc.
- You need a **hypervisor** that manages different virtual machines on server
- Operating system is called the "**host**" while those running in a virtual machine are called "**guest**"
- You can install a completely different operating system on this virtual machine

(Install a Windows VM on your mac [or] here)



VMs, let’s the nostalgics relive some of the glory years



Virtual Machine (VM)

Advantages

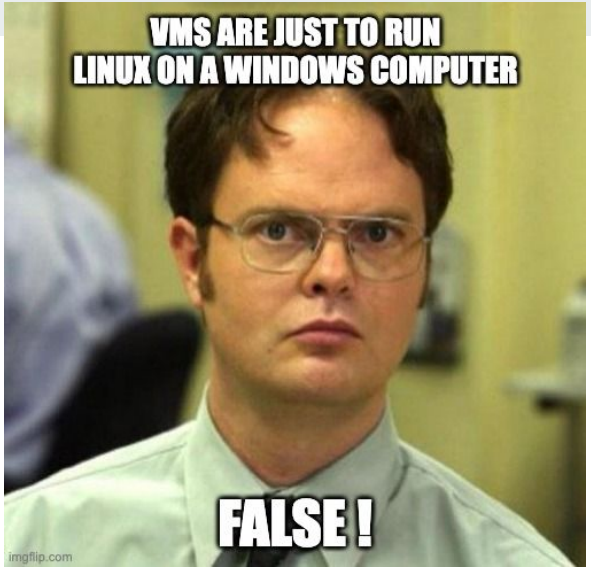
- **Autonomy**: it works like a separate computer system; it is like running a computer within a computer.
- **Secure**: the software inside the virtual machine cannot affect the actual computer.
- **Costs**: buy one machine and run multiple operating systems.

Limitations

- Uses hardware in your local machine (cannot run more than two on an average laptop)
- Takes time to boot-up
- There is overhead associated with virtual machines
  - Guest is not as fast as the host system
  - Takes long time to start up
  - May not have the same graphics capabilities

https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Where can you use VMs?



- VMs are often use locally, to access a separate OS
- You can also create and use VMs in the **Cloud** !
  - Select specific hardware (e.g. GPUs)
  - Pay for what you use
  - Collaborate with your team
  - Isolated and secured environment

Amazon **EC2**

Try it yourself

AzureVM

Try it yourself

Google Compute Engine

Try it yourself

Containerisation

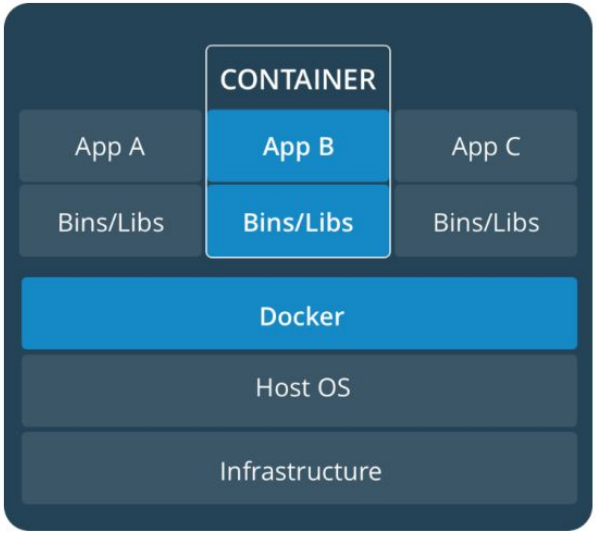

IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

Containers

Containers encapsulate an application as a **single executable package** that contains all the information to **run it on any hardware**:

- Application code
- Configuration files
- Libraries
- Dependencies

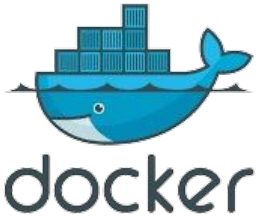
Abstracts the application from its **host operating system.**



Containers can be easily transported from a desktop computer to a virtual machine (VM) or from a Linux to a Windows operating system, and they will run consistently on virtualized infrastructures or on traditional “bare metal” servers, either on-premise or in the cloud.

https://www.ibm.com/topics/containerization

Docker



Docker is a platform designed to make it easier to create and manage containers.

It is essentially the most popular platform to do so, even though there are alternatives:

- Podman
- rkt
- LXC (Linux Containers)
- containerd
- CRI-o

What is the difference between an image and container?

Docker Image is a template aka blueprint to create a running **Docker container**.

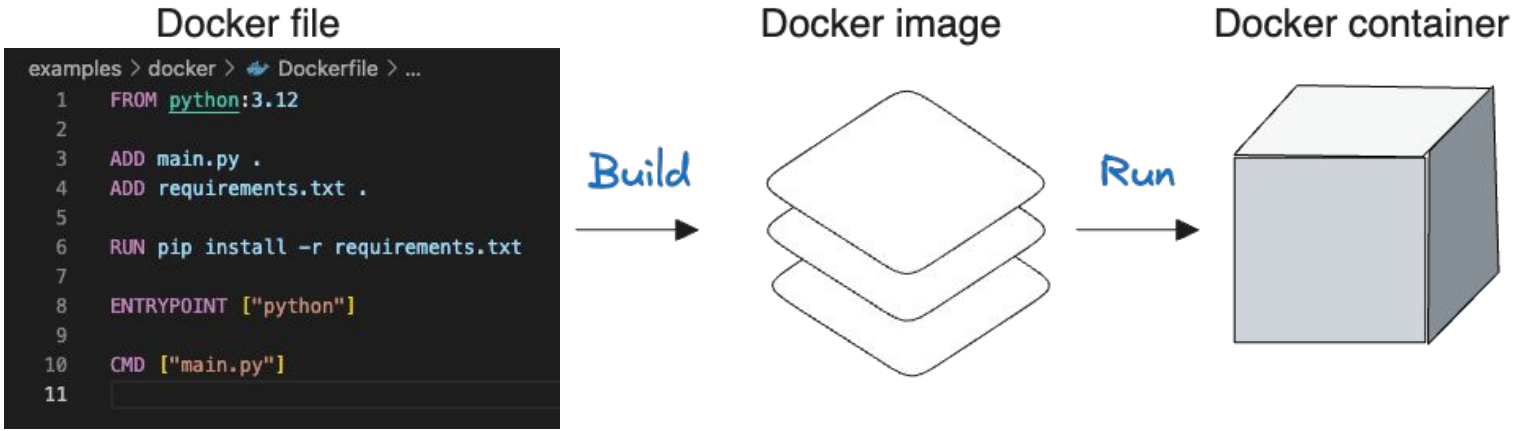
- Docker uses the information available in the Image to create (run) a container.

Image is like a **recipe**, container is like a **dish.**

You can think of an image as a **class** and a container is an **instance** of that class.

What is the difference between an image and container?

We use the **Dockerfile**, a simple text file, to configure and build the Docker Image, which are iso and other files. We run the Docker Image to get Docker Container.



Looking inside a Dockerfile

`FROM`: This instruction in the Dockerfile tells the daemon, which base image to use while creating our new Docker image. Here we use a standard Python installation image (already has python installed).

`ADD`: Add source files to the into the container's base folder (you can also add everything with `` `ADD . .` ``).

`RUN`: Instructs the Docker daemon to run the given commands as it is while creating the image. A Dockerfile can have multiple RUN commands, each of these RUN commands create a new layer in the image.

`ENTRYPOINT`: Used when you would like your container to run the same executable every time. Usually, ENTRYPOINT is used to specify the binary and CMD to provide parameters.

`CMD`: The CMD sets default command and/or parameters when a docker container runs. CMD can be overwritten from the command line via the docker run command.

Multiple containers from same image

How can you run multiple containers from the same image? Wouldn't they all be identical?

Yes, you could think of an image as calling a **class**. You can build an image and run it with different parameters using the **CMD** and therefore different containers will be different.

So you can run the same **image** with different **parameters** (e.g. python arguments).

```
FROM ubuntu:latest
RUN apt-get update
ENTRYPOINT ["/bin/echo", "Hello"]
CMD ["world"]
```

```
> docker build -t hello_world_cmd:first -f
Dockerfile_cmd .

> docker run -it hello_world_cmd:first
> Hello world
> docker run -it hello_world_cmd:first Pavlos
> Hello Pavlos
```

Multiple containers from same image

Python file with multiple arguments

examples > docker > main.py > ...

```python
import argparse

# Create the parser
parser = argparse.ArgumentParser(description="Process some integers.")

# Add arguments
parser.add_argument('--arg1', type=str, default='default_value1',
                    help='A description for arg1')
parser.add_argument('--arg2', type=str, default='default_value2',
                    help='A description for arg2')

# Parse the arguments
args = parser.parse_args()

print(f"Argument 1: {args.arg1}")
print(f"Argument 2: {args.arg2}")

```

Multiple containers from same image

Dcokerfile with multiple CMD options

examples > docker > Dockerfile > ...

```
1   FROM python:3.12
2
3   ADD main.py .
4   ADD requirements.txt .
5
6   RUN pip install -r requirements.txt
7
8   ENTRYPOINT ["python"]
9
10  CMD ["main.py", "--arg1", "value1", "--arg2", "value2"]
11
```

Multiple containers from same image

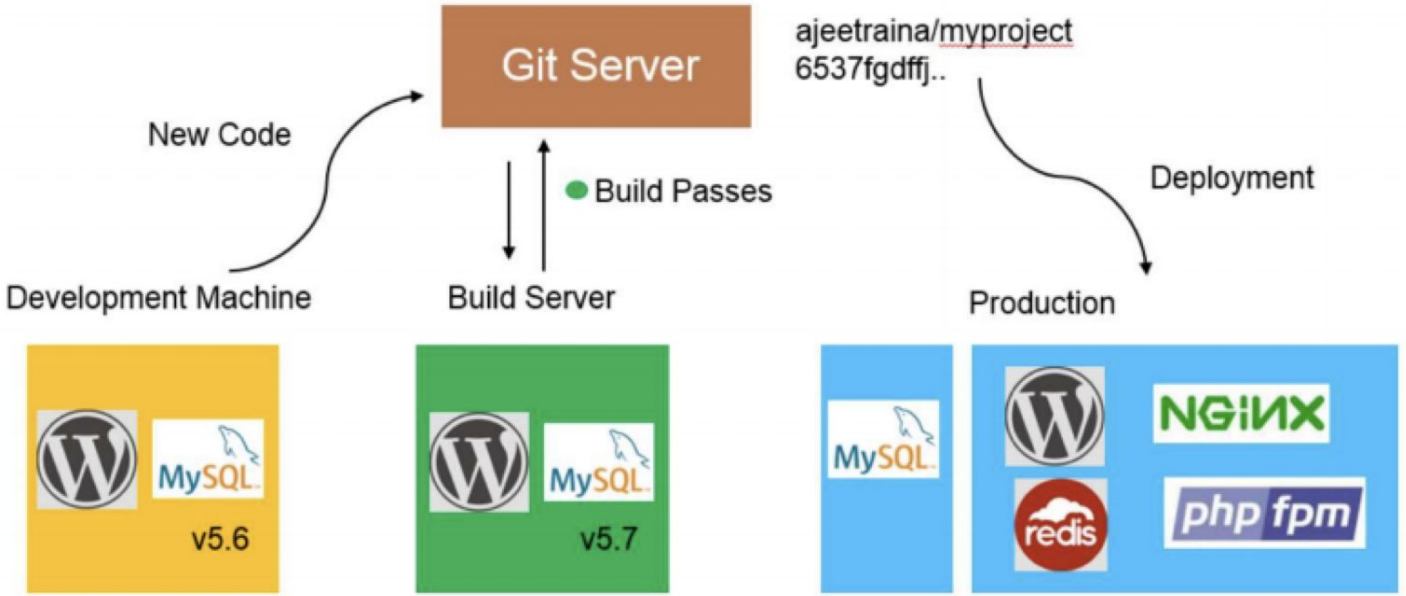
Can change python arguments upon docker run

```
docker git:(main) ✗ docker run -it python-imagename

Argument 1: value1
Argument 2: value2
```

```
docker git:(main) ✗ docker run -it python-imagename main.py --arg1 new_value1 --arg2 new_value2

Argument 1: new_value1
Argument 2: new_value2
```

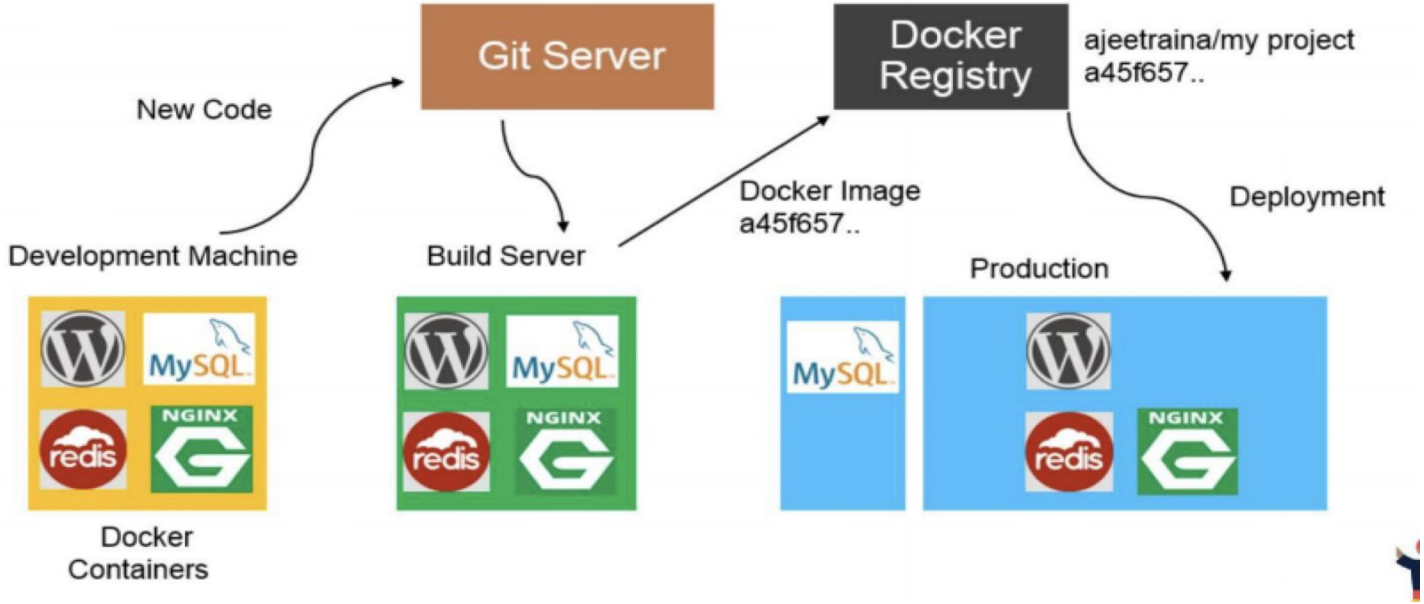
Traditional software development

Without Docker

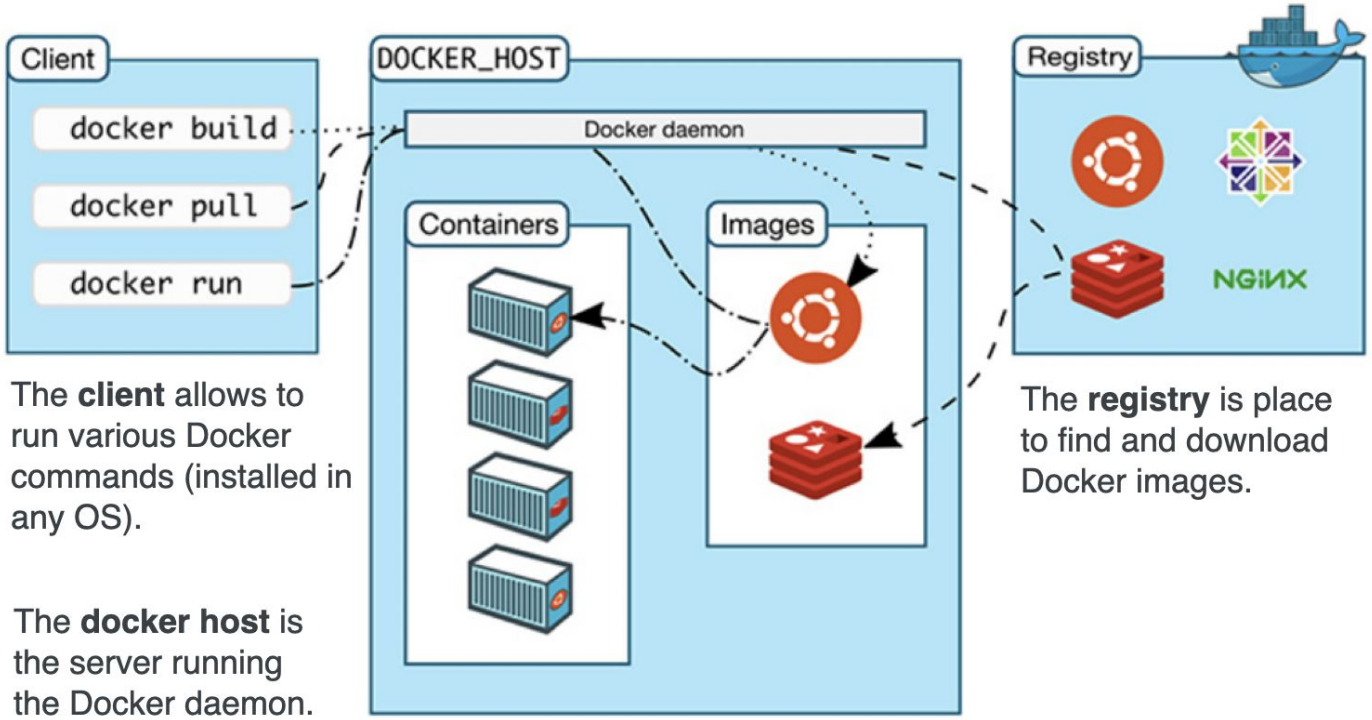


Traditional software development

With Docker



Docker client, host and registry



The **client** allows to run various Docker commands (installed in any OS).

The **docker host** is the server running the Docker daemon.

The **registry** is place to find and download Docker images.

https://nickjanetakis.com/blog/understanding-how-the-docker-daemon-and-docker-cli-work-together

Docker registry services

DOCKER REGISTRY SERVICES

DOCKER HUB

Docker hub is the official image repository of the docker.Its helps to store , share and distribute the docker image

QUAY

It is the docker registry owned by Red hat. Its helps to create on premises and cloud repository

GOOGLE CONTAINER REGISTRY

It is the docker registry created by the google.Its used to setup the private registeries

AMAZON ELASTIC CONTAINER REGISTRY

It is docker registry created by the amazon.This helps the organisation to store and deploy the container in the amazon cloud

Popular base docker images

- Tensorflow
- Pytorch
- Python
- Ubuntu
- Alpine
- Nginx
- PostGreSQL
- Redis
- MongoDB

Why should you use containers?

It has the best of the two worlds because it allows:

- to create isolate environment using the preferred operating system
- to run different systems without sharing hardware

The advantage of using containers is that they only virtualize the operating system and do not require dedicated piece of hardware because they share the same kernel of the hosting system.

Containers give the impression of a separate operating system however, since they're sharing the kernel, they are much cheaper than a virtual machine.

Why should you use containers?

- With container images, we confine the application **code**, its **runtime**, and all its **dependencies** in a pre-defined format.
- With the same image, you can **reproduce** as many containers as you wish. Think about the image as the recipe 📜 and the container as the cake 🎂
    - You can make as many cakes as you'd like with a given recipe!
- A container **orchestrator** is a single controller/management unit that connects multiple nodes together.
    - *To come in "Model Pipeline" lecture!*
- You can create a container on a Windows but install an image of a Linux OS inside that container. The container still works on the Windows machine

Why should you use containers?

- Containers are **application-centric** methods to deliver high-performing, scalable applications on any infrastructure of your choice.
- Containers are best suited to deliver **microservices** by providing portable, isolated virtual environments for applications to run without interference from other running applications.
- Because they're so **lightweight**, you can have many containers running at once on your system.

Containers pros & cons

Pros

- **Portability**: Able to run uniformly and consistently across any platform or cloud.
- **Speed**: Lightweight and only include high level software - fast to modify and iterate on.
- **Efficiency**: OS and infrastructure layer is not contained in the container. Thus, containers are smaller in capacity than a VM and require less start-up time. You can run more containers on the same hardware than VMs.
- **Modularity**: Organise applications into *microservices* that are run independently from each other. Separate development.
- **Fault isolation**: Isolated containerised applications - failure of one container does not affect the continued operation of any other containers. You can identify and correct any technical issues within one container without any downtime in other containers.
- **Ease of management**: Container orchestration platforms (e.g. *Kubernetes*) can ease management tasks such as scaling containerized apps, rolling out new versions of apps, and providing monitoring, logging and debugging, among other functions.

https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Containers pros & cons

Cons

- **Shared host exploits**: Multiple containers often share one hardware. If one container contains an exploit (virus) it could contaminate the entire hardware. Especially as it is common to re-use public pre-made containers.

https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms

Docker definitions recap

Docker Image: The basis of a Docker container. Represent a full application

Docker Container: The standard unit in which the application service resides and executes

Docker Engine: Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

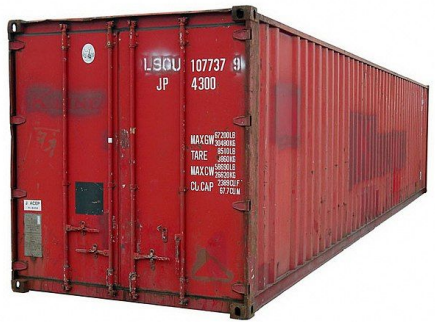
Registry Service: (e.g. Docker Hub or Docker Trusted Registry) Cloud or server-based storage and distribution service for your images

… Also a highly demanded skill !

You need to use it all the time! Therefore highly demanded skill.

“Coming in at the top of the requirements list, and highlighted in 40% of the total group, was knowledge of container tooling, specifically Docker and Kubernetes. Traditionally, this has been the domain of DevOps, Reliability and Platform Engineers, but it has become a fundamental part of MLOps Engineering. A lack of knowledge in this area puts you at a significant disadvantage to those that do and is likely to be a key development area for those moving into MLOps from ML or Data Engineering backgrounds, who may not have had the opportunity to work on container tooling in production.”

- Survey on 310 ML Engineers job positions



https://marvelousmlops.substack.com/p/what-do-teams-really-want-from-an?utm_source=profile&utm_medium=reader2

Virtual Machines (VM) vs Docker Container

Not the same thing!

VMs

- Each VM runs its own OS
- Boot up time is in minutes
- Not version controlled
- Cannot run more than couple of VMs on an average laptop
- Only one VM can be started from one set of VMX and VMDK files

Docker

- Container is just a user space of OS
- Containers instantiate in seconds
- Images are built incrementally on top of another like layers. Lots of images/snapshots
- Images can be diffed and can be version controlled. Docker hub is like Github
- Can run many Dockers in a laptop
- Multiple docker containers can be started from one Docker image

Full overview

	Virtual environment	Virtual Machine	Docker
Computational / memory usage	Low	High	Low
Effort	Easy	High at beginning, then low	Medium
Versatility	Medium	High	High
Portability	Medium	High	High

Lab: Docker

Kubernetes

Why do we need kubernetes?

Virtual Environment

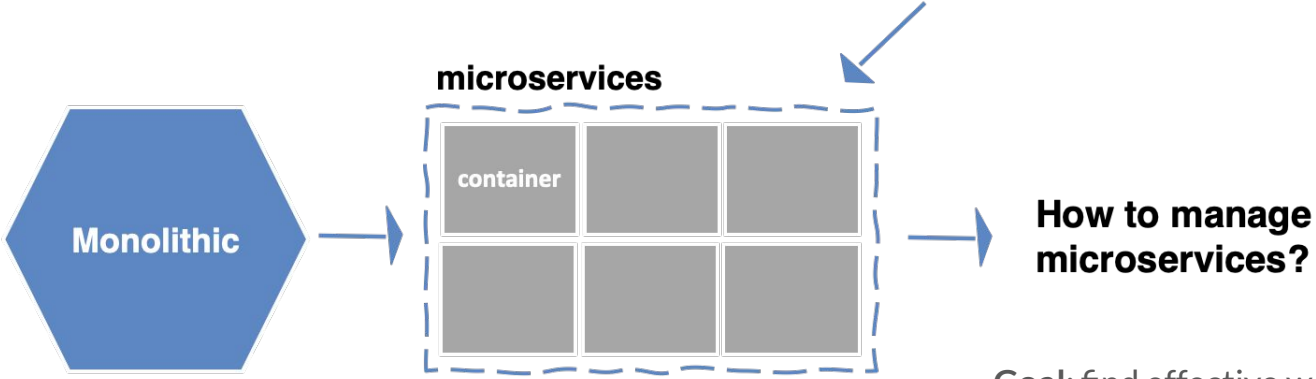
Pros: remove complexity
Cons: does not isolate from OS

Virtual Machines

Pros: isolate OS guest from host
Cons: intensive use hardware

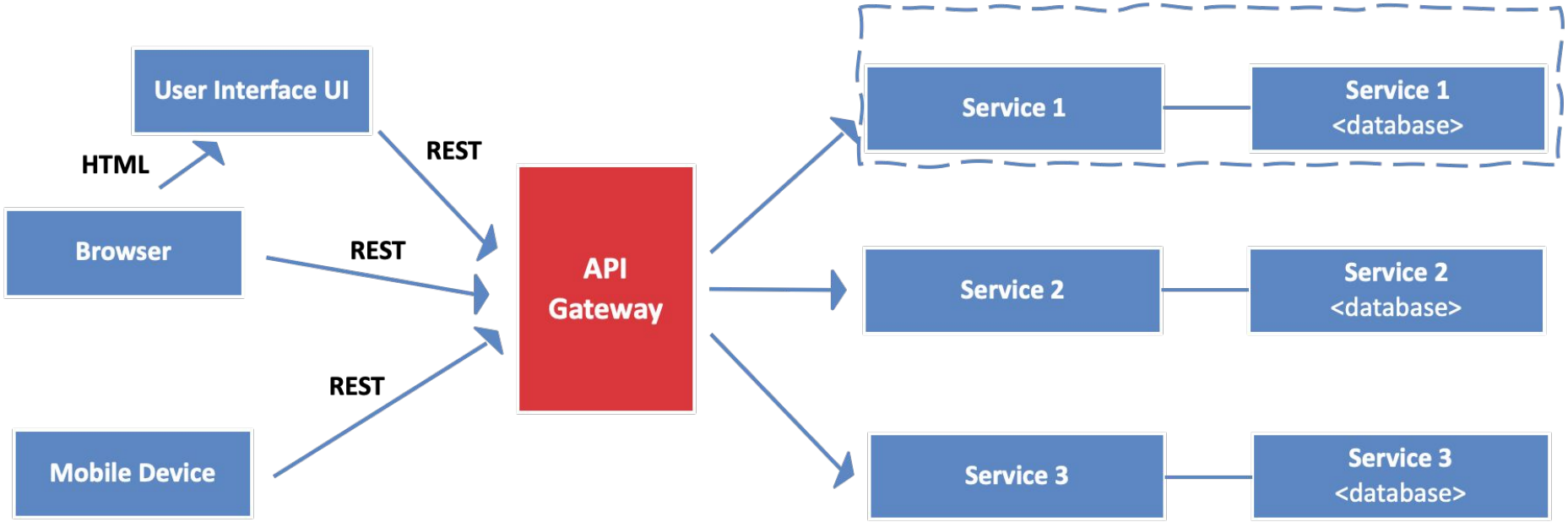
Containers

Pros: lightweight
Cons: issues with security, scalability, and control



Goal: find effective ways to deploy our apps (more difficult than we might initially imagine) and to break down a complex application into smaller ones (i.e. microservices)

Use Microservice Architecture to build App



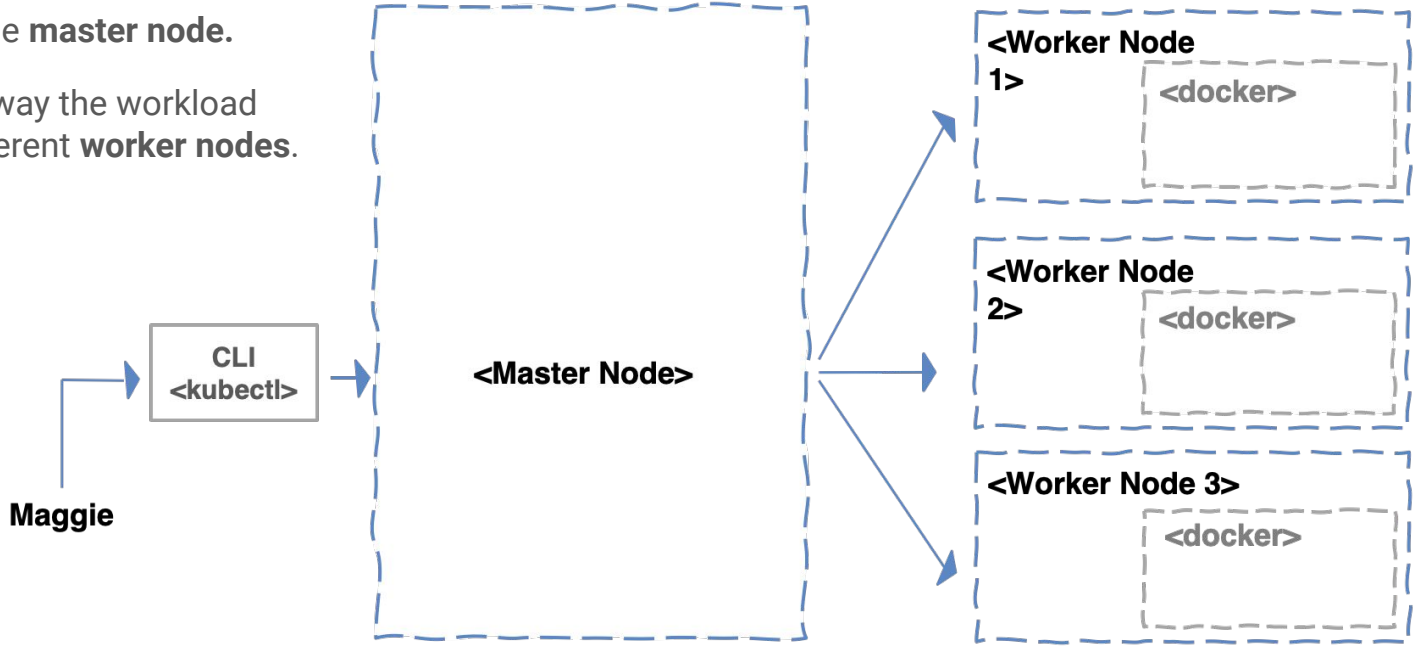
Kubernetes (k8)

- K8s is an **orchestration tool** for managing distributed services or containerized applications across a **distributed cluster of nodes**.
- K8s itself follows a client-server architecture with a **master node** and **worker nodes**.
- Core concepts in Kubernetes include **pods**, **services** (logical pods with a stable IP address) and **deployments** (a definition of the desired state for a pod or replica set).
- K8s users define rules for how container management should occur, and then K8s handles the rest!

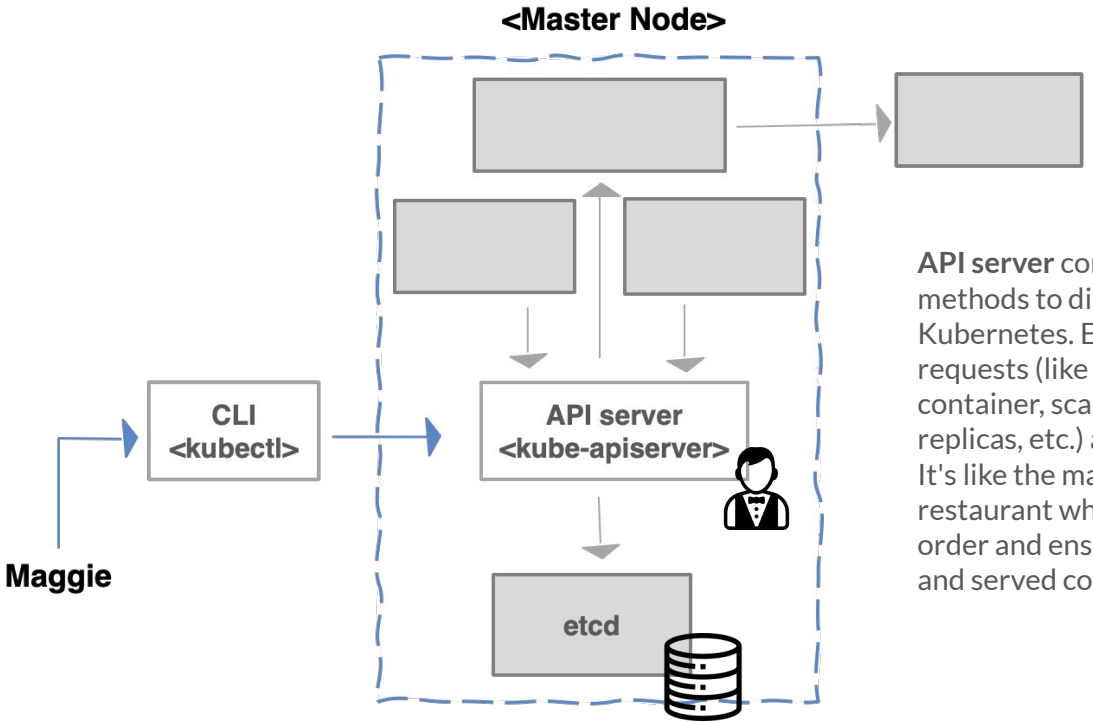
Architecture of kubernetes

Interact with the **master node.**

k8 abstracts away the workload balance to different **worker nodes**.



Architecture of kubernetes

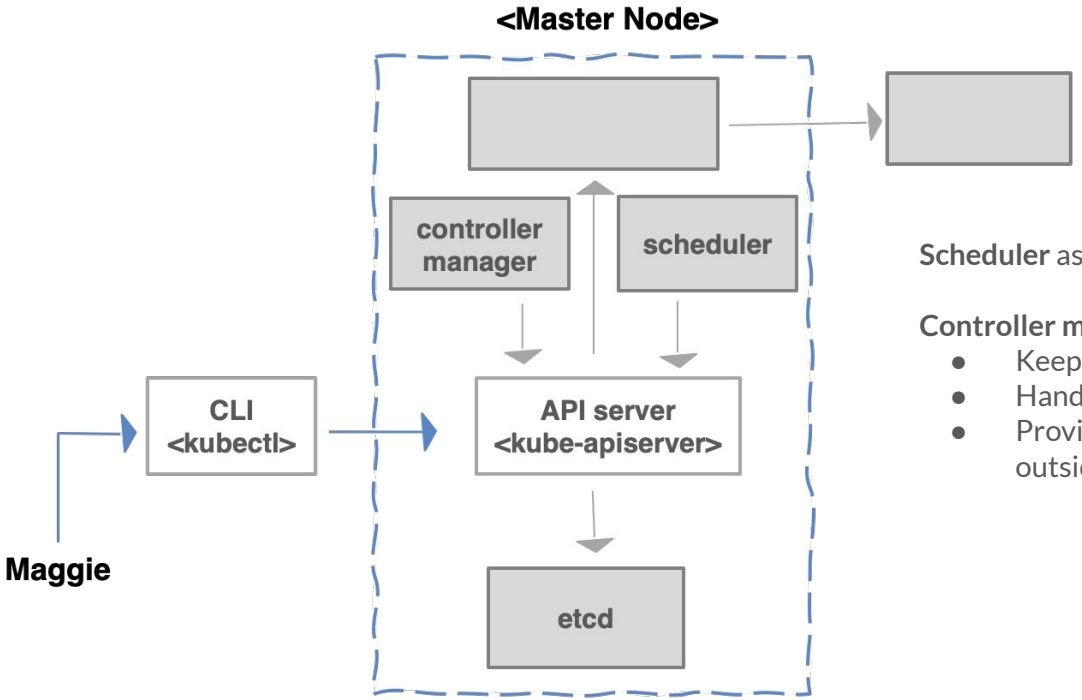


The master node has:

API server contains various methods to directly access the Kubernetes. Ensures your requests (like launching a new container, scaling the number of replicas, etc.) are carried out. It's like the manager of a restaurant who takes your order and ensures it's prepared and served correctly.

etcd works as backend for service discovery that stores the cluster's state and its configuration. database that keeps all the configuration details and the state of your entire Kubernetes cluster

Architecture of kubernetes

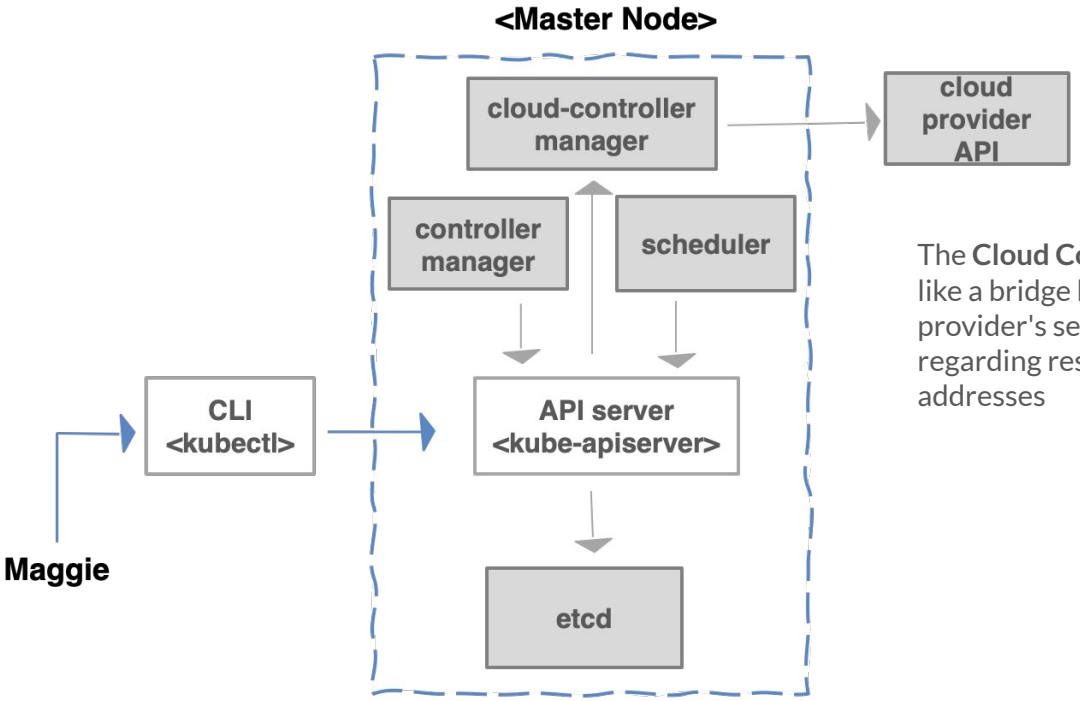


Scheduler assigns microservices to worker nodes.

Controller manager:

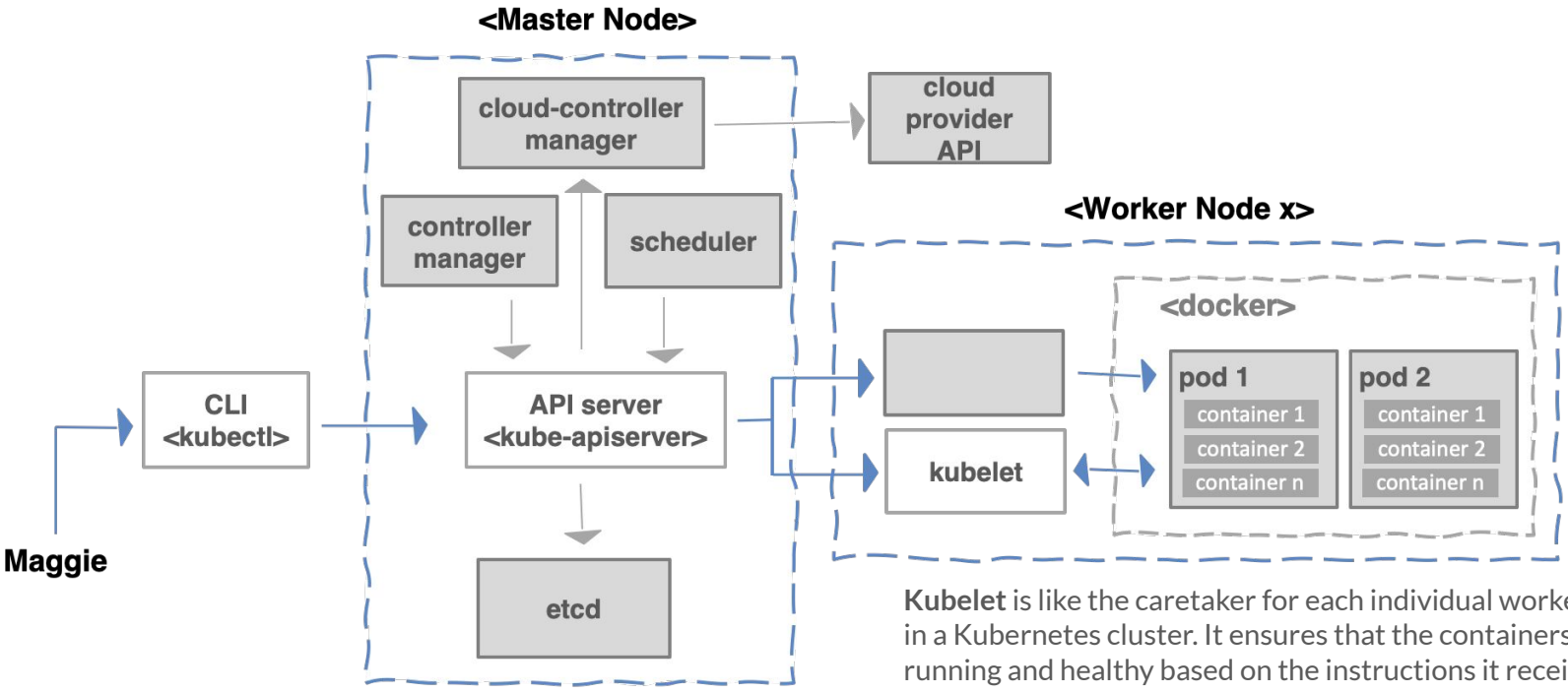
- Keeps track of worker nodes
- Handles node failures and replicates if needed
- Provide endpoints to access the application from the outside world

Architecture of kubernetes



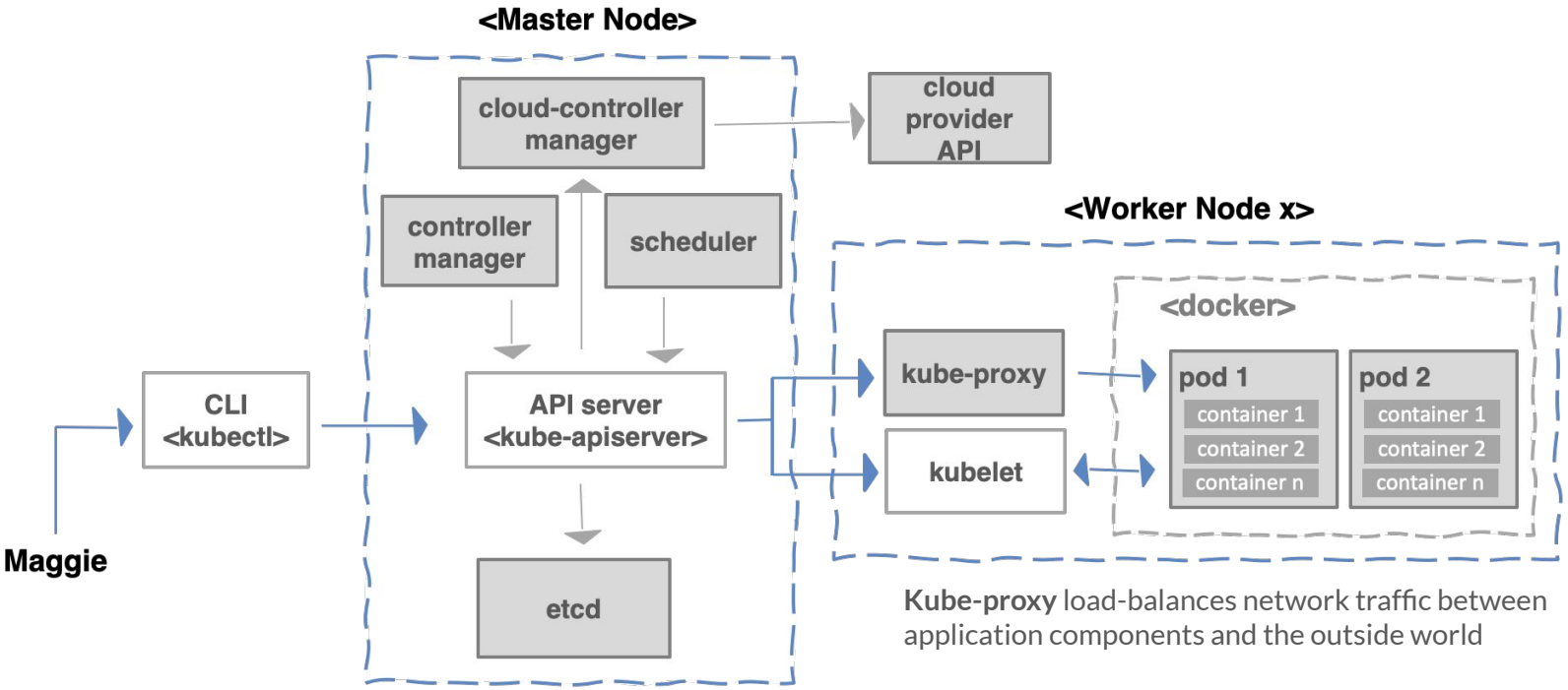
The **Cloud Controller Manager** in a Kubernetes cluster is like a bridge between your Kubernetes cluster and the cloud provider's services. It communicates with cloud provide regarding resources such as nodes, networking and IP addresses

Architecture of kubernetes



Kubelet is like the caretaker for each individual worker node in a Kubernetes cluster. It ensures that the containers are running and healthy based on the instructions it receives from the API server.

Architecture of kubernetes



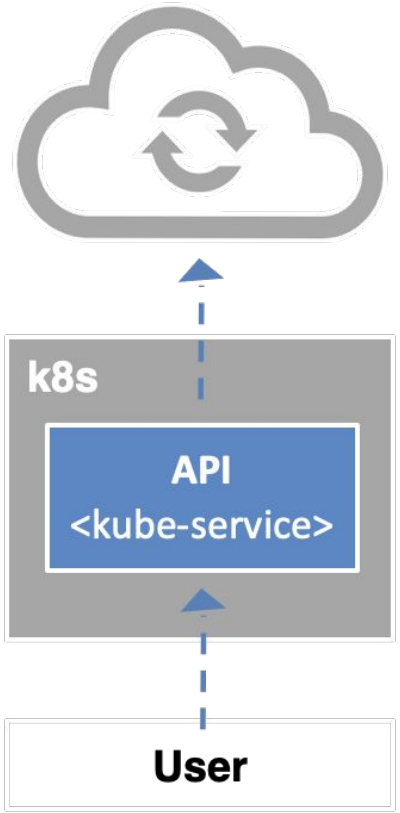
Kube-proxy load-balances network traffic between application components and the outside world

Why use kubernetes

Many reasons:

- Velocity
- Scaling (of both software and teams)
- Abstracting the infrastructure
- Efficiency

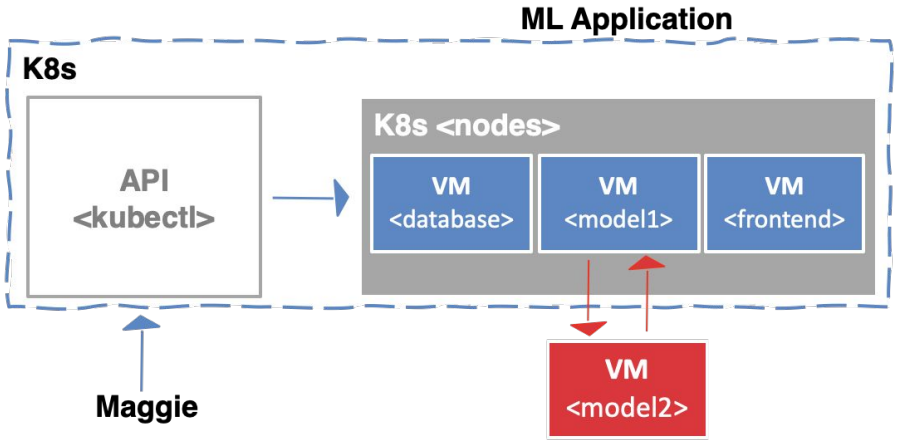
All these aspects relate to each other to speed up process that can reliably deploy software.



Why use kubernetes: Velocity

Velocity, in this context, means the speed and efficiency with which a team can deliver updates, features, or entire applications.

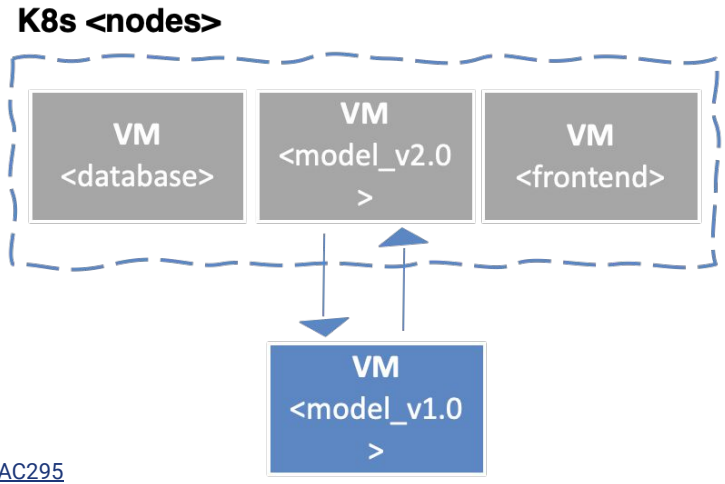
It also includes the speed to which your system adapts to issues.



Why use kubernetes: Velocity

Velocity is enabled by:

- **Immutable system**: you can't change running container, but you create a new one and replace it in case of failure (allows for keeping track of the history and load older images)

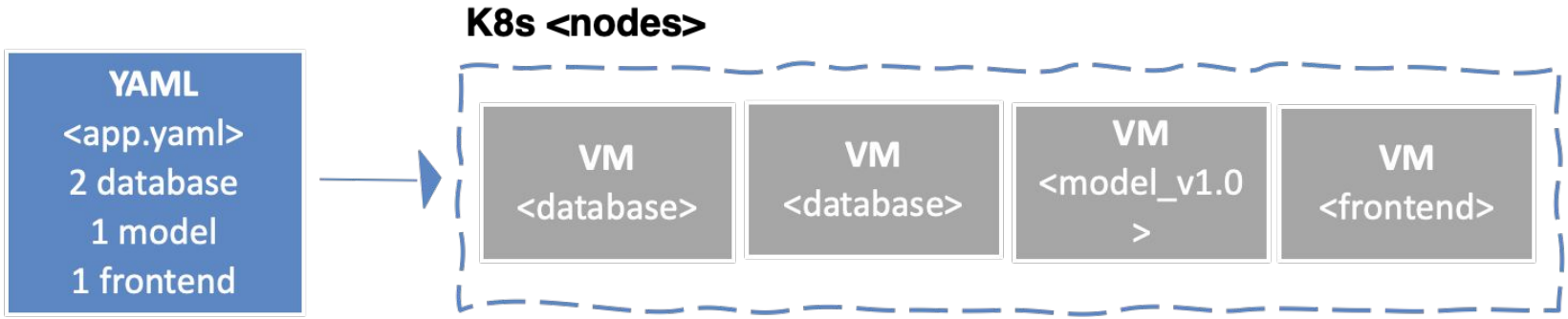


Why use kubernetes: Velocity

Velocity is enabled by:

- **Declarative configuration**: you can define the desired state of the system restating the previous declarative state to go back. Imperative configuration are defined by the execution of a series of instructions, but not the other way around.

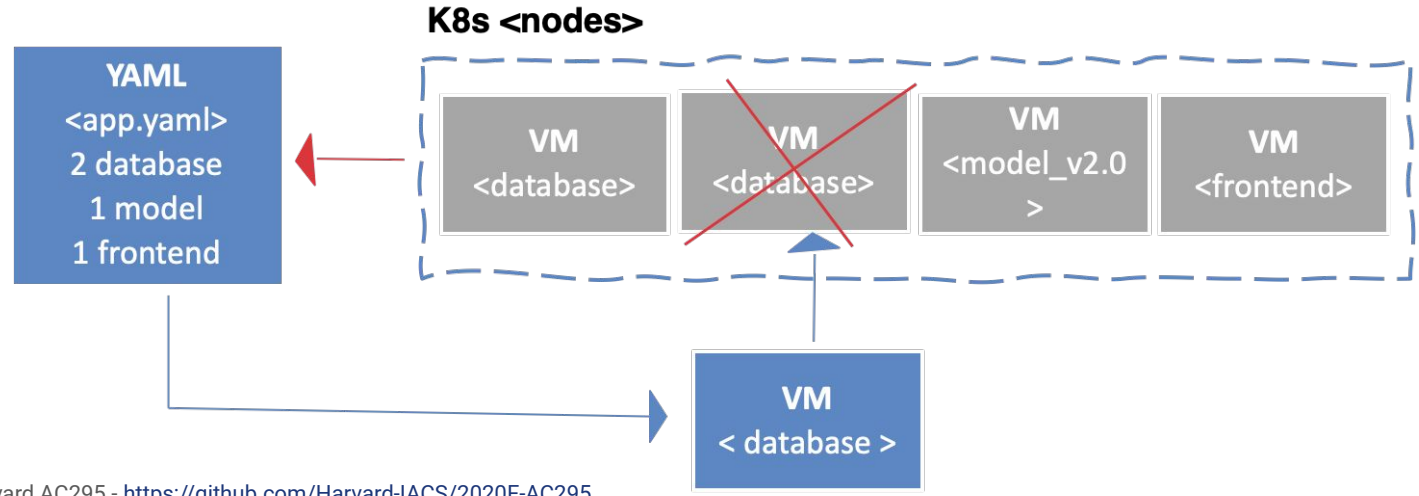
  (Just like SQL, which is also a declarative language!).



Why use kubernetes: Velocity

Velocity is enabled by:

- **Online self-healing systems**: k8s takes actions to ensure that the current state matches the desired state (as opposed to an operator enacting the repair)



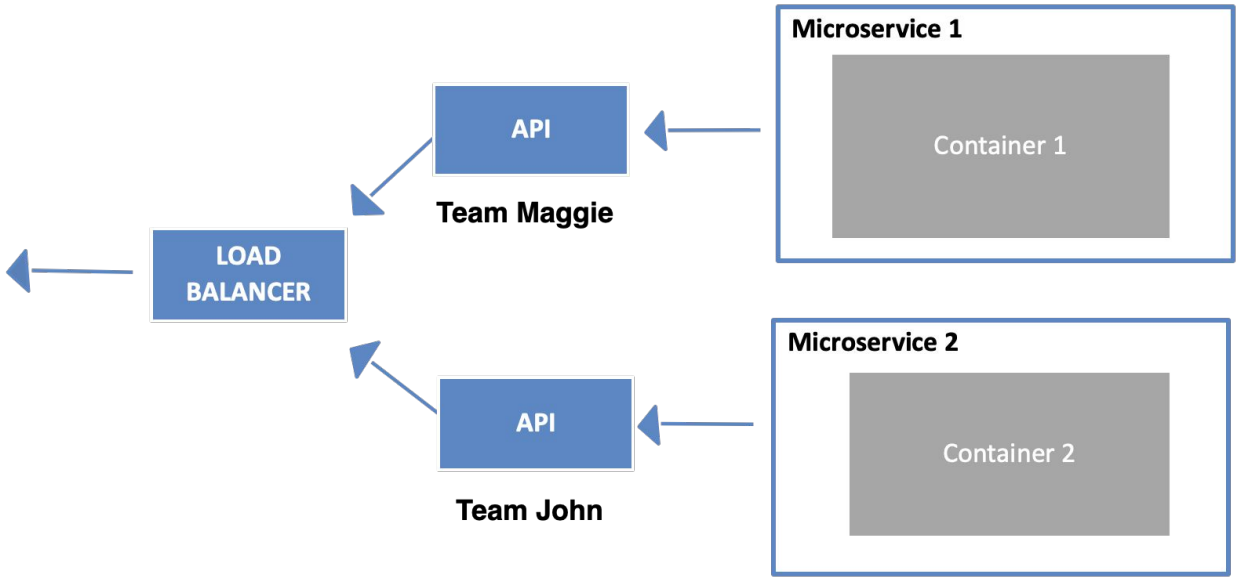
Why use kubernetes: Scaling

You can **scale** in terms of working with different **teams** on different **applications** with more **usage** with the same structure.

Kubernetes provides numerous **advantages** to address scaling:

- **Decoupled architectures**: each component is separated from other components by defined APIs and service load balancers.
- Easy scaling for **applications and clusters**: simply changing a number in a configuration file, k8s takes care of the rest (part of declarative).
- Scaling **development teams** with microservices: small team is responsible for the design and delivery of a service that is consumed by other small teams

Why use kubernetes: Scaling



The **load balancer** directs incoming network traffic (requests) to the appropriate servers (in this case, the containers running in pods) based on the service requested, the current traffic conditions and the health of the servers.

Random note: 2 pizza teams

What is the optimal number of people to work in a specific team?

https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/two-pizza-teams.html

Random note: 2 pizza teams

What is the optimal number of people to work in a specific team?

Team should be able to **be fed with two pizzas!** (~ 6 to 10 people)

Enables

- Collaboration
- Communication
- Culture fit

https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/two-pizza-teams.html

Why use kubernetes: Abstracting your infrastructure

Kubernetes allows to build, deploy, and manage your application in a way that is portable across a wide variety of environments. The move to application-oriented container APIs like Kubernetes has three concrete benefits:

- **Separation**: developers from specific machines
- **Portability**: simply a matter of sending the declarative config to a new cluster
- **Customisation**: simply a matter of sending the declarative config to a new cluster

Why use kubernetes: Efficiency

There are concrete economic benefit to the abstraction because tasks from multiple users can be packed tightly onto fewer machines

- **Consume less energy** (ratio of the useful to the total amount)
- **Limit costs of running a server** (power usage, cooling requirements, datacenter space, and raw compute power)
- Create quickly a developer's **test environment** as a set of containers
- **Reduce cost of development instances** in your stack, liberating resources to develop others that were cost-prohibitive

👆Note: Those advantages come with **scale**. If enough developers/applications share a cluster it becomes really efficient. A single small application on k8 might be overkill.

Wrap-up

Lecture summary

Topic	Concepts	Relevant for…	
		Project	**Exam**
Model testing	• Selecting a test set (split, cross-validation) • Best practices • Online testing		Yes
Containerisation	• Virtual environments • Virtual machines • Docker	Yes	Yes
Lab: Docker		Yes	
Kubernetes	• General introduction to kubernetes		
Lab: Kubernetes	• How to use k8 • Locally with minikube		