Containerisation

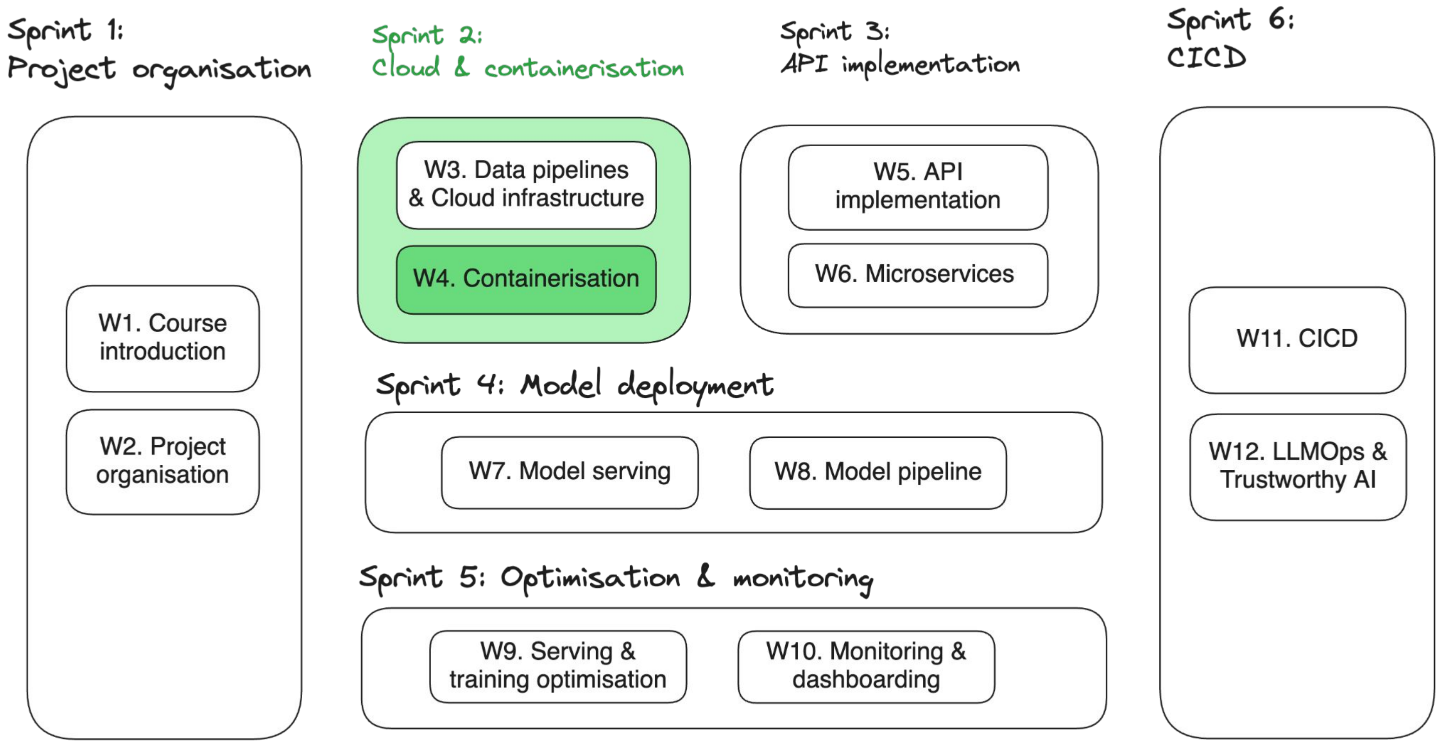
Sprint 2 - Week 4

INFO 9023 - Machine Learning Systems Design

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

2025 Spring

Status on our overall course roadmap



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

Agenda

What will we talk about today

Demo

1. Google Cloud Storage & Big Query

Lecture

2. Cloud infrastructructure (deeper dive)
3. Virtual environments
4. Virtual machines
5. Containers

Directed Work

6. Docker

Project objective for sprint 2

This course focuses on what comes **around** your ML model.
Do **not** spend significant time optimising your data or model.
No grading on the **accuracy** of the model itself.

#	Week	Work package	Requirement
2.1	W03	Prepare your data and run an Exploratory Data Analysis.	Required
2.2	W03	Prepare your Cloud environment. That means creating a Cloud project, granting correct access rights to all members of your group and setting up a billing account. **Attention**: You can have free credits for the Cloud, as explained during the course.	Required
2.3	W04	Train your ML model	Required
2.4	W04	Evaluate your ML model	Required
2.5	W03 & W04	Document your data analysis and model performance	Required

Tour of Google Cloud

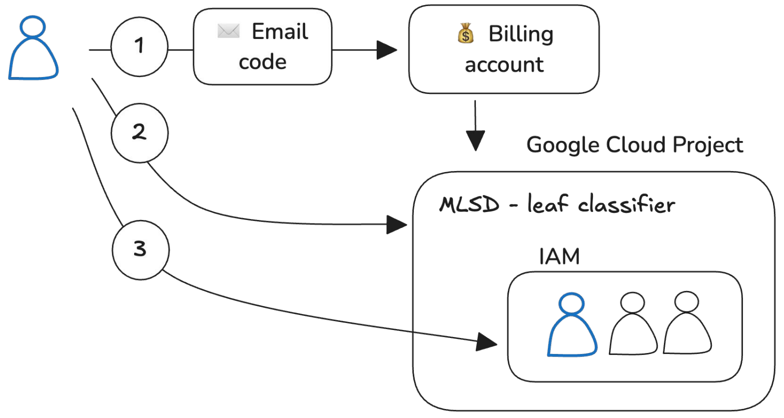
Cloud & credits

How to activate credits and manage projects

Only **one student** per group needs to create a project and activate the credits. If you run out of credits you can use the ones of someone else on your team.

Steps to setting up a project

1. Click on the link received by email to redeem credits and attach them to a billing account
2. Create a Google Cloud Project and link it to the billing account you created
3. Grant access to the other members of your team to this project through the IAM portal



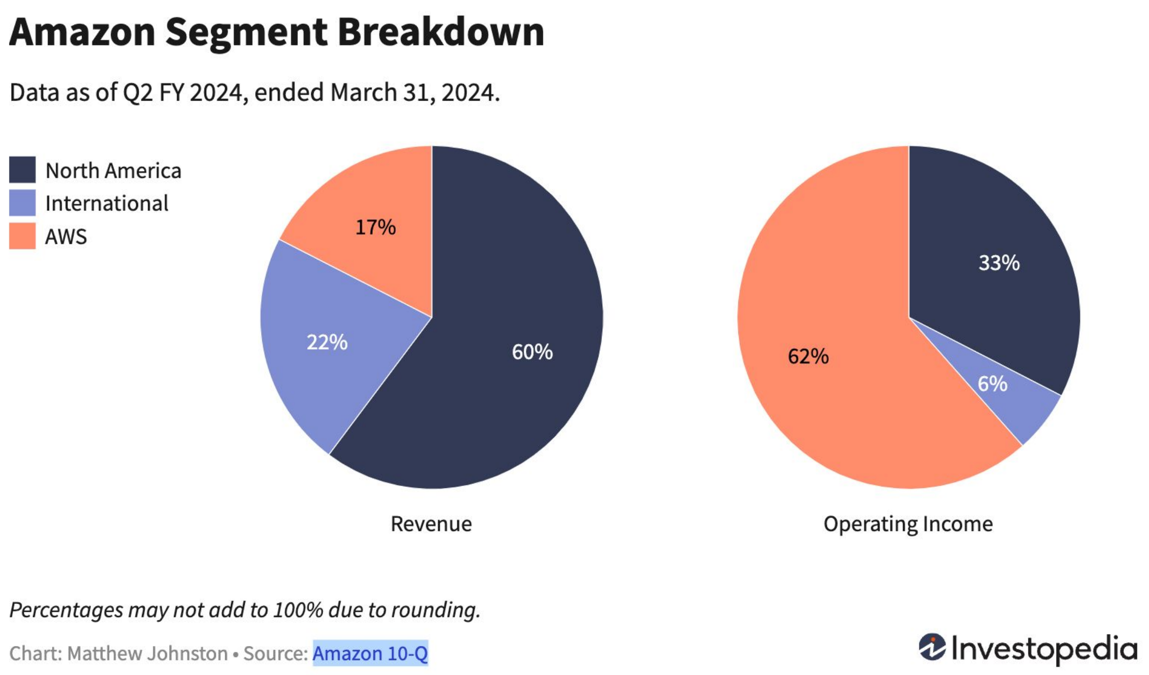
⚠️ We will do a tour of google cloud and we will demo how to setup a google cloud project!
No action needed from your side till then.

Demo: Google Cloud

Heading to Github 👉 https://github.com/ThomasVrancken/info9023-mlops/blob/main/demos/04_gcs_bq/README.md

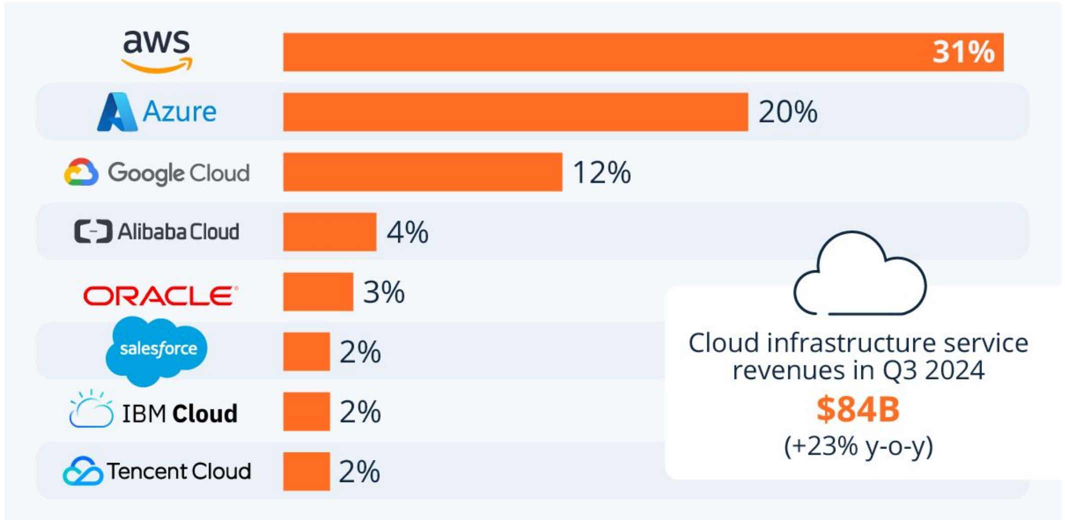
Cloud infrastructure (deeper dive)

Cloud is a profitable business.



https://www.investopedia.com/how-amazon-makes-money-4587523

Usual suspects of the “Cloud”



https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

Usual suspects of the “Cloud”

Cloud provider offer really similar services.

Overview of services types on the Cloud

1. **Infrastructure** as a Service (IaaS)
2. **Platform** as a Service (PaaS)
3. **Software** as a Service (SaaS)
4. **Function** as a Service (FaaS) (or Serverless)

Infrastructure as a Service (IaaS)

On-demand “Pay-as-you-go” **data, compute** and **networking** infrastructure.

Service	**Provider**	**Type**	**Description**
AWS S3	AWS	Data storage	Raw (blob) **storage** infrastructure to store flexible files (documents, images, videos, etc.). Accessible by applications (python sdk). Used to store large amounts of unstructured data (e.g. logs) or pass it between components of your application.
Cloud Storage (GCS)	Google		
Azure Blob Storage	Azure		
AWS EC2	AWS	Compute	Provide **virtual machines** that let you run applications and workloads in the cloud. Give you control over the operating system, software, and configuration while handling the physical hardware and infrastructure. You can SSH into the VM and run what you want.
Compute Engine	Google		
Azure VMs	Azure		

LIÈGE université

Platform as a Service (PaaS)

Delivers and *manages* hardware and software resources for developing, testing, delivering and managing cloud applications.

Compute

Service	Provider	Type	Description
Elastic Beanstalk	AWS	Application deployment	Build and host web apps, APIs, and mobile backends. Often used for front-end applications. Infrastructure is managed by Cloud provider.
App Engine	Google		
App Service	Azure		
Elastic Kubernetes Services (EKS)	AWS	Kubernetes hosting	Cloud-managed kubernetes (k8s) platform. Cloud provider hosts a kubernetes server, by opposition to hosting it on a private server or on-premise compute. Microservices can then be implemented and deployed on the kubernetes servers. The kubernetes server will then manage the scaling of the services.
Google Kubernetes Engine (GKE)	Google		
Azure Kubernetes Service (AKS)	Azure		

Platform as a Service (PaaS)

Data

Service	Provider	Type	Description
Redshift	AWS	Analytical relational database	Analytical database used to store and run aggregation operations on those data. Often used in ML applications due to the need to perform analytics.
BigQuery	Google		
Synapse Analytics	Azure		
RDS	AWS	Transactional relational database	Fixed-schema structured database. Used for transactions rather than analytics. Working with relationship tables (fact and dimension) and SQL.
Cloud SQL	Google		
SQL Database	Azure		
DynamoDB	AWS	NoSQL databases	Document and key-value data model services. Designed for high availability, scalability, and flexible data structures. Support semi-structured (JSON) data and offer low-latency, globally distributed access.
Firestore	Google		
Cosmos DB	Azure		

LIÈGE université

Platform as a Service (PaaS)

Machine Learning

Service	Provider	Type	Description
Bedrock	AWS	ML Platforms	Centralised platform for the implementation of ML models. Centered around the following offerings ● Feature engineering ● Model development ○ E.g. “workbenches” to access notebooks on GPU enabled notebooks ● Model experimentation ● Model serving & deployment ● ML pipelines ● Model monitoring
Vertex	Google		
ML Studio	Azure		

Buzzword list of this course...

Software as a Service (SaaS)

Mostly refers to software hosted on the Cloud.

Arguably, some data visualisation tools are SaaS.

Service	**Provider**	**Type**	**Description**
Tableau	AWS	Dashboarding	Build dashboards and data visuals directly on databases hosted in the Cloud (or imports such as xlsx). Low-code. Facilitate business intelligence. *Realm of data analysts.*
Looker	Google		
Power BI	Azure		

Function as a Service (SaaS) or serverless

Covered in week 06 of this course

Service	Provider	Type	Description
Fargate (or Lambda)	AWS	Serverless compute	Fully managed, serverless hosting of microservices. Host containerised applications (mostly APIs). Automatically scales the application based on demand and abstracts all infrastructure management.
Cloud Run	Google		
Container Apps	Azure		
Lambda	AWS	Function compute	Run small isolated pieces of codes (functions). Typically event based. So triggered or scheduled.
Functions	Google		
Functions	Azure		

Not a typo, just unoriginal naming...

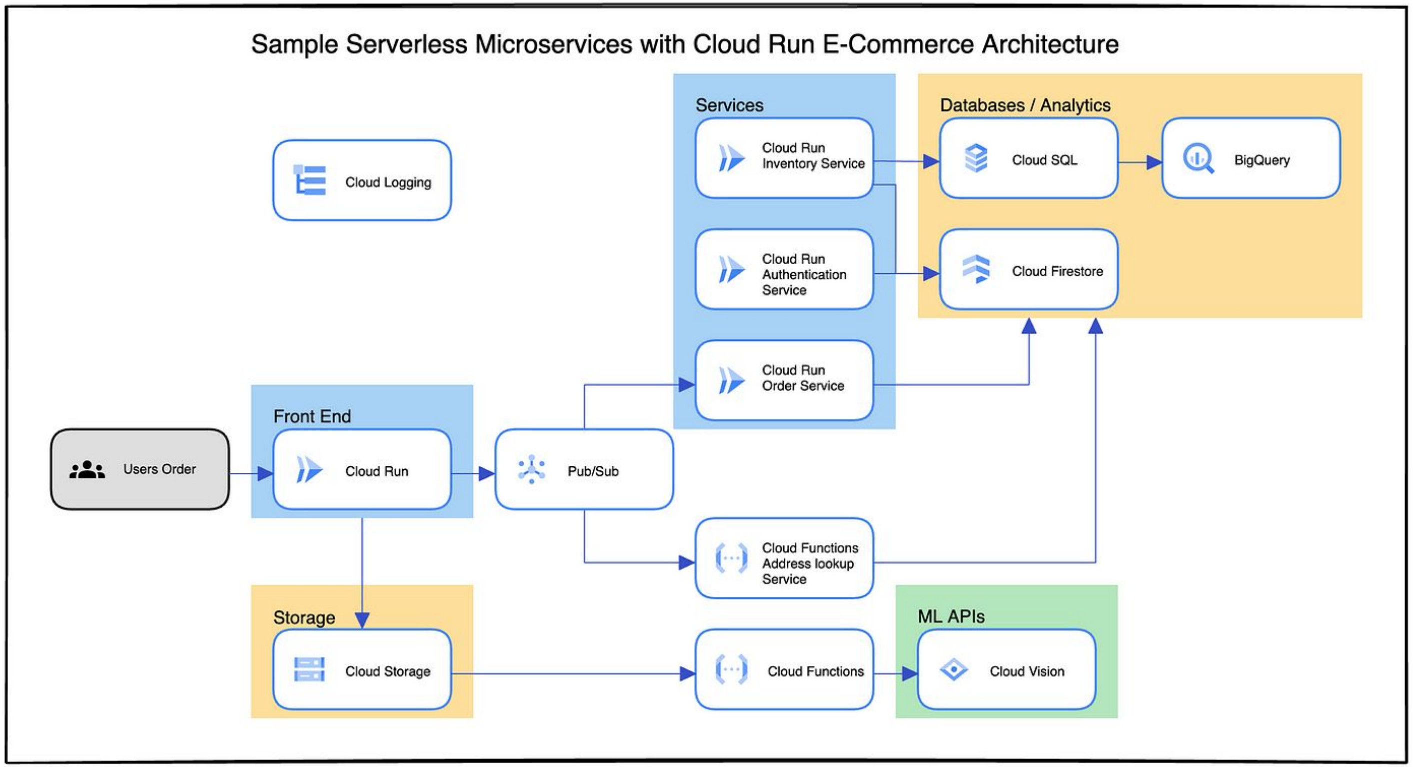
Out-of-the-box AI models.

Cloud providers also offer pre-trained AI models, such as:

- LLMs (Azure Open AI, Gemini…)
- Document AI
- Optical Character Recognition (OCR)
- Translation
- Speech-to-Text
- …

Example application architecture

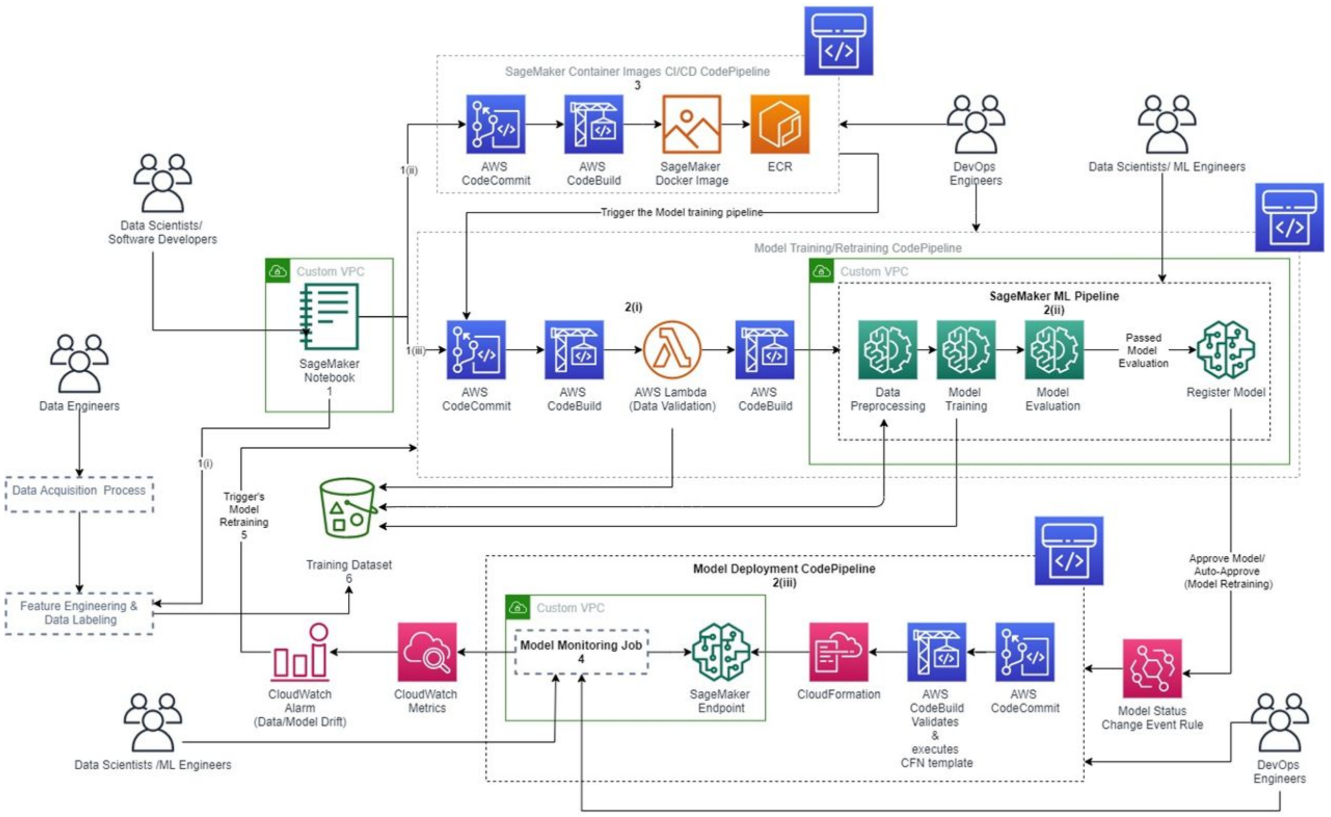
Google Cloud



https://medium.com/google-cloud/20-google-cloud-reference-architecture-to-start-your-gcp-architect-journey-2a85e8728507

Example application architecture

AWS

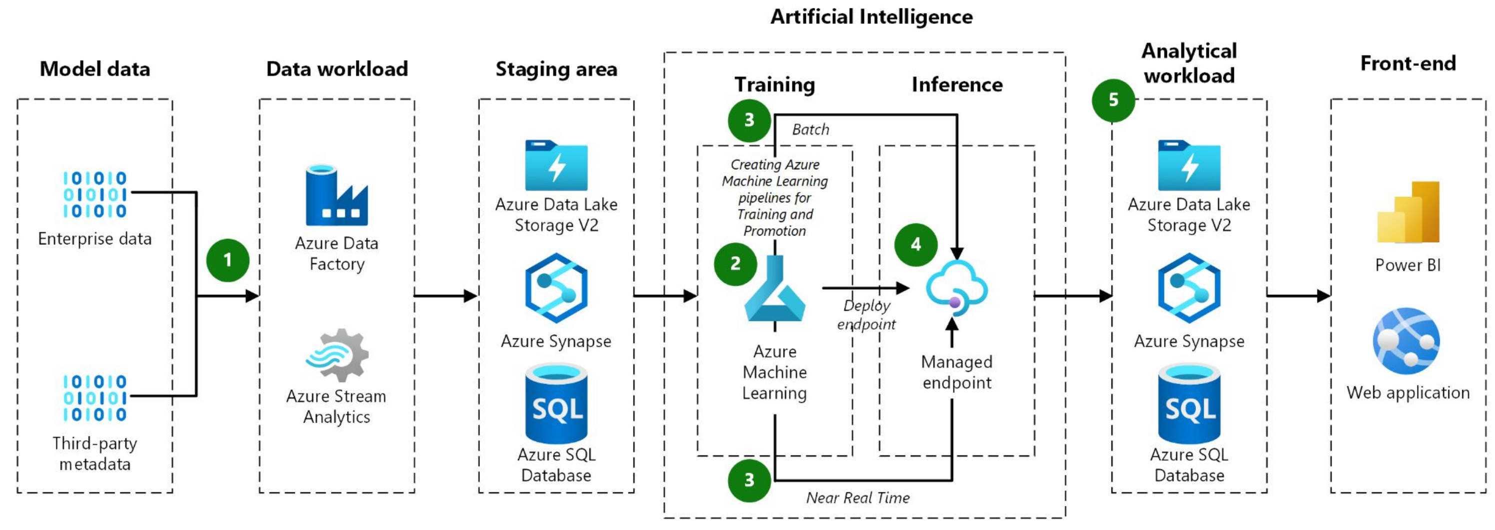


SageMaker Container Images CI/CD CodePipeline
3
AWS CodeCommit
AWS CodeBuild
SageMaker Docker Image
ECR
DevOps Engineers
Data Scientists/ ML Engineers
Data Scientists/ Software Developers
1(ii)
Trigger the Model training pipeline
Model Training/Retraining CodePipeline
Custom VPC
SageMaker Notebook
1
1(iii)
2(i)
AWS CodeCommit
AWS CodeBuild
AWS Lambda (Data Validation)
AWS CodeBuild
Custom VPC
SageMaker ML Pipeline
2(ii)
Data Preprocessing
Model Training
Model Evaluation
Passed Model Evaluation
Register Model
Data Engineers
Data Acquisition Process
1(i)
Trigger's Model Retraining
5
Training Dataset
6
Feature Engineering & Data Labeling
Approve Model/ Auto-Approve (Model Retraining)
Model Deployment CodePipeline
2(iii)
Custom VPC
Model Monitoring Job
4
SageMaker Endpoint
CloudFormation
AWS CodeBuild Validates & executes CFN template
AWS CodeCommit
Model Status Change Event Rule
CloudWatch Alarm (Data/Model Drift)
CloudWatch Metrics
Data Scientists /ML Engineers
DevOps Engineers

https://aws.amazon.com/blogs/apn/taming-machine-learning-on-aws-with-mlops-a-reference-architecture/

Example application architecture

AWS



Microsoft Azure

https://learn.microsoft.com/en-us/azure/architecture/ai-ml/idea/many-models-machine-learning-azure-machine-learning

Virtual environments

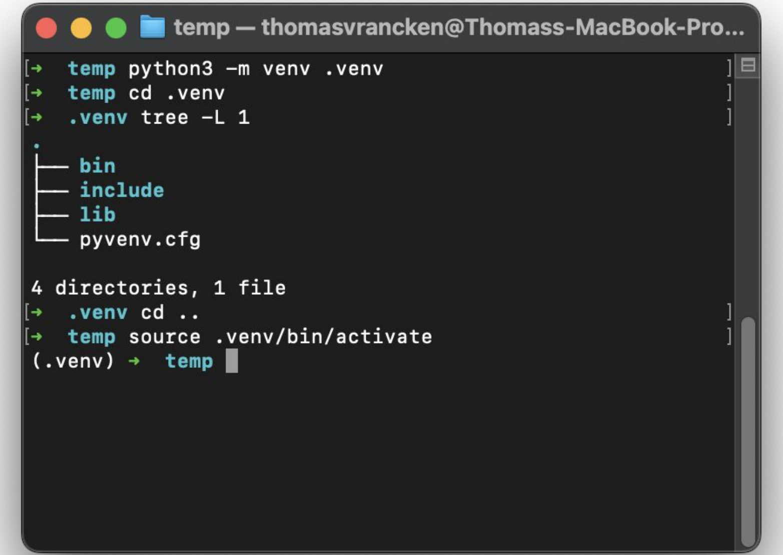
What is a virtual environment?

- Virtual environments keep **dependencies** (e.g. libraries…) in separate “**environments**” so you can switch between both applications easily with totally different packages/versions.
- 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/package versions) for **different projects**. Abstract away the dependencies by using a virtual environment.
- It will help maintaining dependencies (e.g. `requirements.txt`)

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



```
→  temp python3 -m venv .venv
→  temp cd .venv
→  .venv tree -L 1
.
├── bin
├── include
├── lib
└── pyvenv.cfg

4 directories, 1 file
→  .venv cd ..
→  temp source .venv/bin/activate
(.venv) →  temp
```

What is a virtual environment?

Here's what happens when using the virtual environment:

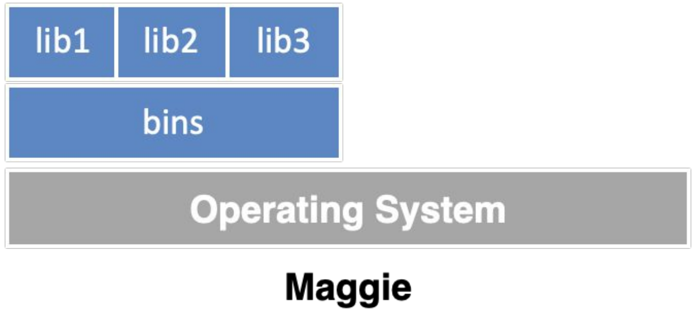
1. **Activation**: When you activate the virtual environment, your shell's PATH is updated to prioritize this bin (or Scripts) directory. This means that when you type python or pip, your shell will use the versions in the virtual environment instead of the system-wide versions.

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?

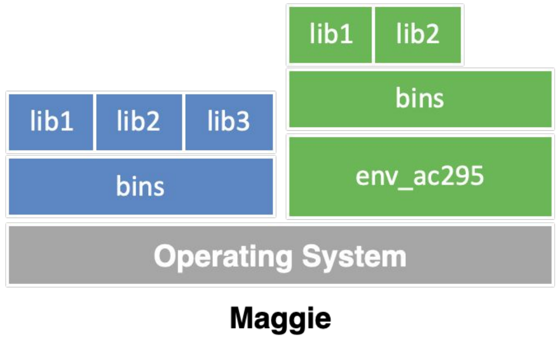
Maggie took “Intro to Machine Learning”. She used to run her Jupyter notebooks from anaconda prompt. Every time she installed a module it was placed in the either of `bin`, `lib`, `share`, `include`folders and she could import it in and used it without any issue.



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

Why should you use virtual environments?

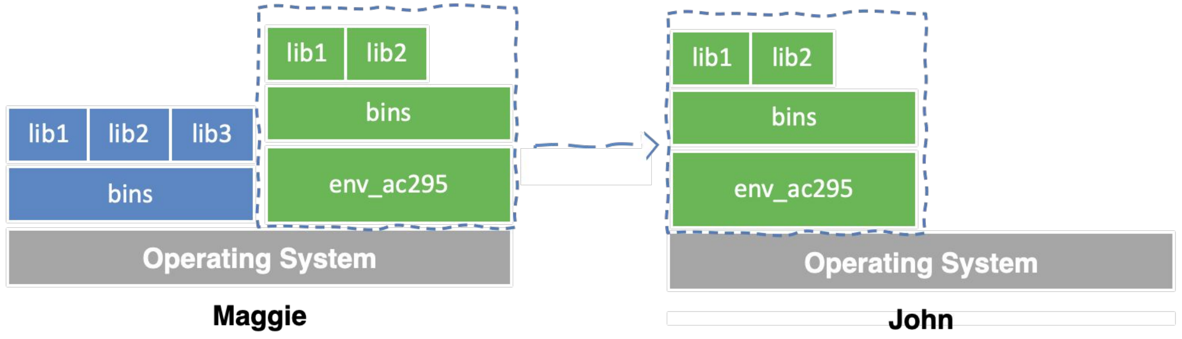
Maggie starts taking “Machine Learning Systems Design”, and she thinks that it would be good to isolate the new environment from the previous environments avoiding any conflict with the installed packages. She adds a virtual environment that helps her keep the modules organized and avoid misbehaviors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



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

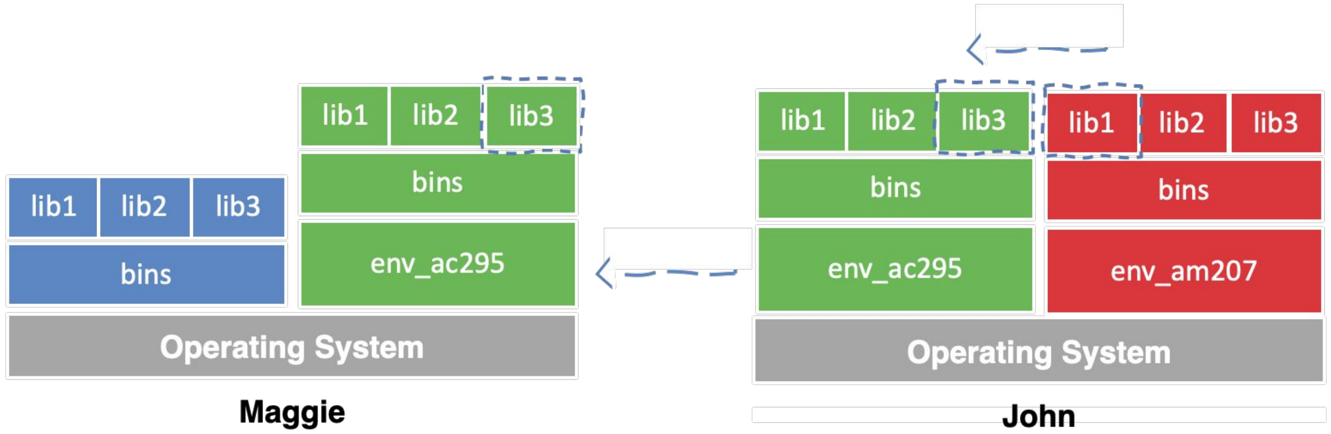
Why should you use virtual environments?

Maggie collaborates with John for the final project and shares the environment she is working on through `requirements.txt` 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 `requirements.txt` 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

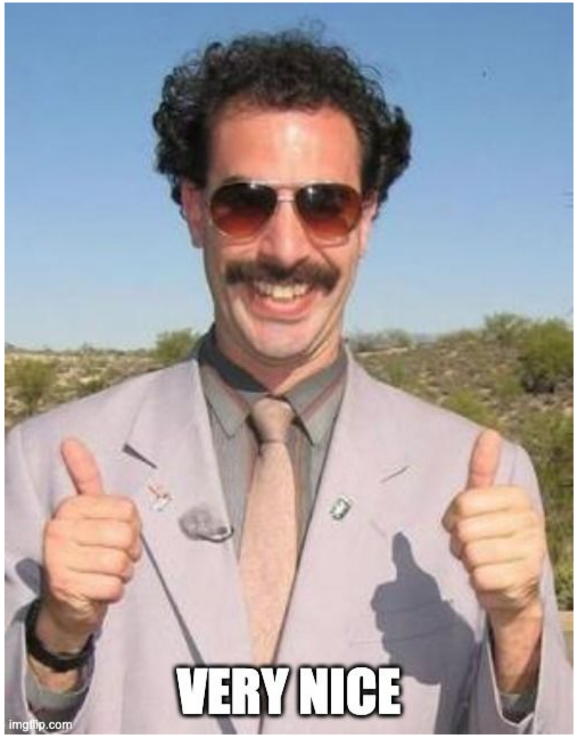
`conda` **environment**

- 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?

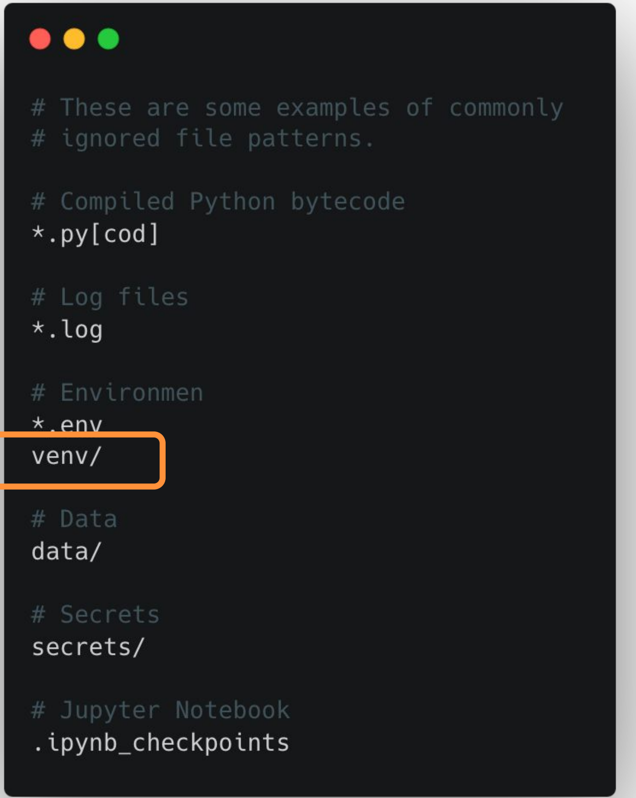
Pros

- Reproducible research
- Dependency management (forces you to maintain a requirements.txt with <u>all</u> 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)

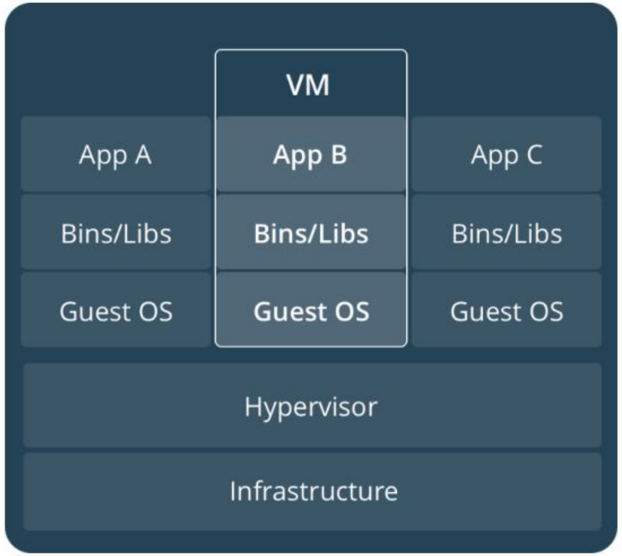
Virtual Machine (VM)

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

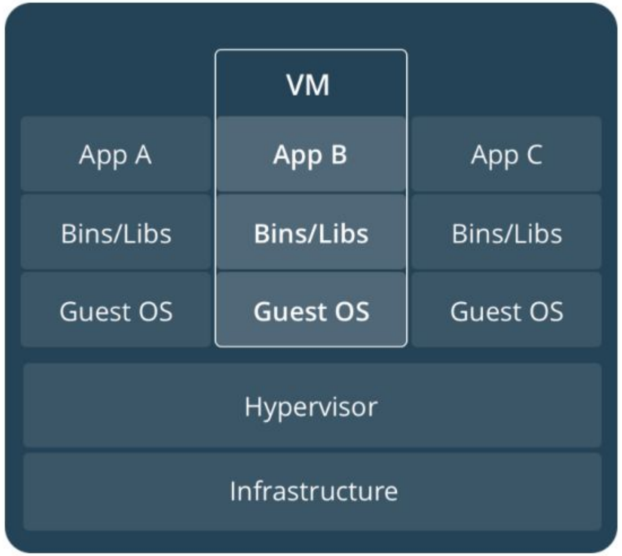
- We have our isolated systems, and after we set up the environment with our colleagues' machine, we expect to get identical results, right? Unfortunately, it is not always the case. Why? Most likely because we run on a different OS.
- Even though using virtual environments, we isolate our computations, 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])



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

The best OS to exist was Windows XP, it could run on 64Mb of ram and had a talking paperclip
OK GRANDPA LET'S GET YOU BACK TO BED
imgflip.com

Virtual Machine (VM)

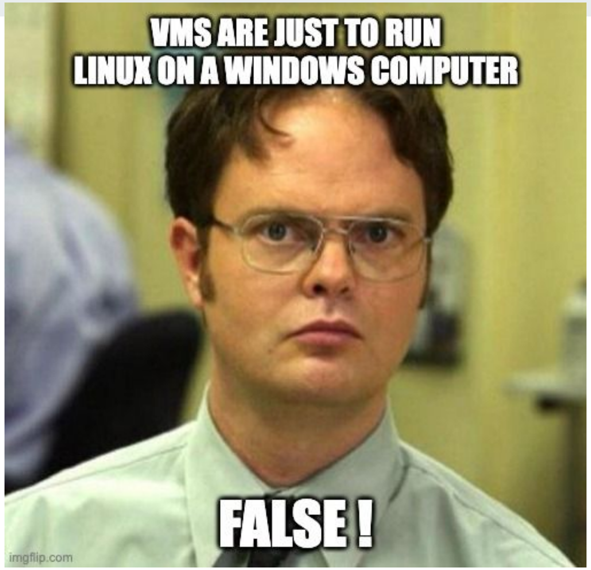
Pros

- **Isolation**: 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.

Cons

- 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

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
    - Often you cannot download customer data on your own machine

Amazon **EC2**

Try it yourself

AzureVM

Try it yourself

Google Compute Engine

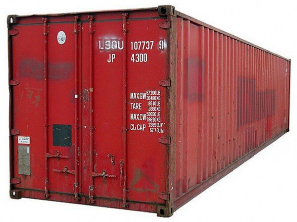
Try it yourself

LIÈGE université

Containerisation

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

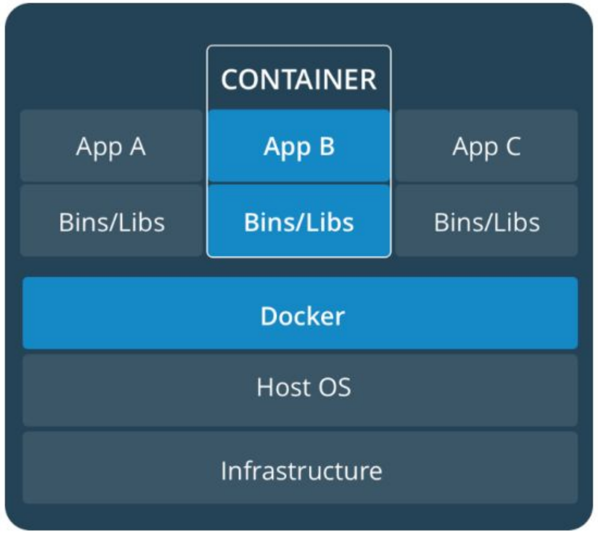
Containers



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.

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.

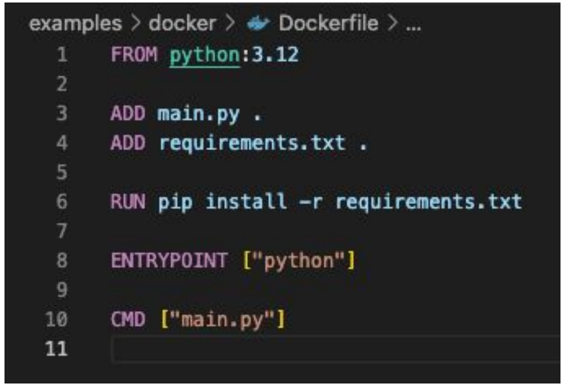
Image is like a **recipe**, container is like a **dish.**

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?

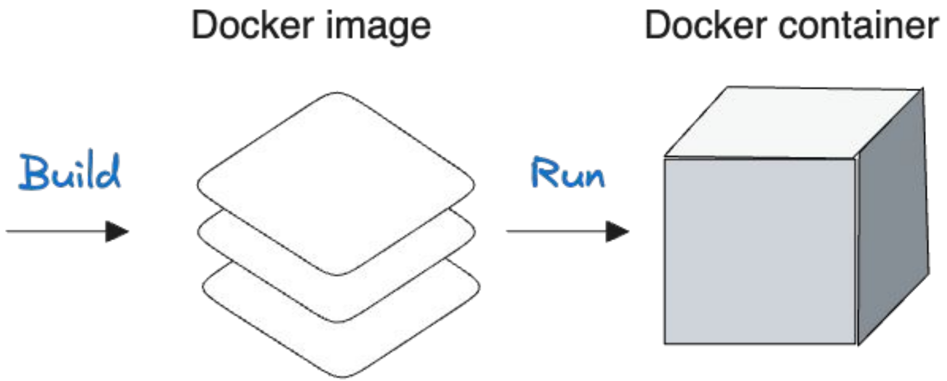
We use the **Dockerfile**, a simple text file, to configure and build the Docker Image. We run the Docker Image to get Docker Container.

Docker file

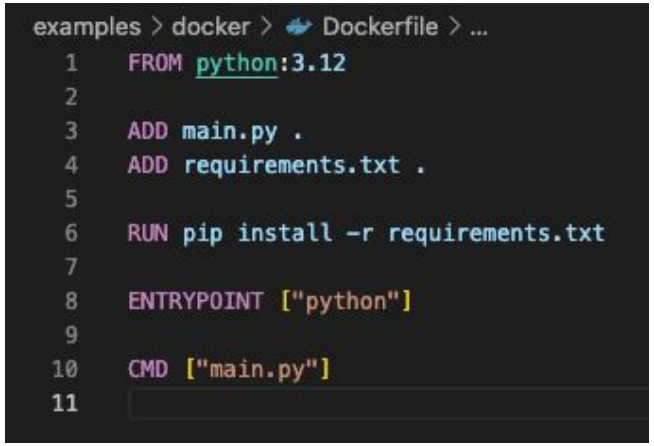


examples > docker > Dockerfile > ...

```
FROM python:3.12

ADD main.py .
ADD requirements.txt .

RUN pip install -r requirements.txt

ENTRYPOINT ["python"]

CMD ["main.py"]

```



Looking inside a Dockerfile



```
FROM python:3.12

ADD main.py .
ADD requirements.txt .

RUN pip install -r requirements.txt

ENTRYPOINT ["python"]

CMD ["main.py"]
```

`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.

Popular base docker images

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

Multiple containers from same image

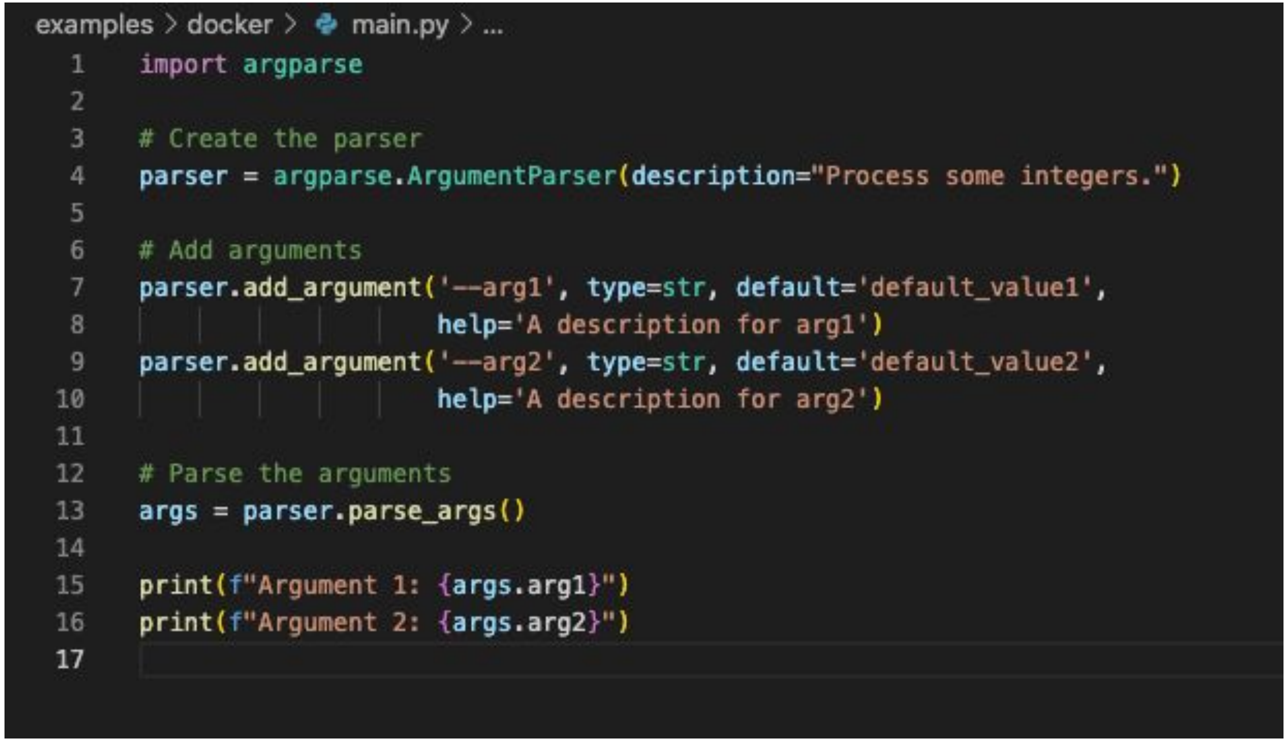
How can you run multiple containers from the same image? Wouldn’t they all be identical?

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).

Multiple containers from same image

Python file with multiple arguments

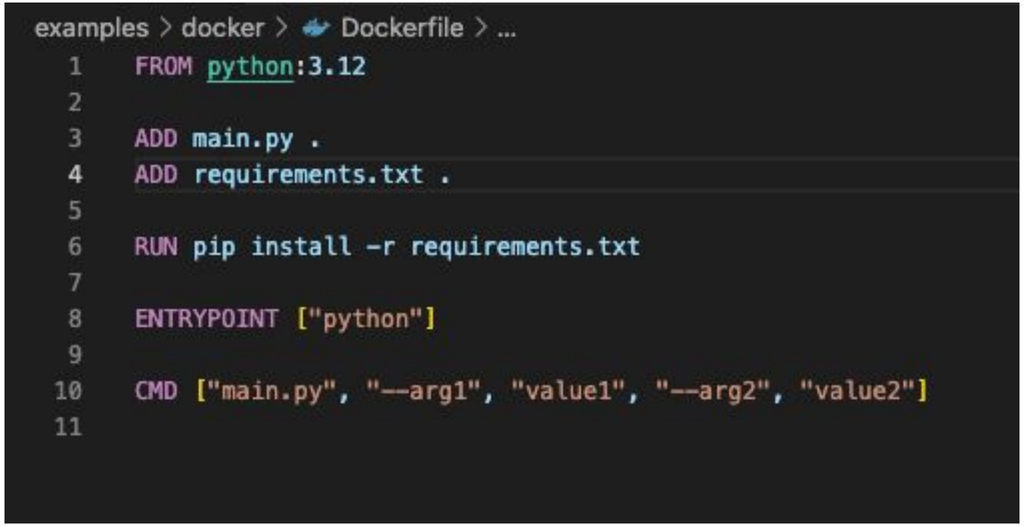


examples > docker > main.py > ...

```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

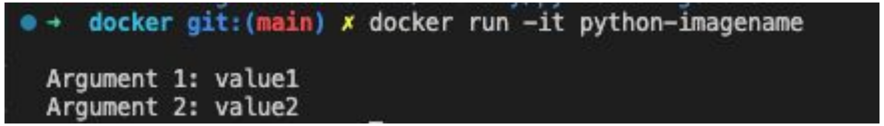
Dcokerfile with multiple CMD options


examples > docker > Dockerfile > ...

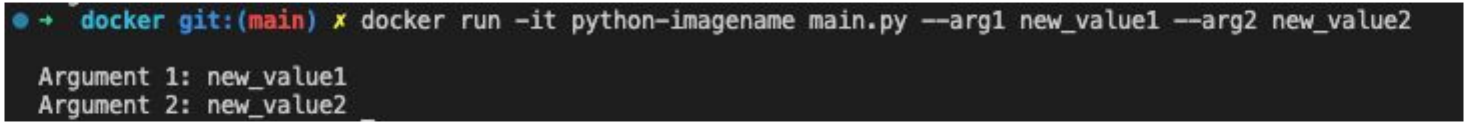
```dockerfile
FROM python:3.12

ADD main.py .
ADD requirements.txt .

RUN pip install -r requirements.txt

ENTRYPOINT ["python"]

CMD ["main.py", "--arg1", "value1", "--arg2", "value2"]
```

Multiple containers from same image

Can change python arguments upon docker run

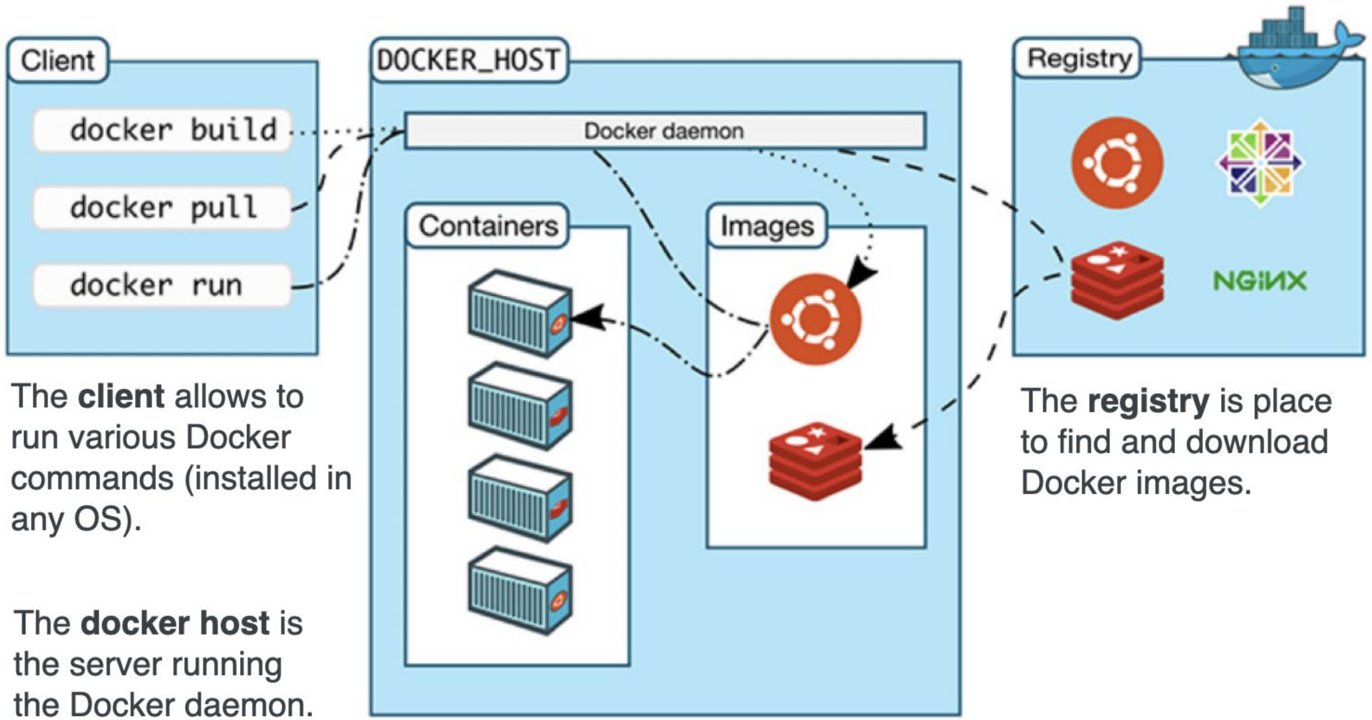


```
● → docker git:(main) ✗ docker run -it python-imagename

  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
```

Docker client, host and registry



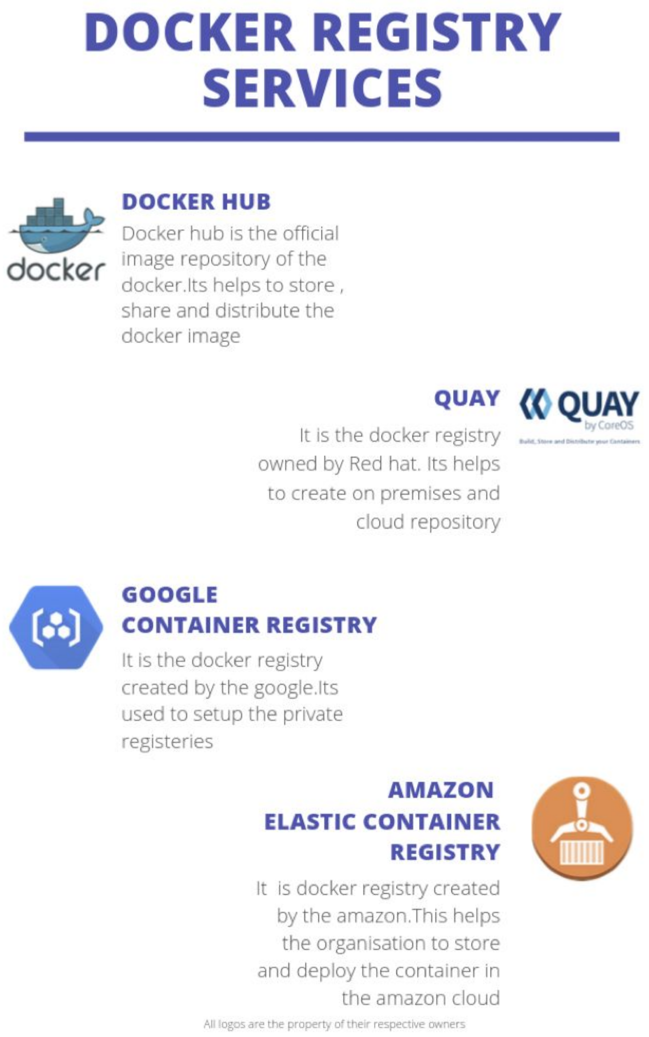
The **client** allows to run various Docker commands (installed in any OS).

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

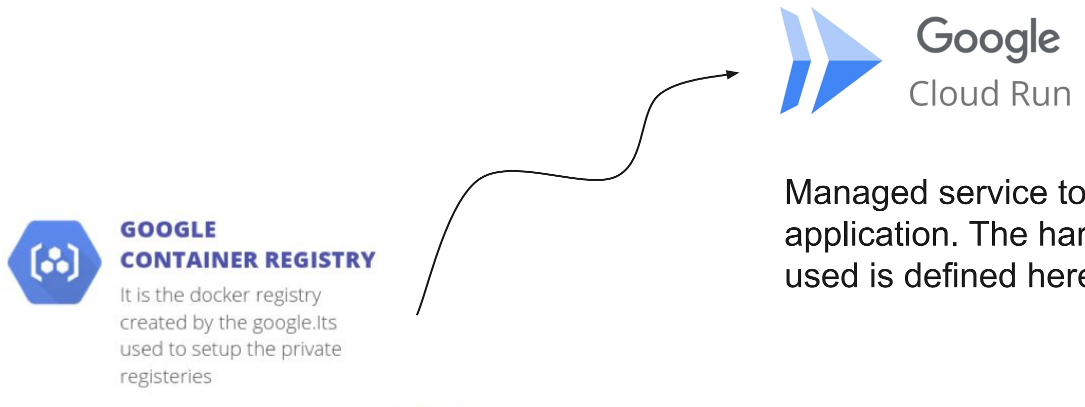
It is the docker registry created by the google.Its used to setup the private registeries

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

All logos are the property of their respective owners

Docker registry services



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

Stores your docker image, which includes the codes to run your application

Managed service to run your application. The hardware used is defined here.

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

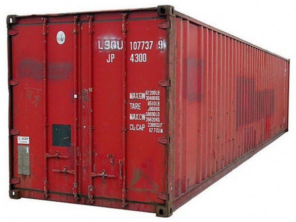
The advantage of using containers is that they only virtualize the operating system and do not require dedicated piece of hardware because they share the same kernel of the hosting system.

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.
- 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.

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.

… 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

Full overview

	Virtual environment	Virtual Machine	Docker
Effort	Easy	High at beginning, then low	Medium
Versatility	Medium	High	High
Portability	Medium	High	High

Lab: Docker

Wrap-up

Project objective for sprint 2

This course focuses on what comes **around** your ML model.
Do **not** spend significant time optimising your data or model.
No grading on the **accuracy** of the model itself.

#	**Week**	**Work package**	**Requirement**
2.1	W03	Prepare your data and run an Exploratory Data Analysis.	Required
2.2	W03	Prepare your Cloud environment. That means creating a Cloud project, granting correct access rights to all members of your group and setting up a billing account. **Attention**: You can have free credits for the Cloud, as explained during the course.	Required
2.3	W04	Train your ML model	Required
2.4	W04	Evaluate your ML model	Required
2.5	W03 & W04	Document your data analysis and model performance	Required

LIÈGE université

Lecture summary

Topic	Concepts	Relevant for…	
		Project	**Exam**
Cloud infrastructure	● Cloud providers ● Cloud services	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		

That’s it for today!