LIÈGE université

Containerisation

Sprint 2 | Week 4

INFO9023 - ML Systems Design

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


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

Agenda

What will we talk about today

Lecture

1. Exploratory Data Analysis (EDA)
2. Cloud infrastructure (deeper dive)
3. Virtual environments
4. Virtual machines
5. Containers

Directed Work

6. Docker

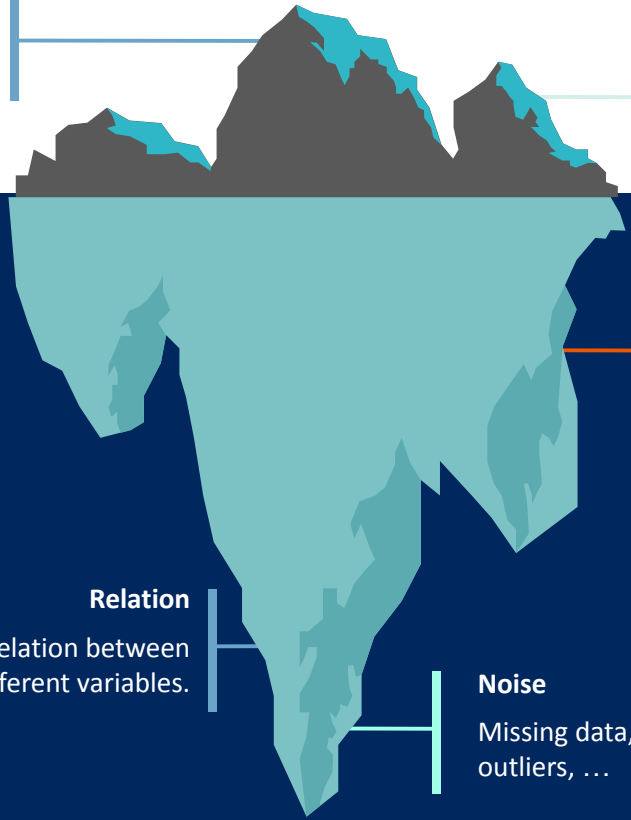
Exploratory Data Analysis

LIÈGE université

Assuming you understand your data is a common pitfall

Looking at a few observations

Reading a few rows and assuming it's representative



Looking at a few columns

Assuming from a column's title that you understand what it represents

Distribution

Skewed or unexpected distribution.

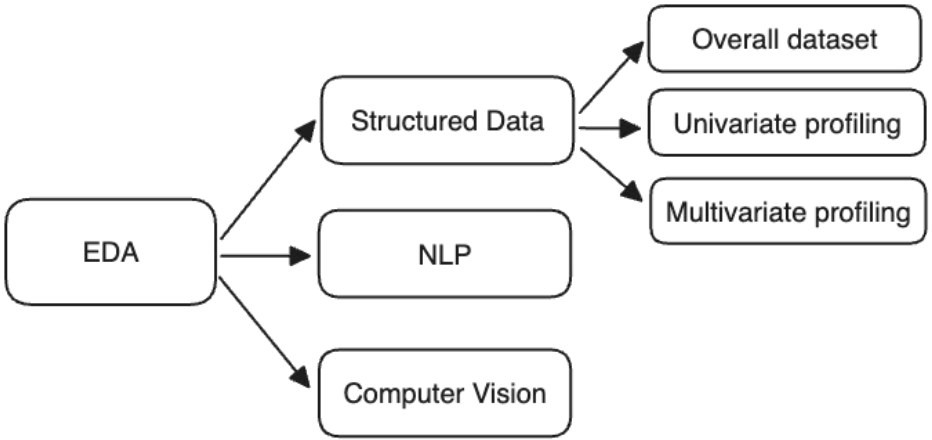
Relation

Unknown relation between different variables.

Noise

Missing data, formatting issue, outliers, …

Different types of data to explore and steps to take



EDA on structured data

First, know that there are different tools to use for the EDA

- Doing it yourself: **Pandas** (or else)
  - More freedom / customisation
  - More effort
  - Risk of overlooking important aspects
- Useful tool: **YData Profiling**
  - More managed
  - Risk of not digging deep enough into specific cases

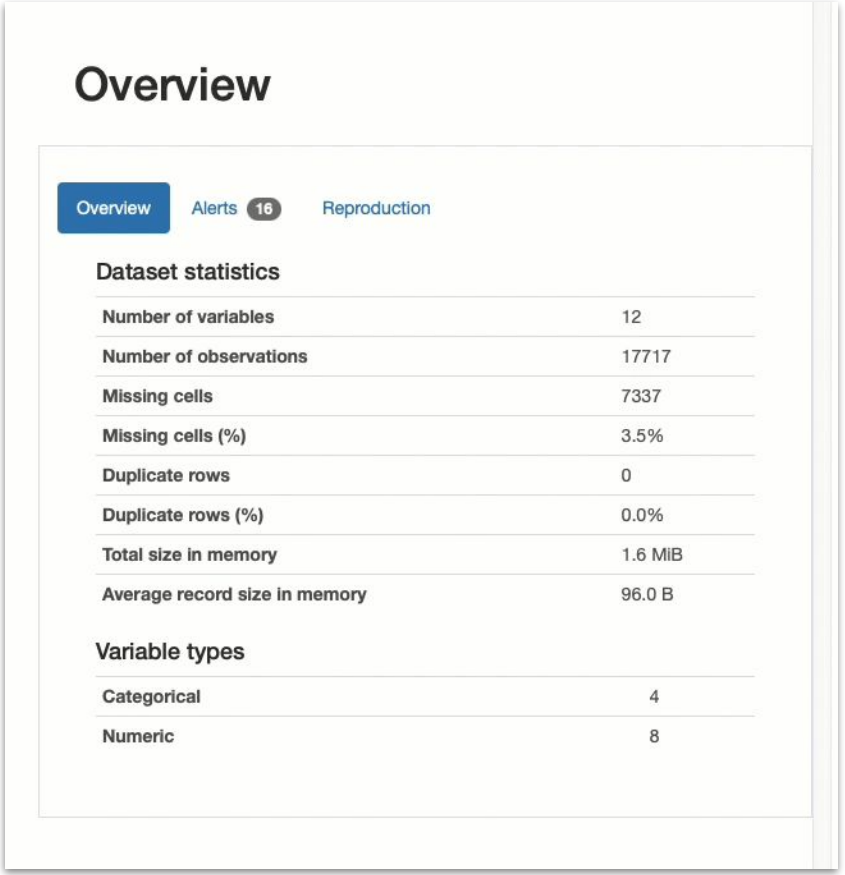
```
import pandas as pd
from pandas_profiling import ProfileReport


# Read the HCC Dataset
df = pd.read_csv("hcc.csv")

# Produce the data profiling report
original_report = ProfileReport(df, title='Original
Data')
original_report.to_file("original_report.html")
```


https://docs.profiling.ydata.ai/4.6/ – https://pandas.pydata.org/

Dataset sanity and noise

Look for

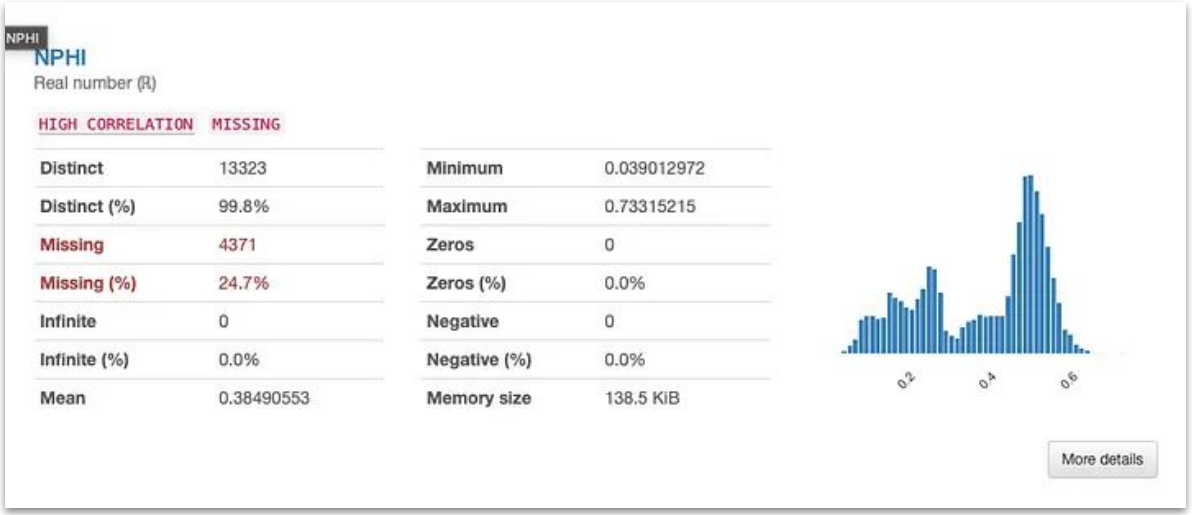
- General statistics
  - Memory size
  - Number of rows
  - Column types
  - …
- Duplicated columns
  - `df.duplicated(subset=[columns])`
- Rows with many missing values



https://towardsdatascience.com/5-powerful-python-libraries-you-need-to-know-to-enhance-your-eda-process-f0100d563c16

Univariate profiling

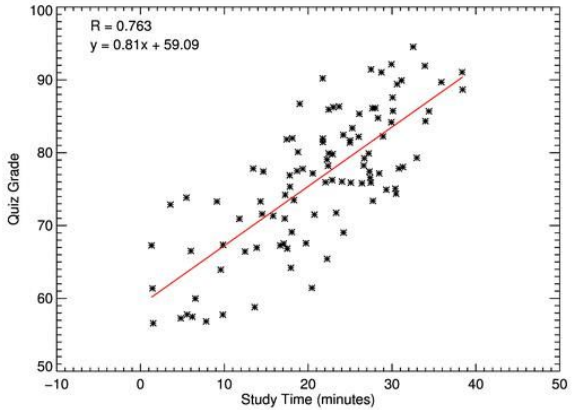
- Understanding **independent** variables

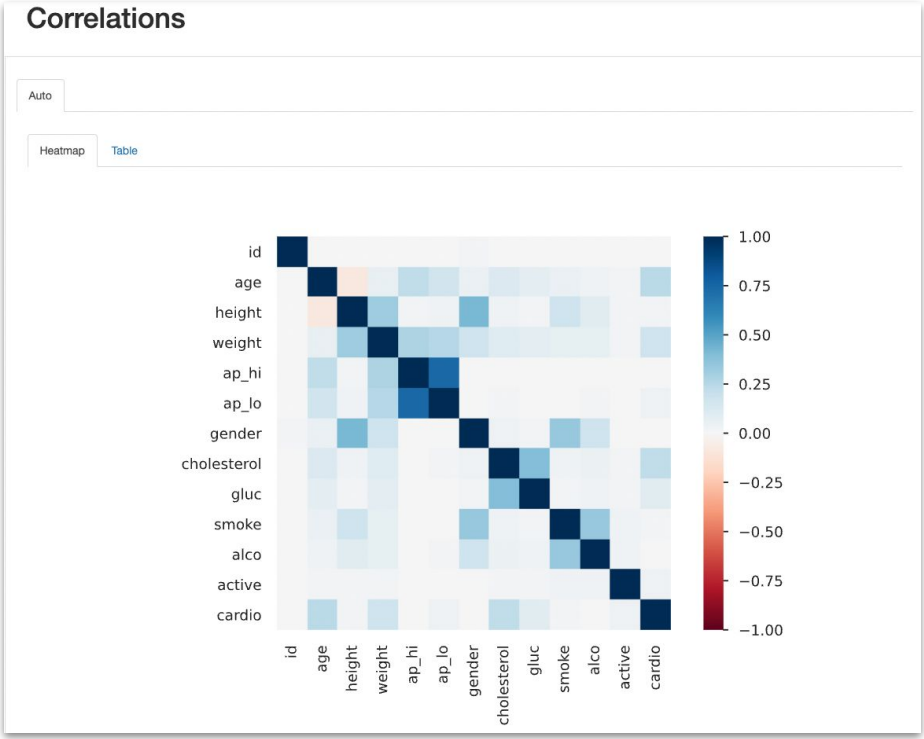


https://towardsdatascience.com/5-powerful-python-libraries-you-need-to-know-to-enhance-your-eda-process-f0100d563c16

Multivariate profiling

- Relationship between different variables
- Plot **pair of variables** against each other
- Plot **correlation matrix**



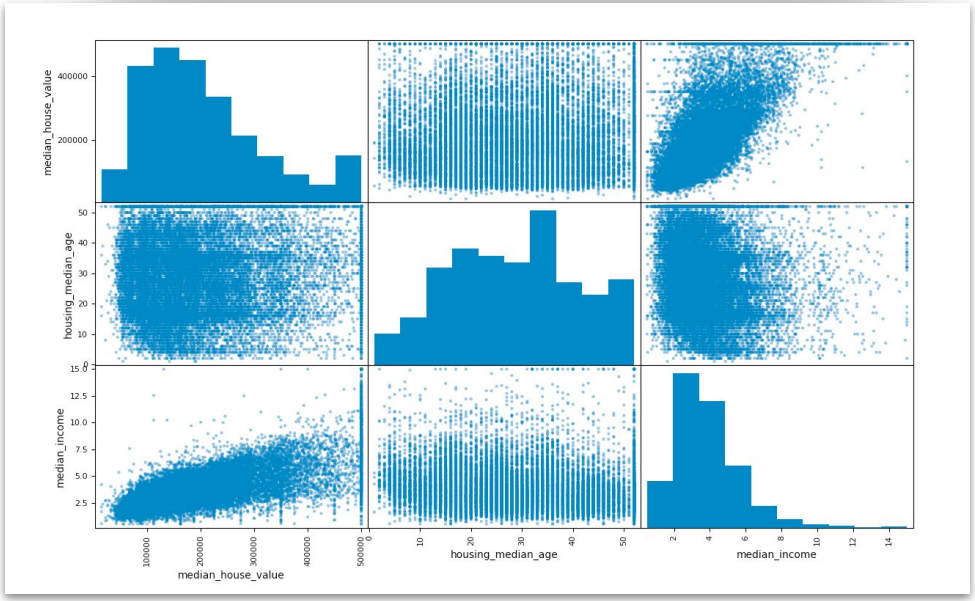


Ajibade, Samuel & Taiwo, Fele & Adedara,. (2016). A Systematic Review of Data Visualization Methods.

Multivariate profiling

- Plot **matrix** of **scatter plots** for each pair of variables

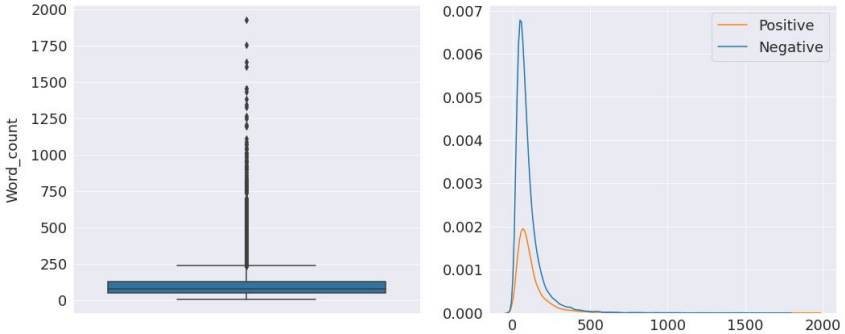
```python
from pandas.plotting import scatter_matrix

df = pd.DataFrame(np.random.randn(1000, 4), columns=["a", "b", "c",
"d"])
scatter_matrix(df, alpha=0.2, figsize=(6, 6), diagonal="kde");
```

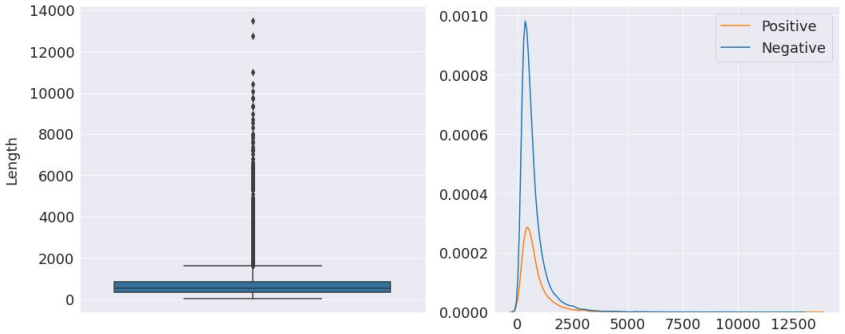


https://pandas.pydata.org/docs/user_guide/visualization.html#scatter-matrix-plot

EDA on natural language

- Statistical distribution of your texts
  - *Useful to understand token size distributions when using APIs (embedding, generation, etc.)*





EDA on natural language

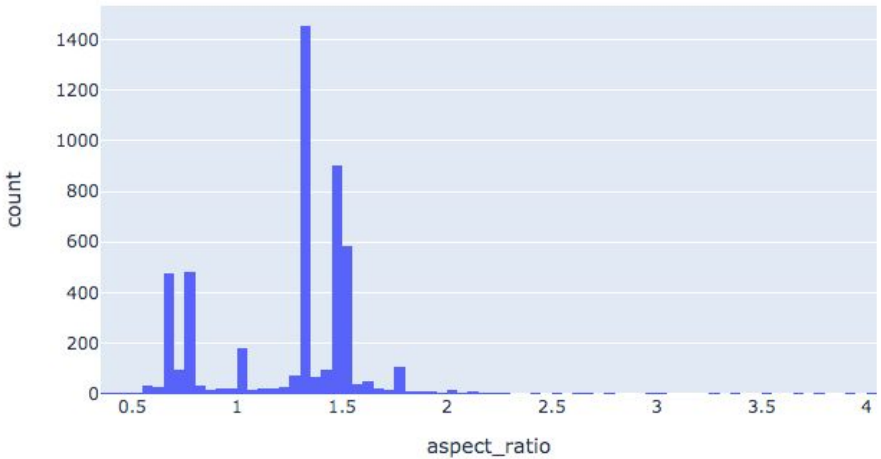
Organically explore and read through **data samples** of texts

- General understanding
- Get a feeling of amount of noise or cleaning steps required (e.g. run into HTML formatted data)
- Easy to develop a biased understanding! Can’t go through a significant sample

EDA on images

Look at image **sizes** and **aspect ratios**.

Can help you decide on **type of resizing** (destructive or not, desired output resolution, padding, …).



https://neptune.ai/blog/data-exploration-for-image-segmentation-and-object-detection

EDA on images

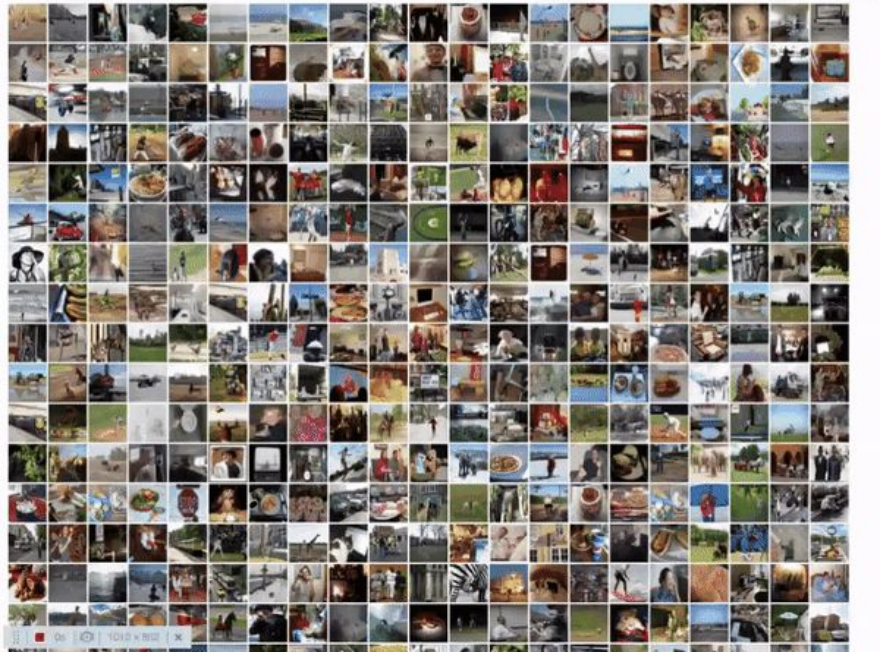
Organically explore and look through data samples of images

- General understanding
- Get a feeling of amount of noise or specific cleaning steps required (e.g. see specific ratio issues)
- Easy to get biased understanding! Can’t go through a significant sample

Can be done with

- Matplotlib
- A dedicated tool like Google Facets
- HTML rendering to visualize and explore in a notebook.

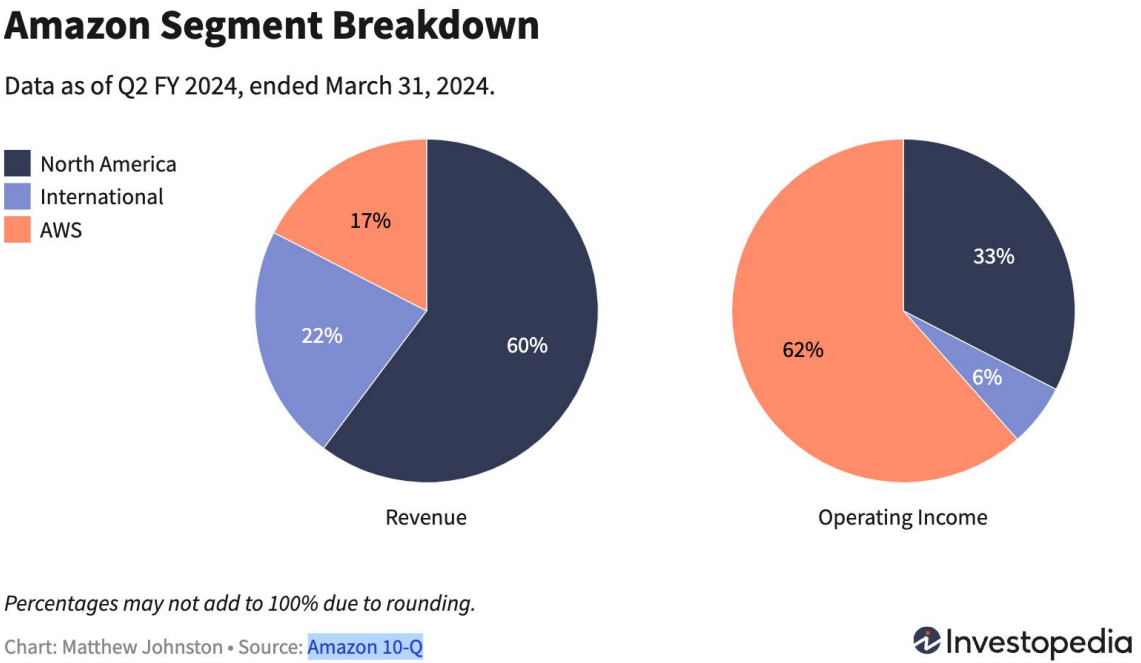
```
from easyimages import EasyImageList

Li = EasyImageList.from_folder('/Users/jcieslik/Downloads/val2017/')
Li.symlink_images()
Li.html(sample=500, size=44)
```



LIÈGE université

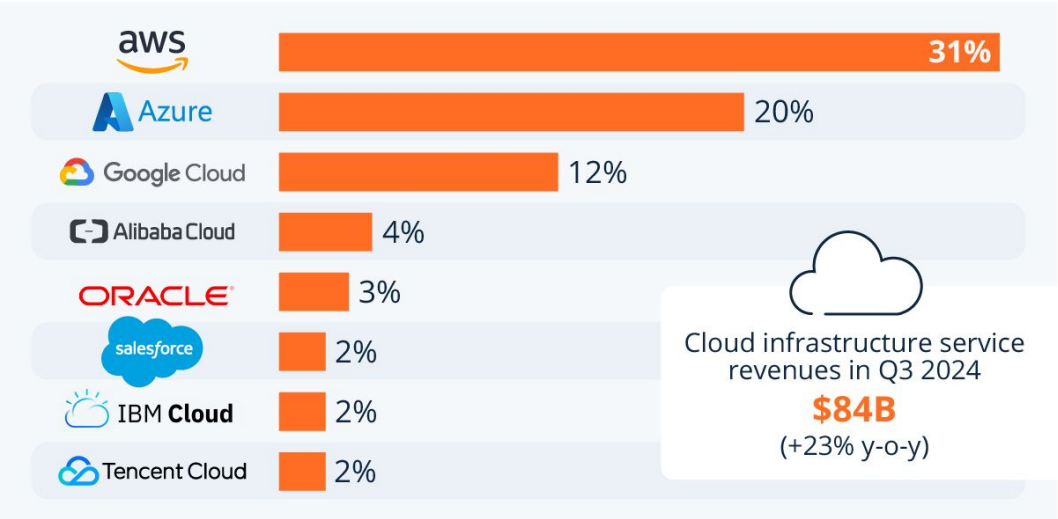
Cloud infrastructure (deeper dive)

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/

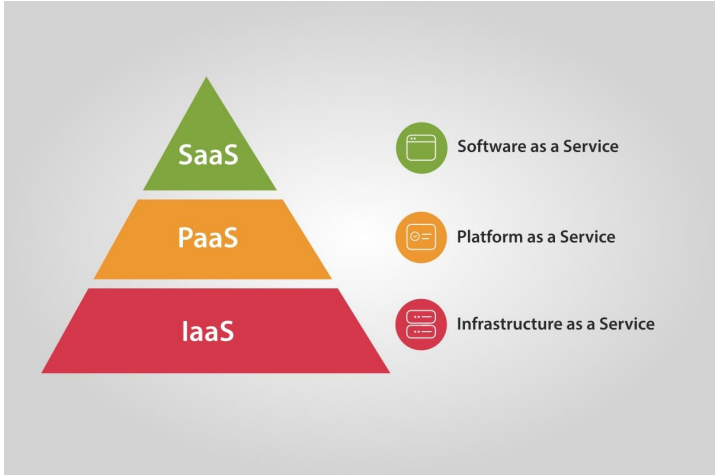
Usual suspects of the “Cloud”

Cloud providers offer similar services.

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

Overview of services types on the Cloud

- **IaaS**: Raw building blocks: virtual machines, networking, and storage.
- **PaaS:** Managed environment to run your applications, like hosting a web API or a database, without worrying about the underlying infrastructure.
- **SaaS**: Fully finished applications that you can just use, like email, chat, or an AI model accessible via API.



https://mlops4ecm.be/handleidingen/ml-infrastructure/#5-infrastructure-management-cloud-on-premise-and-infrastructure-as-code

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		

Platform as a Service (PaaS)

Compute

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 - triggered or scheduled.
Functions	Google		
Functions	Azure		

Not a typo, just unoriginal naming...

Platform as a Service (PaaS)

Data

Service	Provider	Type	Description
Redshift	AWS	Relational database - OLAP	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	Relational database - OLTP	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		

Platform as a Service (PaaS)

Machine Learning

Service	Provider	Type	Description
Sagemaker	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...

Platform as a Service (PaaS)

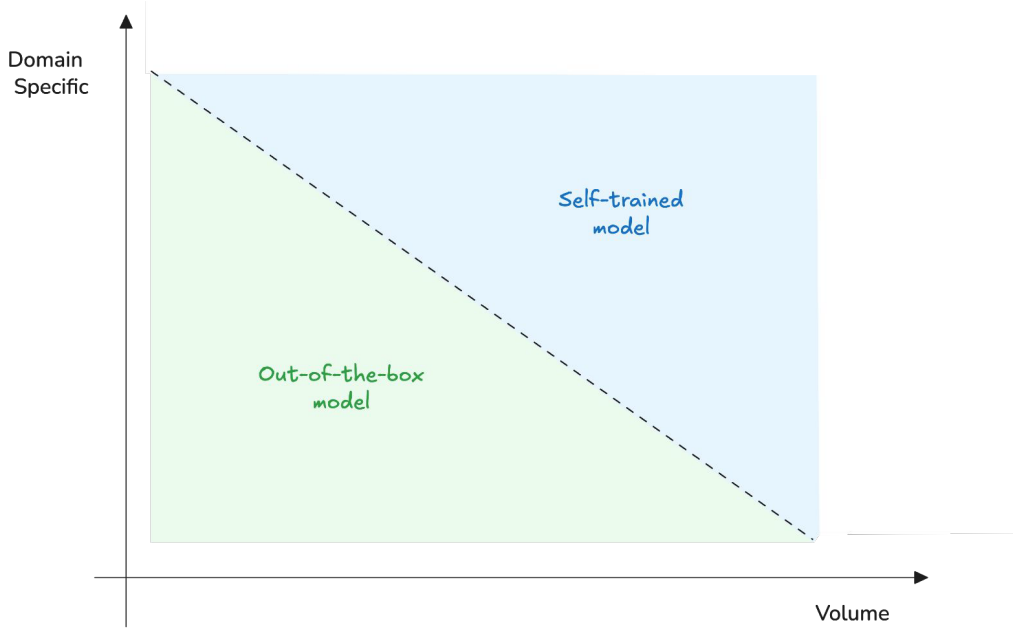
AI models

Service	Provider	Type	Description
Bedrock	AWS	AI models	Provide AI models (usually generative) through paid APIs. Models can be: • Large Language Models (LLMs): Call GPT, Gemini… and more models. Can be multi-modal. • Document processing: For example processing of receipt scans. • Optical Character Recognition (OCR): Extract text from images • Translation: Translate text • Speech-to-text (STT): Parse audio into text • … and more
Vertex	Google		
Azure OpenAI	Azure		
OpenAI	OpenAI		

Software as a Service (SaaS)

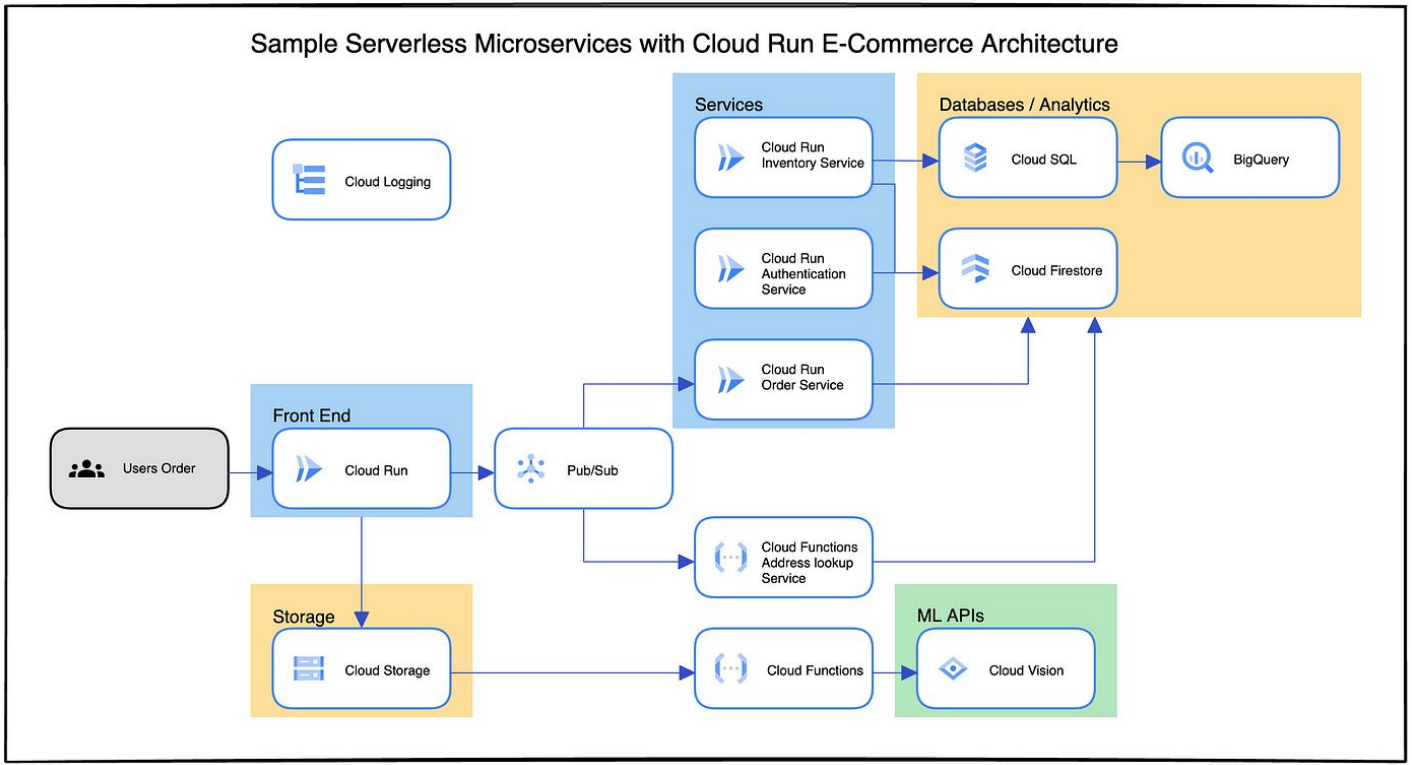
AI models

When to opt for out-of-the-box generalised models or when to train your own.



Example application architecture

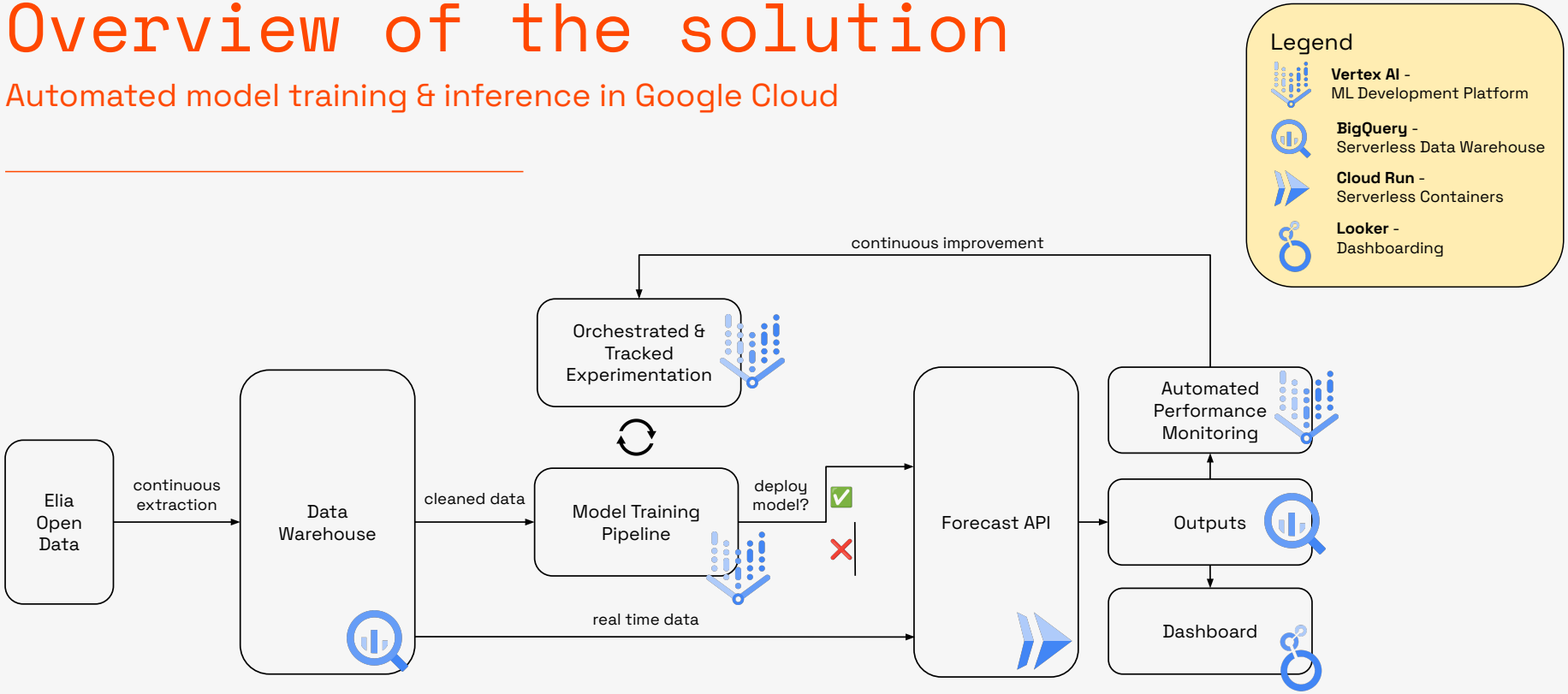
Google Cloud



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

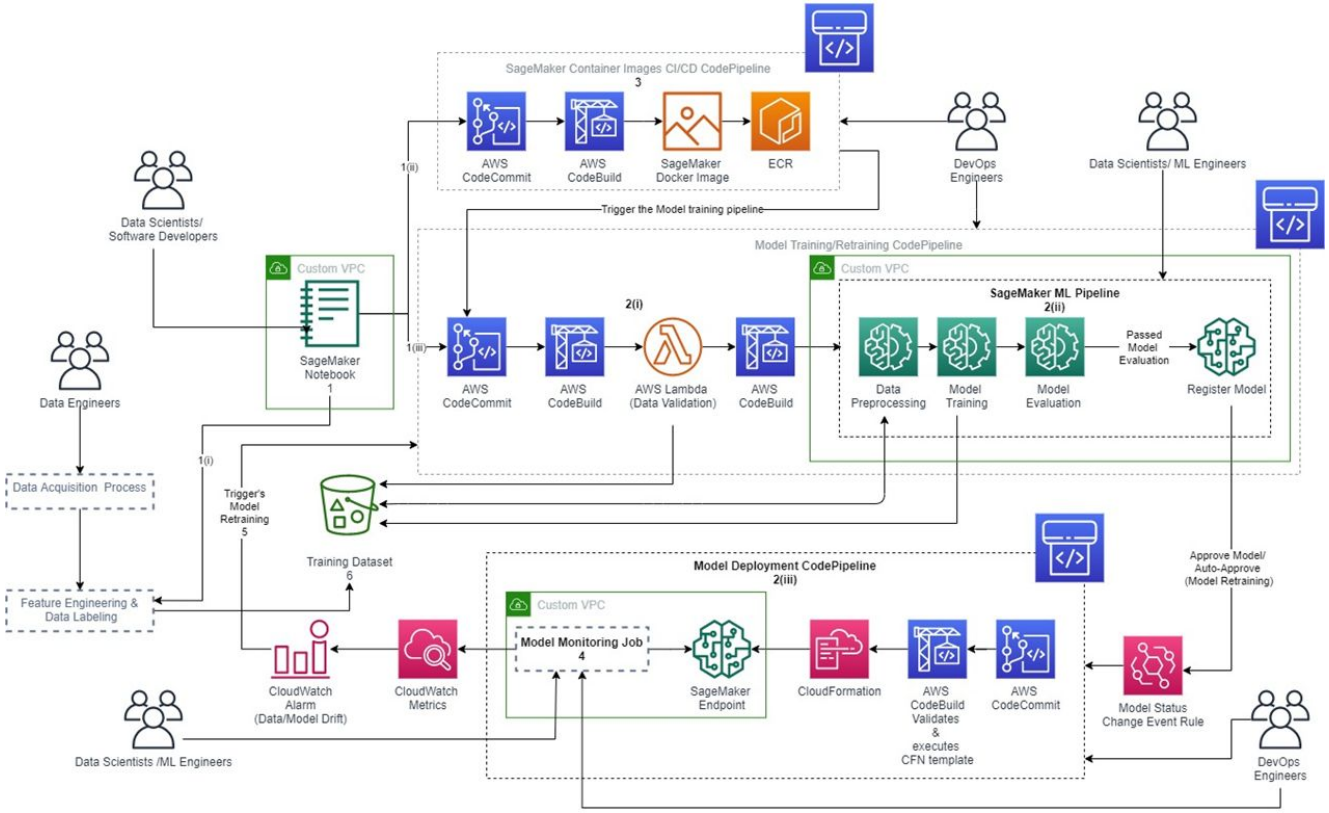
Overview of the solution

Automated model training & inference in Google Cloud



Example application architecture

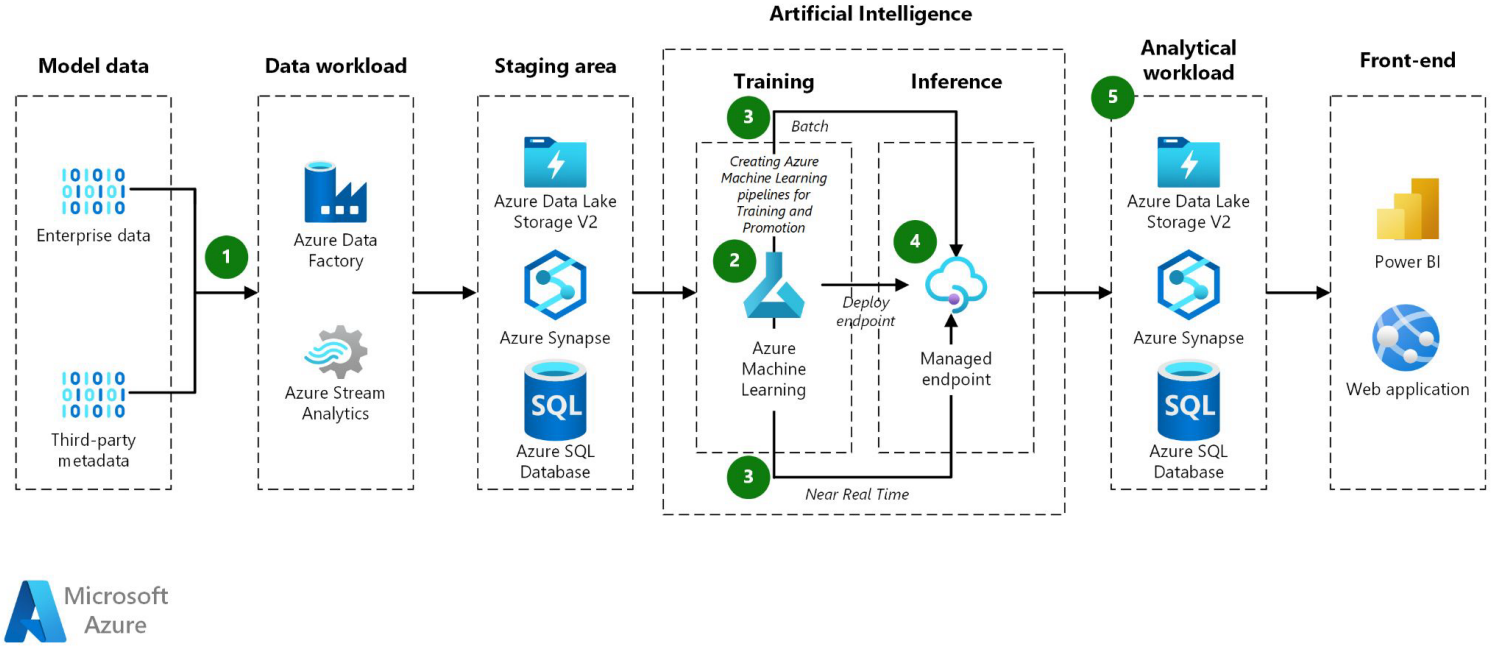
AWS



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

Example application architecture

AWS



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

Virtual environments

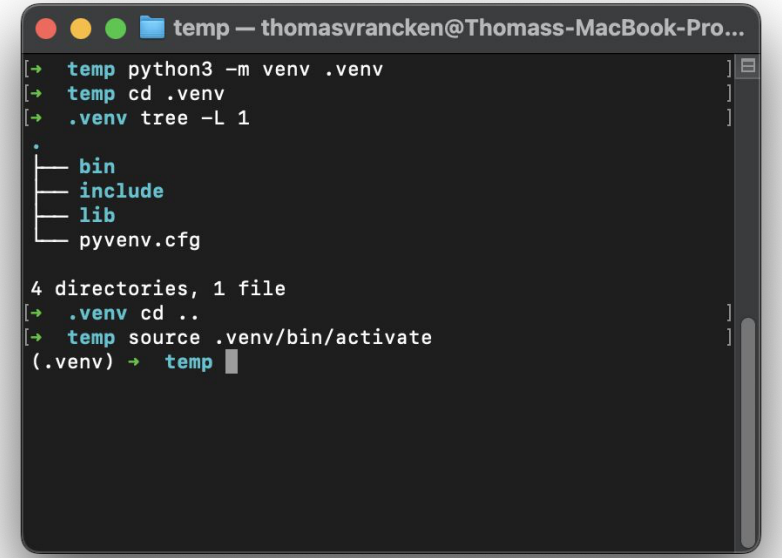
What is a virtual environment?

- Virtual environments keep **dependencies** (e.g. libraries…) in separate “**environments**”
- 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.

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:

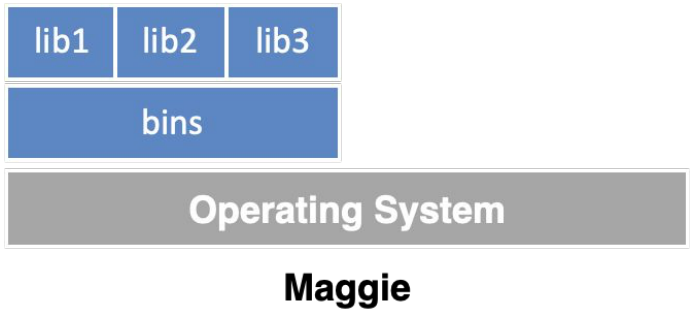
1. **Activation**: When you activate the virtual environment, your shell's PATH is updated to prioritize this bin (or Scripts) directory. This means that when you type python or pip, your shell will use the versions in the virtual environment instead of the system-wide versions.

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?

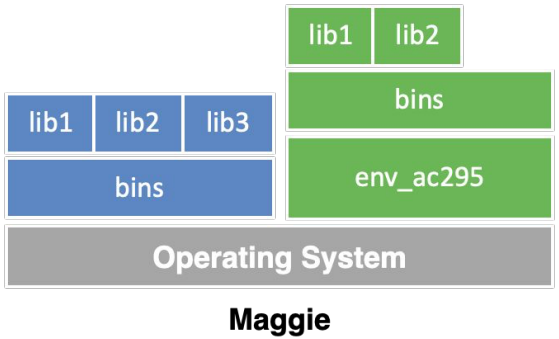
Maggie took “Intro to Machine Learning”. She used to run her Jupyter notebooks from anaconda prompt. Every time she installed a module it was placed in the either of `bin, lib, share, include`folders and she could import it in and used it without any issue.




```
$ 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 virtual environment that helps her keep the modules organized and avoid misbehaviors while developing a new project.

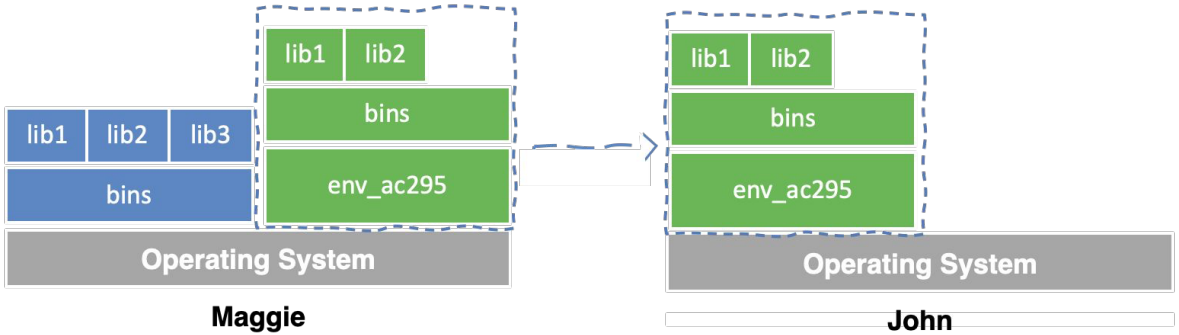


Maggie

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

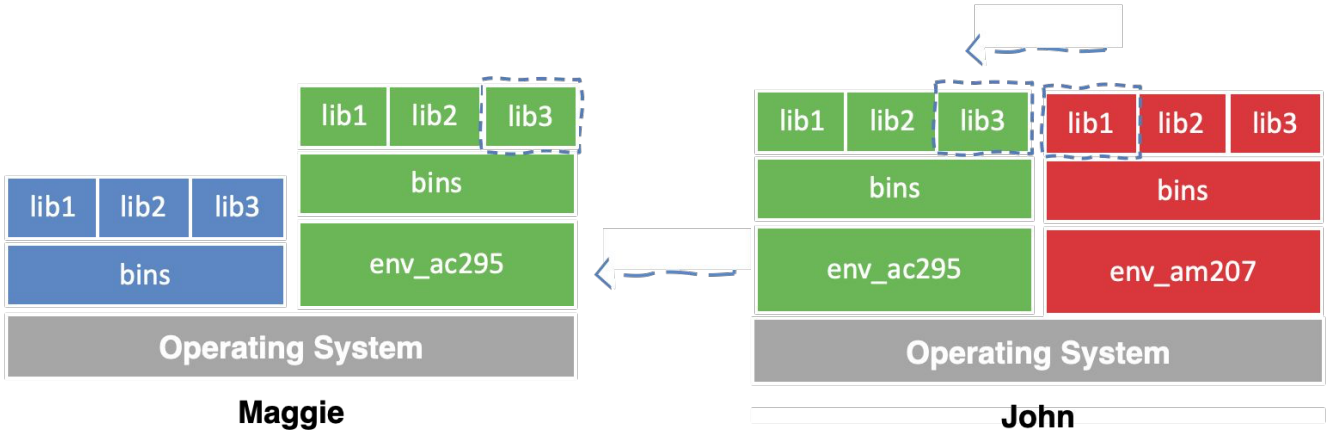
Why should you use virtual environments?

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



Solutions for virtual environments

venv: Utilizes the native Python specification to create isolated directory trees for project dependencies. It serves as the architectural blueprint that ensures local environment integrity. *It is decreasing in popularity…*

pip: The traditional interface for interacting with the Python Package Index (PyPI). It remains the baseline standard for package installation but lacks native support for complex lockfile orchestration. *It is decreasing in popularity…*

uv: A modern, Rust-based engine that replaces the logic of both venv and pip while maintaining compatibility with their standards. It provides a unified interface that accelerates environment instantiation and ensures deterministic dependency resolution. *It is increasing in popularity!* 🚀

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

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
```

uv

uv replaces venv + pip in a single, fast tool.

- 10-100x faster than pip (written in Rust)
- No manual activate/deactivate, uv run handles it
- Lockfile (uv.lock) guarantees exact reproducibility across machines
- Replaces: venv + pip + pip-compile + pip-tools (all in one)
- Compatible with requirements.txt: uv pip install -r requirements.txt still works

```
# Setup (one-time)
  $ uv init my-project              # Creates pyproject.toml + project structure
  $ cd my-project

# Add dependencies
  $ uv add numpy pandas scikit-learn # Adds to pyproject.toml + updates uv.lock

# Run your code
  $ uv run python train.py           # Runs in the managed environment (no activate needed!)

# Sync on another machine
  $ uv sync                          # Recreates the exact environment from uv.lock
```

LIÈGE université

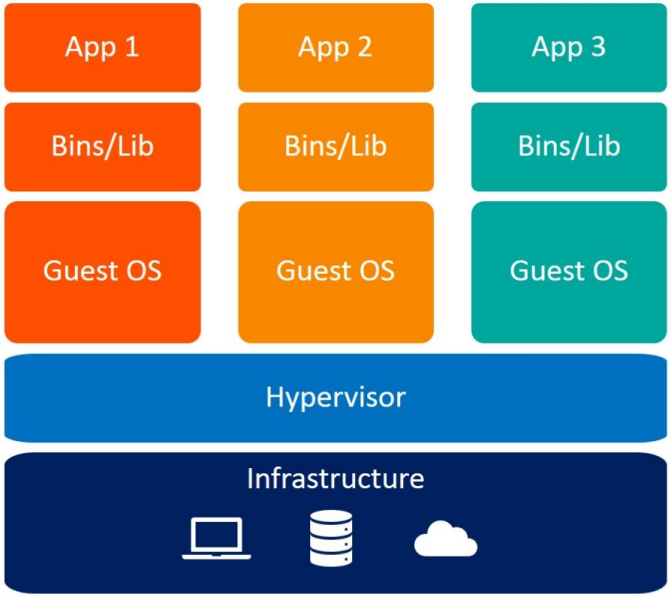
Virtual Machines (VM)

Virtual Machine (VM)

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

Motivation

- Fully control the OS
- Run applications while in development
- Externalised hardware, which might enable:
  - Security
  - Avoid storing sensitive data on local machine
  - More/different compute
  - Shared resources



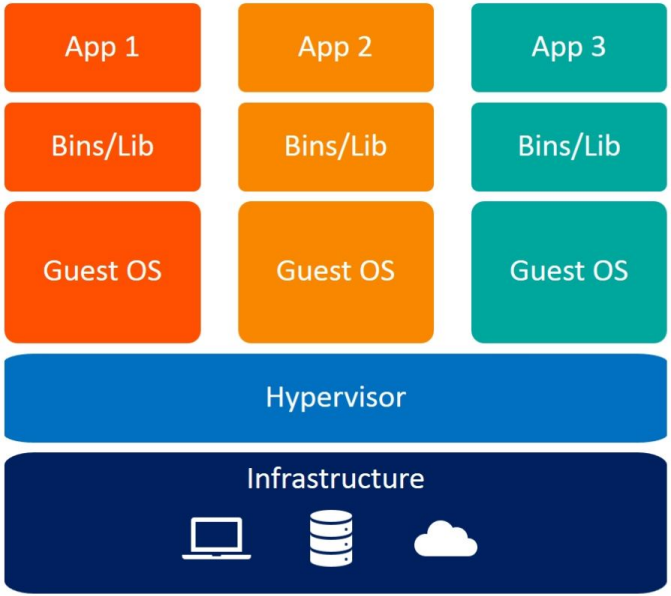
Virtual Machine (VM)

Each VM includes its own:

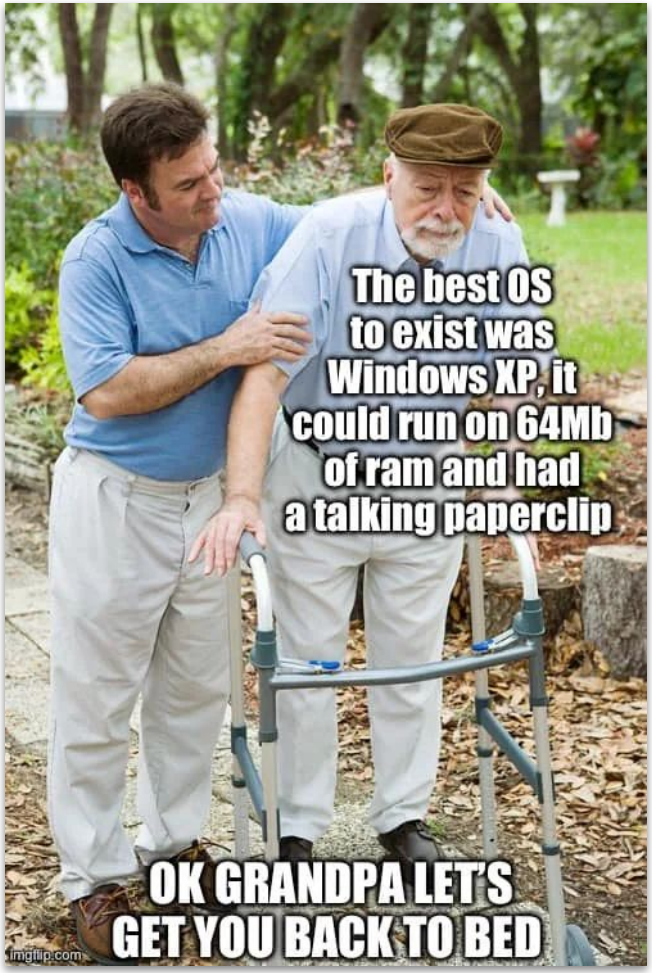
- Operating system kernel
- File system and libraries
- Networking and system tools
- Processes, memory, and compute space

Operating system is called the **host** while those running in a virtual machine are called the **guests**.

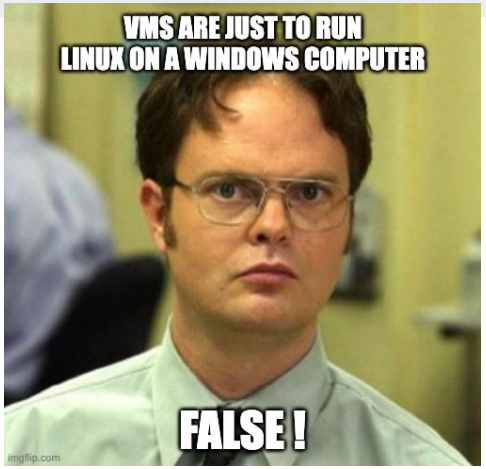
The host machine/server needs a **hypervisor** to manage the different virtual machines running on it.



VMs: let the nostalgic relive some of the glory years



Where can you use VMs?



- VMs can be used locally, to spin different Operating Systems on a single machine
- 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**

AzureVM

Google Compute Engine

Containers

Containers

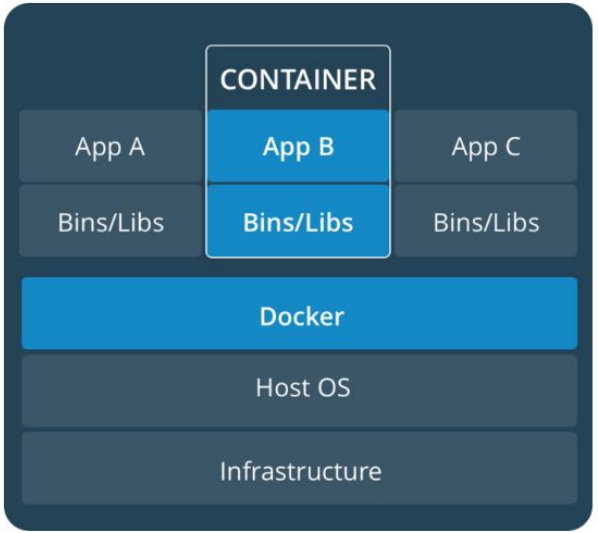
Containers encapsulate an application as a **single executable package** that contains all the information to **run it on any hardware**:

A Docker container will include:

- Application code
- Configuration files
- Libraries
- Dependencies

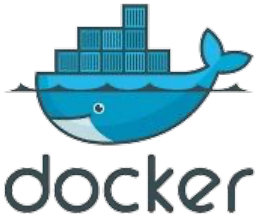
It does not emulate an entire operating system. Rather, a container shares the host OS kernel, which makes containers:

- Much faster to start (milliseconds or seconds)
- Much lighter on system resources (megabytes instead of gigabytes)
- Much easier to deploy at scale



https://mlops4ecm.be/handleidingen/ml-infrastructure/#what-is-a-virtual-machine

Docker



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

Essentially the most popular platform to do so, though alternatives exist:

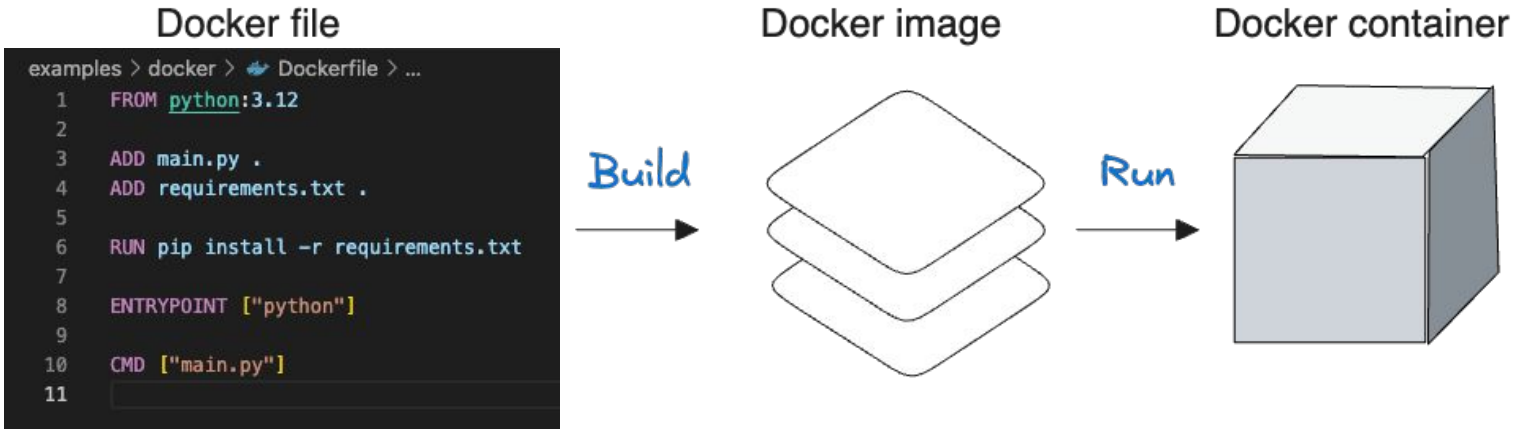
- **Podman**: A daemonless container engine compatible with Docker's CLI, often used in more secure environments (e.g. Red Hat) - doesn't require the background service (daemon) running.
- **LXC** (Linux Containers): A lower-level container runtime focused on system-level isolation. Closer to running lightweight VMs.
- **CRI-O**: A lightweight container runtime used by Kubernetes. It runs simple, light, containers requiring less developer configurations.

What is the difference between an image and container?

Dockerfile: The “script” (text file) used to configure and build the Docker Image.

Docker image: The “class” of your docker instance. It’s its general definition, which can be called with different parameters.

Docker container: An “instance” of your docker image. The actual container running your application.



Looking inside a Dockerfile

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"]
11
```

`FROM`: Base image to use while creating our new Docker image. It can include basic installations (python, pytorch, …) - avoiding having to reinstall them upon each container build.

`ADD`: Add source files to the into the container's base folder.

`RUN`: Commands executed during image build. Install dependencies, copy configs, etc.

`ENTRYPOINT`: Defines the primary executable for the container. Any arguments passed during container instantiation are appended to this command rather than replacing it.

`CMD`: Specifies default arguments or a starting command. Unlike `ENTRYPOINT`, these instructions are auxiliary and are fully replaced if the user provides a command during container invocation.

Popular base docker images

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

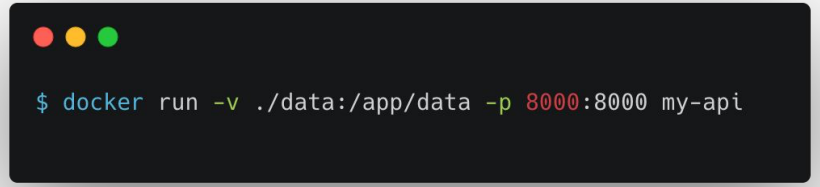
Volumes & Networking

Volumes: Persist and share data

- By default, data inside a container is lost when it stops. A volume mounts a storage location into the container, so data survives restarts.
- Two types (defined in `docker run`):
  - Named volume (managed by Docker): `-v model_store:/app/models`
  - Bind mount (maps a host folder): `-v ./data:/app/data`

Networking: let containers communicate

- Containers on the same Docker network can reach each other by service name.
- Expose ports to the host with “-p host_port:container_port” (e.g. `-p 8000:8000`)
- In Docker Compose, all services share a network automatically.



LIÈGE université

https://docs.docker.com/engine/storage/volumes/

Docker best practices

Build efficient, small and fast Docker images.

- **.dockerignore**: Exclude unnecessary files (e.g. .venv, .git, data/) from the build context. Reduces image size and avoids leaking sensitive files.
- **Layer caching**: Docker caches each instruction. Place rarely-changing layers first (e.g. install dependencies before copying code). Changing a layer invalidates all subsequent layers.
- **Multi-stage builds**: Use a build stage to install/compile, then copy only what's needed into a slim final image. Drastically reduces image size.
- **Minimal base images**: Prefer python:3.12-slim over python:3.12. The slim variant is ~5x smaller (~150MB vs ~1GB).

```dockerfile
# --- Stage 1: Build ---
FROM python:3.12 AS builder
RUN apt-get update && apt-get install -y gcc build-essential
COPY requirements.txt .
RUN pip install --prefix=/install -r requirements.txt

# --- Stage 2: Runtime ---
FROM python:3.12-slim
COPY --from=builder /install /usr/local
COPY . .
CMD ["python", "main.py"]
```


https://docs.docker.com/build/building/best-practices/

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}")

```

LIÈGE université

Multiple containers from same image

Dockerfile 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
```

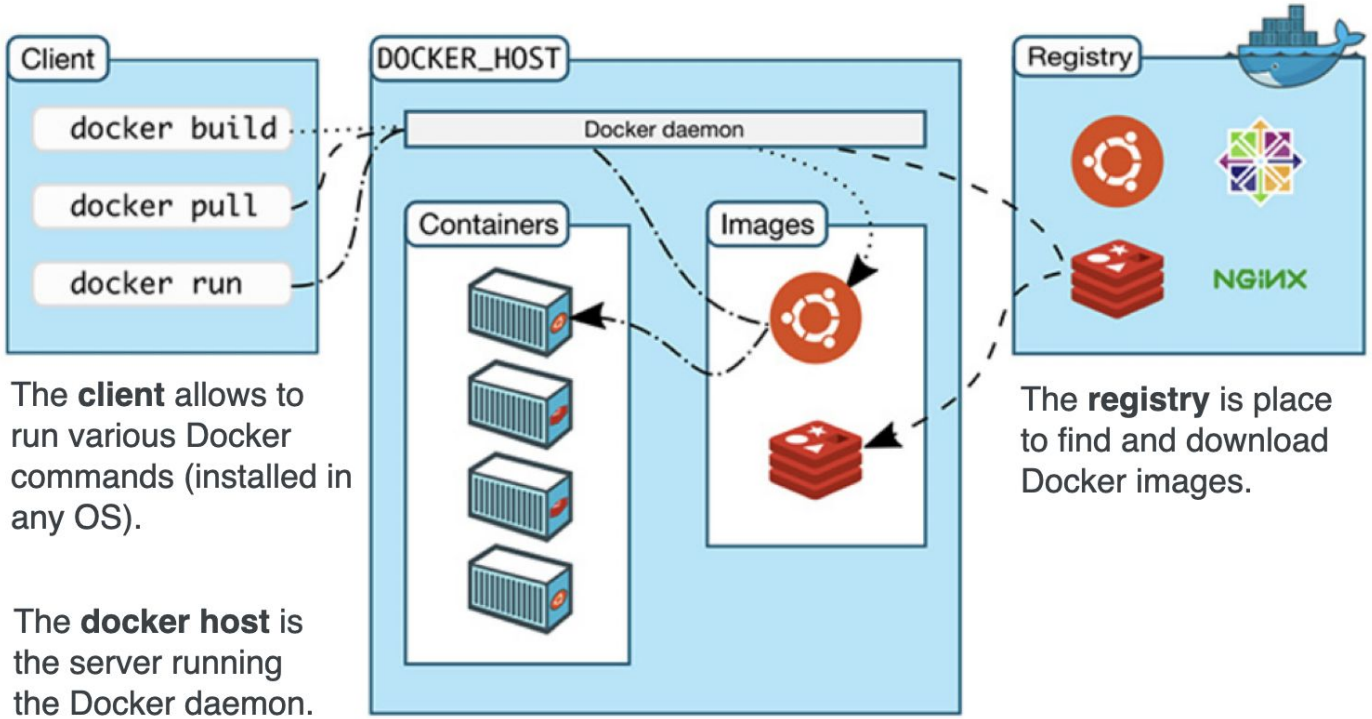
Multiple containers from same image

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
```

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

Docker registry services

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

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

Docker Compose

Run multi-container applications

- Define multiple services, their images, ports, dependencies and volumes in one file
- Services can communicate by name (the api container can reach the database at host "db")
- One command to start everything, no manual coordination
- Volumes to share data across containers. Persists on different container re-starts.

Used mostly locally, not a significant focus in this course!

```yaml
services:
  training:
    build: .
    command: python train.py --epochs 10
    volumes:
      - model_store:/app/models

  api:
    build: .
    command: python serve.py
    ports:
      - "8000:8000"
    volumes:
      - model_store:/app/models
    depends_on:
      - training

  dashboard:
    build: ./dashboard
    ports:
      - "8501:8501"
    depends_on:
      - api

volumes:
  model_store:
```


https://docs.docker.com/compose/

Why containers matter in ML

Pros

- **Reproducibility:** ML models depend on exact versions of libraries like NumPy, PyTorch, and CUDA. A container ensures the entire environment, not just the Python code, is packaged and versioned.
- **Portability:** A container that runs on your laptop will also run on a server, on a colleague's machine, or in a cloud cluster. You don't need to "set up the environment" every time.
- **Deployment:** You can deploy training jobs, APIs, dashboards, or batch scripts as isolated services, without interfering with other tools or environments.
- **Traceability:** If a model was trained using container ml-env:2.1.3, you can later re-run that exact container and trace the results
- **Lightweight**: Only includes high-level software, fast to modify and iterate on. It can be deployed on tiny hardware (on the edge)
- **Isolation**: Runs the model training/serving outside of your local machine, enabling more security, privacy, collaboration and hardware options (e.g. GPUs)

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

… 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

Full overview

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

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
2.1	W03	Prepare your data and run an Exploratory Data Analysis (EDA).
2.2	W03	Train and evaluate your ML model.
2.3	W03	Make sure to document the result of your EDA and model evaluation (e.g. through slides or a markdown file in your "docs/" folder)
2.4	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.
2.5	W03	Store your data to the cloud and modify your training script so that it can automatically run with the data you stored on the cloud. Carefully select the database service you will use on the cloud, and document this choice. Different options might be more appropriate: • Relational SQL database (BigQuery) • NoSQL database (firestore) • Blob storage (cloud storage) If data storage or the choice of database is not trivial for your use case, make sure to discuss it with the teaching staff.

Lecture summary

Topic	Concepts	Relevant for…	
		Project	**Exam**
Exploratory Data Analysis (EDA)	• EDA on structured data, natural language and images • Univariate vs multivariate profiling; Tools: YData Profiling, Pandas, Matplotlib	Yes	Yes
Cloud infrastructure	• Cloud providers (AWS, GCP, Azure) • Service types: IaaS, PaaS, SaaS • Key services: storage, compute, serverless, databases, ML platforms	Yes	
Virtual environments	• Why isolation matters; uv as modern replacement for venv + pip; requirements.txt and lockfiles	Yes	Yes
Virtual machines	• VMs vs containers; Hypervisors; Cloud VMs (EC2, Compute Engine)		Yes
Containers	• Dockerfile, image, container; Docker registry • Key instructions: FROM, RUN, COPY, CMD, ENTRYPOINT;;	Yes	Yes
DW: Docker		Yes	

That’s it for today!

LIÈGE université