Cloud, Big Data, and Docker
Tools for Scalable AI

Andrea Karlova

10/3/2023

Content

Motivation: Building Machine Learning End-to-end Application

- Prototyping machine learning model is an easy task, developing the machine learning system is challenging
- Scaling the implementation and building data driven end-to-end apps brings many problems:
  - the model safeguarding
  - the large volume data pipelines
  - the model is a system component

- End-to-end stack issues:
  Matrix from Hell

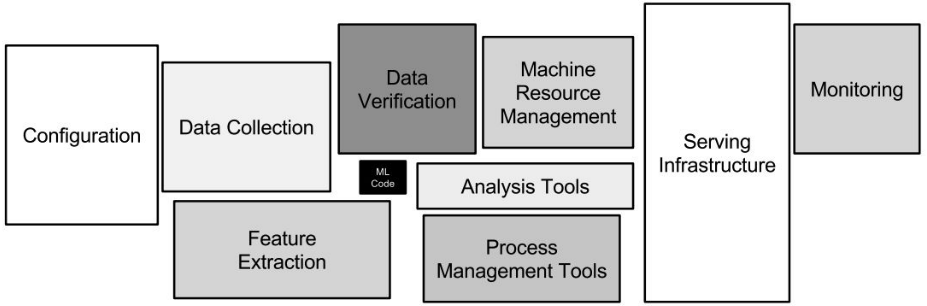


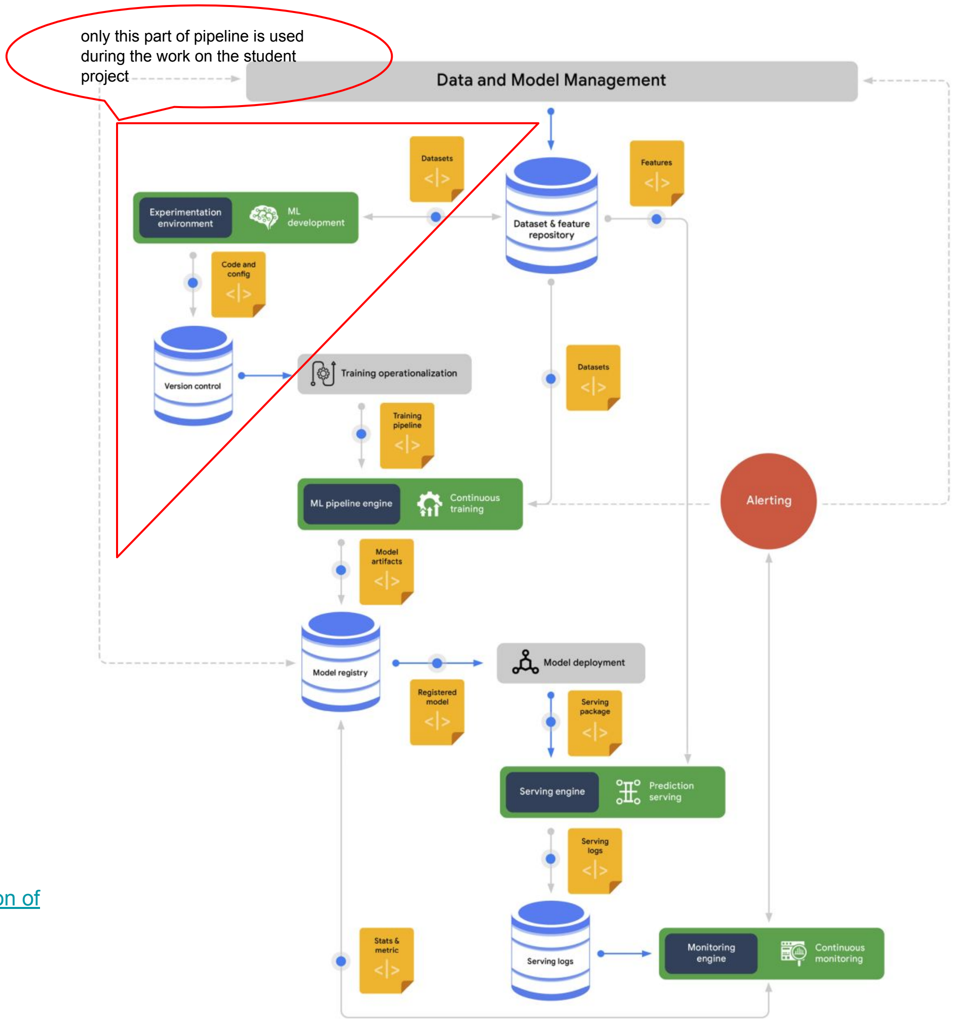
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Hidden Technical Debt in Machine Learning Systems

Motivation: ML System Design

- Components:
  - Data Processing
  - Model Training
  - Model Evaluation
  - Model Serving
  - Experimentation / Online Experimentation
  - Model Monitoring
  - ML Pipelines
  - Model Register

[Practitioners guide to MLOps: A framework for continuous delivery and automation of machine learning.]



only this part of pipeline is used during the work on the student project

Motivation: Engineering Challenges

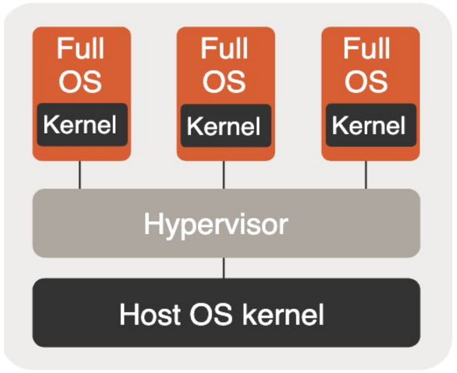
- Each part of the ML System consists of other applications:
  - database or storage units
  - web server
  - messaging system
  - orchestration

- Communicate with other applications via API (Application Programming Interface)
  - different types based on the architecture design:
    1. SOAP API
    2. RPC API
    3. Websocket API
    4. REST API

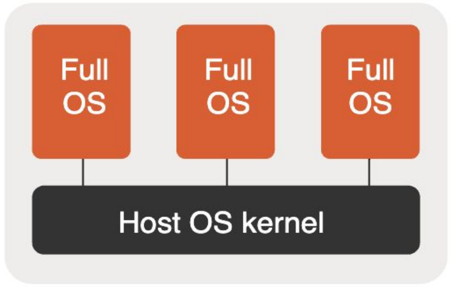
- **Can we isolate the parts of the solutions into separate environments such that the changes in one solution do not impact the entire system?**

Virtual Environments: Containerisation

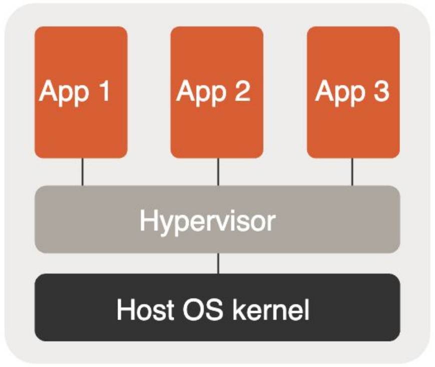
- How does the container differs from virtual machine?
- **Virtual Machine** (VM)
  - VM emulate a physical machine, using the hardware of the host system from a completely separated host OS
- **System Containers**:
  - System containers use the OS kernel of the host system instead of creating their own one
  - provide a full solution of libraries, applications, databases



Virtual machines



System containers



Application containers

Virtual Environments: Application vs System Containers

- Open Source Project: https://linuxcontainers.org
- LXC:
  - set of tools to manage the containers and templates to create a virtual environments for the most common Linux OS
- LXD:
  - image based, provides images for Linux distros

- Application Containers:
  - package a single process or application

Why use containers?

- light-weight → fast
- isolation of the app from hardware architecture
- separates dependencies between different types of the end-to-end apps

Virtual Environments: Docker

- Docker is a high-level tool utilising LXD containers
- uses namespaces to create separation of the environments
- **designed for microservice based applications**
- the image can be booted instantly
- the container becomes the unit for distributing and testing the application:
  a. the developer writes the code locally and share their work using Docker container
  b. use Docker platform to push the app into the testing environment and execute automated and manual tests
  c. when the developer finds bugs, they can fix them in the dev environment and re-deploy the container to the test environment
  d. once the testing is completed, the container can be pushed to the production environment

Virtual Environments: Docker Architecture

- **Docker Daemon:**
  - listens for Docker API requests
  - manages Docker objects such as images
- **Docker Client:**
  - user uses command
  - the client sends these commands to the Docker Daemon which executes the commands
  - the Docker command uses the Docker API
- Docker Registries:
  - stores Docker images
- Docker Objects:
  - **Image:**
    - read-only template with instructions for creating a Docker container
    - to build own image, we need to write the Dockerfile
    - each instruction in the Dockerfile creates a layer in the image
  - **Containers:**
    - container is a runable instance of the image

Virtual Environments: Docker CLI

Docker Commands:

docker build <name_of_the_container> : build an image from a Dockerfile

docker pull <name_of_the_image>: download image

dicker run <name_of_the_container>: run a container

docker ps: list containers

docker ps -a: list all containers

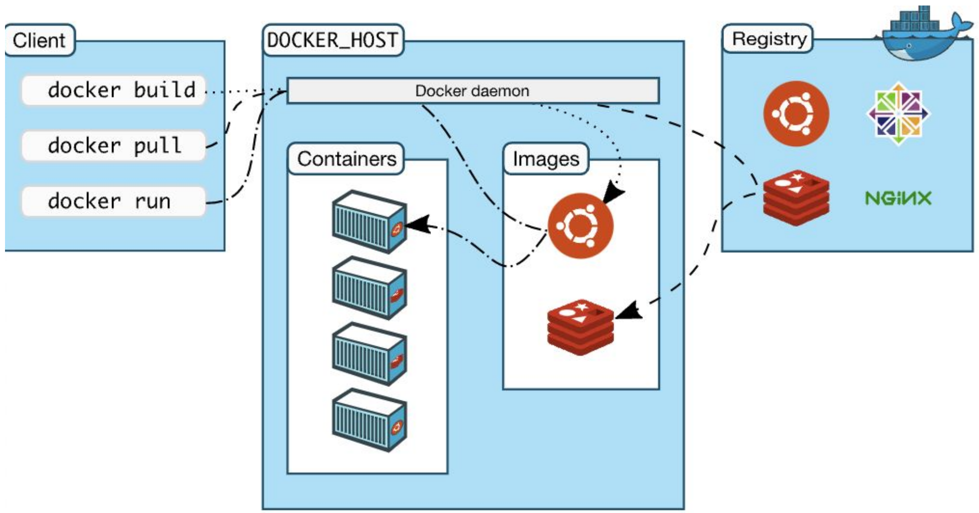
docker stop <name_of_the_container>: stop a container

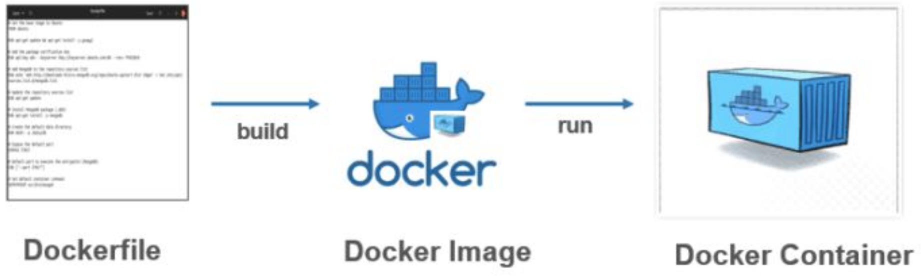
docker rm <name_of_the_container>: remove a container

docker images: list images

docker rmi <name_of_the_images>: remove image

docker exec: execute a command on the docker container





Virtual Environments: Writing Dockerfile

- Dockerfile:
  - allows creating the containers in a simple and maintainable way
  - each instruction in the Dockerfile creates a layer versioning the changes made to the container image
- Selected commands:

FROM <image>: setting up the parent image

RUN <command>: run executes during building of the container

CMD <command> <param1><param2>: command which is executed
when the container is started

ENV <key> <value>: sets environmental variable

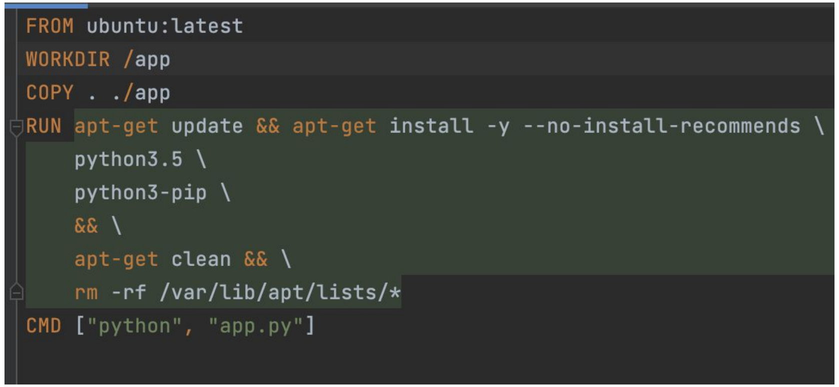
ADD <src> <dest>: copies new files / dirs /remote URLs and add these
to the filesystem of the image

COPY <src> <dest>: transfer the data from outside into container

ENTRYPOINT <command> <param1> <param2>: allows you to
configure a container that will run as an executable

WORKDIR </path/to/workdir>: set the working directory in the container

ARG <name>[=<default value>]: Defines a variable that users can pass at build-time to the builder with the `docker build` command using the `--build-arg <varname>=<value>` flag



```
FROM ubuntu:latest
WORKDIR /app
COPY . ./app
RUN apt-get update && apt-get install -y --no-install-recommends \
    python3.5 \
    python3-pip \
    && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
CMD ["python", "app.py"]
```

Virtual Environments: Layering of the Docker Image

- ımage consists of multiple layers
- the layers are combined into single virtual filesystem of the image → initial filesystem state
- layers are are immutable
- each step in Dockerfile creates a new 'layer' that is ‘a diff’ of the filesystem changes since the last step
- we can start the container from a particular layer
- layers allow build steps to be cached to avoid redundant work
- docker pull: layers enable partial image pulls

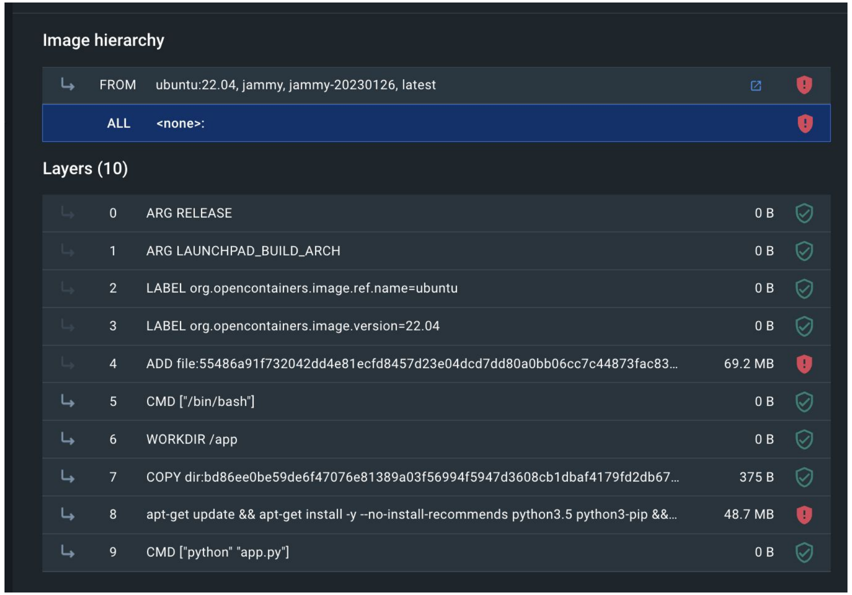


Image hierarchy

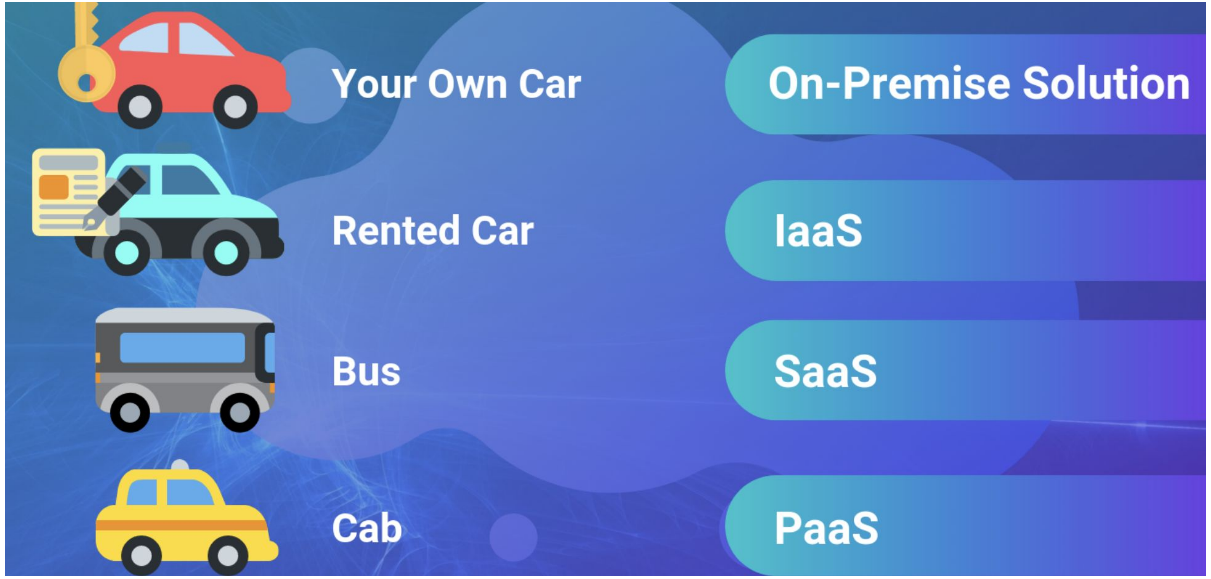
FROM	ubuntu:22.04, jammy, jammy-20230126, latest	
ALL	<none>:	

Layers (10)

0	ARG RELEASE	0 B
1	ARG LAUNCHPAD_BUILD_ARCH	0 B
2	LABEL org.opencontainers.image.ref.name=ubuntu	0 B
3	LABEL org.opencontainers.image.version=22.04	0 B
4	ADD file:55486a91f732042dd4e81ecfd8457d23e04dcd7dd80a0bb06cc7c44873fac83...	69.2 MB
5	CMD ["/bin/bash"]	0 B
6	WORKDIR /app	0 B
7	COPY dir:bd86ee0be59de6f47076e81389a03f56994f5947d3608cb1dbaf4179fd2db67...	375 B
8	apt-get update && apt-get install -y --no-install-recommends python3.5 python3-pip &&...	48.7 MB
9	CMD ["python" "app.py"]	0 B

Cloud Computing:

- cloud computing is on-demand availability of the computational resources over the internet
- uses network to connect users to the cloud platform
- users can request and access the rented computing services
- it is a service based model



Cloud Computing:

- Cloud computing is enabled by virtualisation
- OS-level virtualisation creates a scalable system of multiple independent computing devices
- Commercial cloud service providers: AWS, Google Cloud, Azure, IBM Cloud
- Open-source cloud: OpenStack, AppScale (Eucalyptus)
- **What is the difference between cloud computing and cluster?**
  - Cluster:
    - computing is a high performance computing framework where the computation task is being shared between multiple computers on the cluster enabling parallel computing
    - cluster system and resources are tightly coupled whereas cloud allows for dynamic computing infrastructure and resources allocation
  - Cluster on Cloud:
    - Kubernetes cluster: group of nodes running containerized applications using an open-source system for deploying, scaling and management of containerized applications

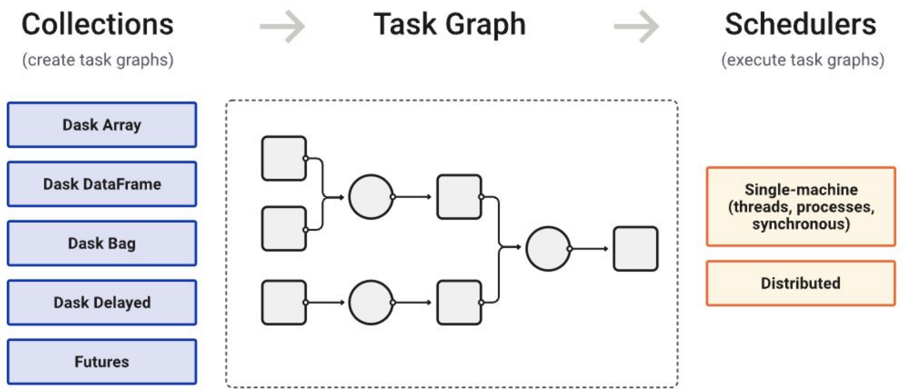
Big Data

- How to process the datasets which are too large to fit into RAM?
  - Can I reduce the size? Do I really need all data points or can I subsample or filter part of the data?
  - What is the computational bottleneck?
    - Do I need to train or predict on the out-of-memory datasets?
    - Do I need to train one model on the out-of-memory dataset?
    - Do I need to train many models on small datasets?
- Parallel programming
- Algorithmic design can improve the efficiency:
  - lazy evaluation
  - memory mapping

Big Data: Dask

- Dask: lightweight library for parallel computing in Python with interface mimicking the Pandas DataFrame, Numpy Array and PySpark
- Components:
  - dynamic task scheduling optimised for interactive computational workload
  - ‘Big Data’ collections such as parallel arrays, dataframes, lists, iterators extended to larger-than-memory or distributed environment

1. the data can be stored and read from remote locations and in various formats in parallel and lazily
2. it reads only the sample of the datasets into the memory and creates the task graph



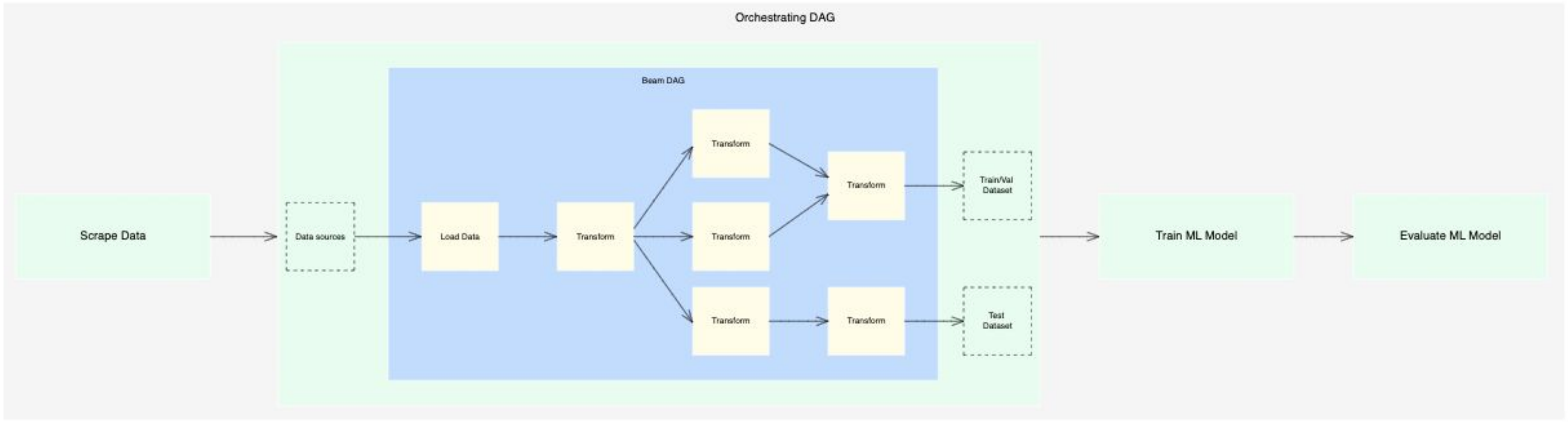
High level collections are used to generate task graphs which can be executed by schedulers on a single machine or a cluster.

Big Data: PySpark

- Python wrapper around Spark which is a distributed computing tool for tabular datasets, a part of fairly matured ecosystem
- able to deal with datasets larger than **1TB**
- integrated SQL engine, so the dataset can be manipulated using SQL
- the API does not resembles the Pandas and Numpy library, the output is semi-structured
- Comparison to Dask:
  - Dask is more lightweight
  - Dask relies on and interoperates with existing libraries like Scikit-learn and XGBoost, results in a less-cohesive whole
  - Dask has a real-time streaming functionality that is lower-level than Spark streaming which allows for more creative and complex use-cases

Big Data: Apache Beam

- unified programming model for defining both batch- and streaming-data parallel-processing pipelines
- offers a high level way to represent the flow of data:
  - pipeline is a user-constructed graph of transformations
  - data processing transformations: PCollection, PTransform, Aggregation, User-defined functions
- the runners can be specified: e.g. local compute power (Direct Runner), Spark or Dask (Distributed Runner)

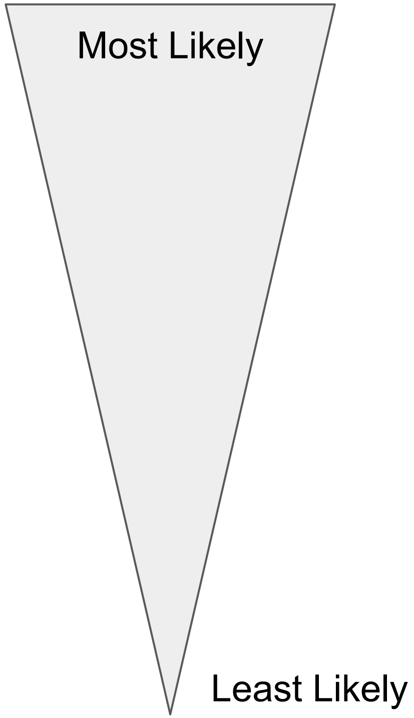


Summary: Which Tools to Use for the Project?

- Need to process datasets 5 - 10 smaller than available RAM? Use Pandas
- Need to process datasets smaller than 500GB? Use Dask, otherwise use Spark
- Need to control the preprocessing data pipeline? Use Apache Beam (or Tensorflow Extended)
    - Where is your input data stored? How many of input datasets do you have?
    - How does the input data look like?
    - What do you want to do with your data?
    - How does the output data looks like and where it should go?
- Need to perform hyper-parameters tuning or run multiple parallel experiments? Package the model into Docker container and run multiple containers in parallel with different input parameters.

How to get your project experiments running

- In you personal computer/laptop
- In department infrastructure
- In the Cloud (e.g. Google Cloud, AWS, Azure etc.)
- In your supervisor’s infrastructure
- High Performance Grid



Running Experiments: Department Infrastructure

- Available free-of-charge/ But can get occupied really quickly
- **3 Labs in MPEB (Open 8:00 am to 7:00 pm weekdays only)**
  - **Lab 121**
    - 30 Desktops
    - 16GB RAM
    - Nvidia GeForce GTX 650 Graphic Cards
  - **Lab 105**
    - 18 Desktops
    - Intel i9 processors
    - 128GB RAM
    - Nvidia GeForce RTX 3090 Graphic Cards
  - Lab 406: Monitors with Internet (BYOD)
- Details about labs:
  - https://tsg.cs.ucl.ac.uk/labs-2/
  - Any further information: https://tsg.cs.ucl.ac.uk/contact-us/ (Can walk in during office hours **9.30am – 5pm** term time)

Homework

- Install Docker platform on your computer: https://docs.docker.com/get-docker/
- Follow the Docker tutorial available via Docker Labs and build simple ToDo list manager running in Node.js. The Docker tutorial repo is available at https://github.com/docker/getting-started
  - Play around the Docker Platform:
    - explore the Images tab: explore Image hieararchy and Layers
    - read in detail about commands used in the Image Layers: https://docs.docker.com/engine/reference/builder/
  - Follow the tutorial and download the app.zip
  - Write the Dockerfile as instructed and build the image
  - Run the container and experiment with making changes to the app in src/static/js/app.js and see the changes within image.
  - Read the manual for docker ps, a command which helps to monitor the processes related to container, e.g CPU time, processses running for spefic user group, etc.

Homework: cont.

- Follow instructions to run ubuntu container with the bash command to generate random number and store it to the file inside the container:
  ```
  docker run -d ubuntu bash -c "shuf -i 1-10000 -n 1 -o /data.txt && tail -f /dev/null"
  ```
  Inspect the output stored within the container
  ```
  docker exec $(docker ps | grep 'ubuntu' | awk '{print $1}') cat /data.txt
  ```
- Learn about the Container Volumes which allows to connect the container with specific file-system back to the host machine. Work with the bind mounts to learn to control the location on your machine where the data are stored.
- Remember that each container should do just one think which it can do well. Learn to link two containers by using container networking, one with SQL database, second with the ToDo app.
- Learn how to use docker compose tool which allows you to set up and share multi-container applications
- Conclude by the setting up the multistage build and explore the layer caching.

You are ready to learn how to build the experimentation platform for you machine learning project.