Machine Learning in Production

Dr. Bhaskar Chakraborty

About me

- Background
  - Machine Learning and Artificial Intelligence research, S/W engineering and Architecture
  - PhD in Computer Vision
- Now
  - AI Architect/ML Engineer @UniCredit AI team
  - Building end-to-end AI applications for banking workflows

LinkedIn, Github

Content

- Introduction to *ML in production*
- Catch points → *ML in production*
- ML Serving
- ML model drift
- MLOps - the backbone of ML applications in production
- ML app lifecycle
- Compliance
- Demo
- Q&A and Discussion

Quick recap: what is ML?

Quick recap: what is ML?

$$\overrightarrow{\boldsymbol{X}} \;\square\; \overrightarrow{\mathbf{y}}$$

Symbolic representation of **Input data vector transforming** into a **meaningful output data vector**

Quick recap: what is ML?

$$\vec{X} \ \square \ \vec{y}$$

🪄

ML techniques

Quick recap: what is ML?

$$\overrightarrow{\boldsymbol{X}} \;\square\; \overrightarrow{\mathbf{y}}$$

ML techniques

ML teaches us different tools and techniques to transform an input vector $\overrightarrow{X}$ to a meaningful output vector $\overrightarrow{\mathbf{y}}$

Examples of (X, y)

- Input: Transactional details (amount, location, time and user history)
- Output: Fraud / Not fraud
- Use-case: Fraud detection in Banking

Examples of (X, y)

- Input: Email text, sender info, metadata
- Output: Spam / Not Spam
- Use-case: Spam email filtering

Examples of (X, y)

- Input: Camera images, Censor data (LiDAR), GPS signal
- Output: Steering angle, acceleration and braking command
- Use-case: Autonomous driving

Examples of (X, y)

- Input: Patient data (symptoms, lap results, scanned images)
- Output: Disease prediction, diagnosis report
- Use-case: Medical diagnosis

ML in production: ML phase

Business use-case

ML in production: ML phase

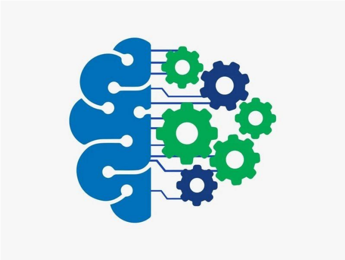
Business use-case

Data (X)

ML in production: ML phase

Business use-case

Data (X)

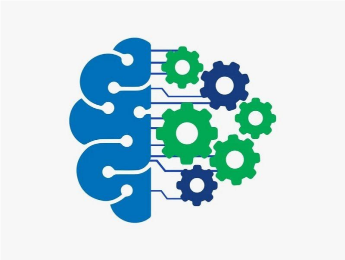


ML Model ($X \rightarrow \mathbf{y}$)

ML in production: ML phase

Business use-case

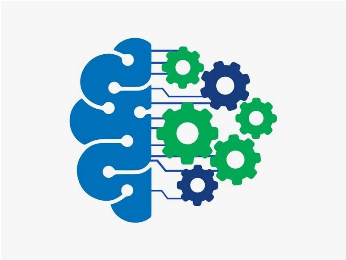
Data (X)



ML Model ($X \rightarrow \mathbf{y}$)

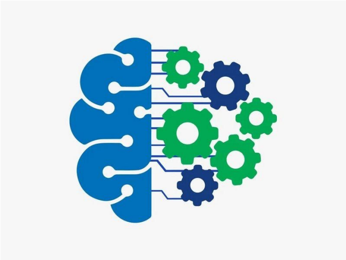
ML

ML in production: Production phase



ML Model ($\boldsymbol{X} \longrightarrow \mathbf{y}$)

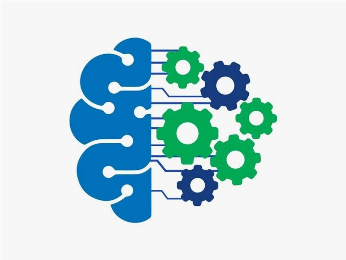
ML in production: Production phase



ML Model ($\boldsymbol{X} \longrightarrow \mathbf{y}$)

Deployment

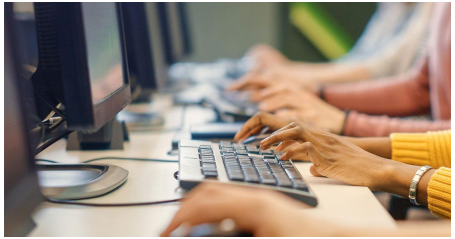
ML in production: Production phase



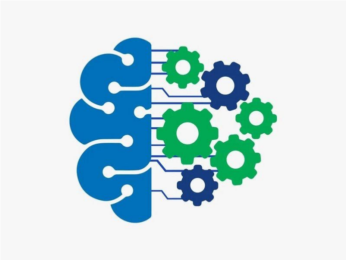
ML Model ($X \rightarrow \mathbf{y}$)

Deployment

X



ML in production: Production phase

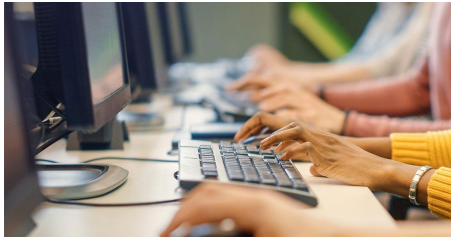


ML Model ($X \rightarrow \mathbf{y}$)

Deployment

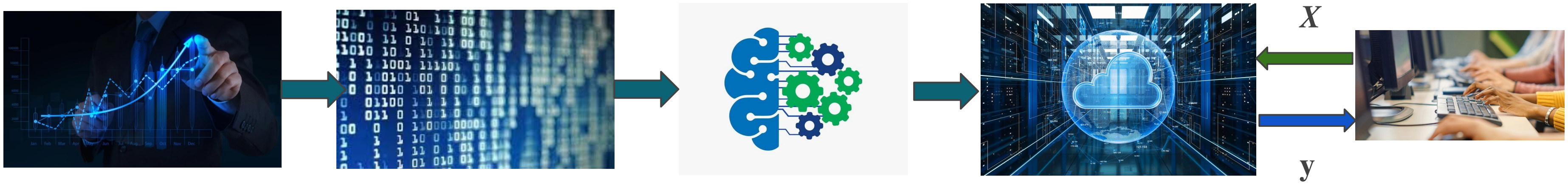
X

$\mathbf{y}$



Production

ML in production



Business use-case

Data (X)

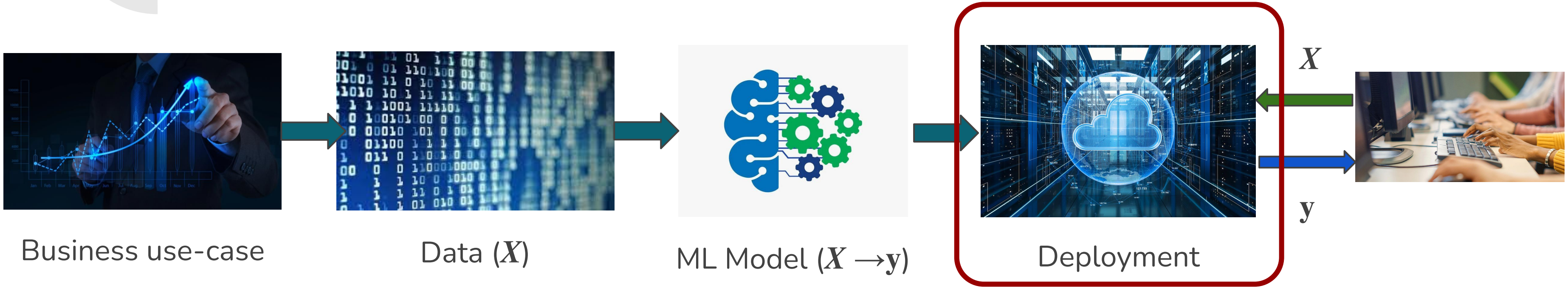
ML Model ($X \rightarrow \mathbf{y}$)

Deployment

ML

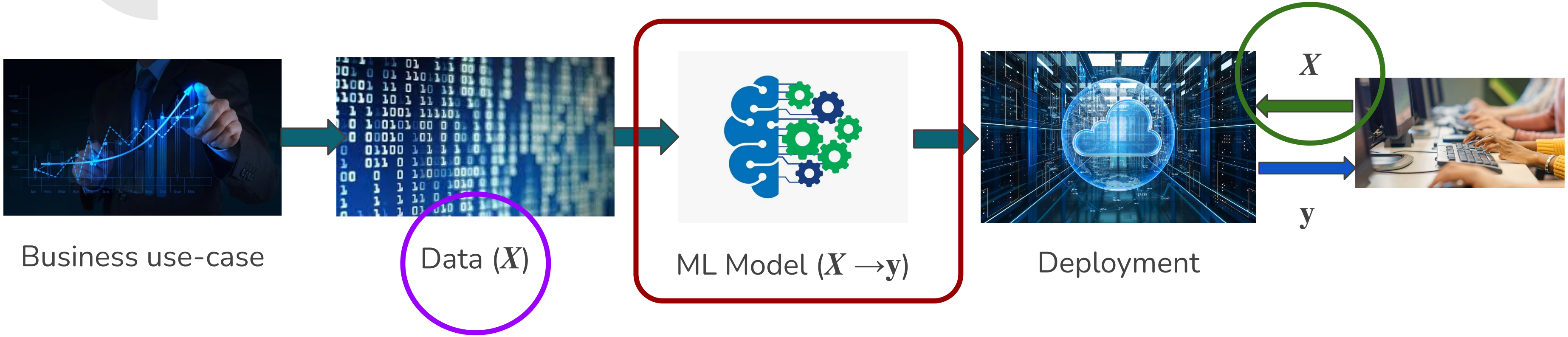
Production

Catch point: Deployment



- ➔ What is deployment and how does it work?
- ➔ How to ensure a reliable deployment?
- ➔ How to make the ML model available to the users?

Catch point: Model drift



- ➔ The data distribution in Production often varies from the data distribution used for the initial model training → Model drift
- ➔ How to identify the model drift?
- ➔ How to properly address this model drift?

ML Model to the user

- What is inside the ML model?

ML Model to the user

- What is inside the ML model?
  - A set of real numbers representing parameters and hyper-parameters of several hyperplanes
  - Usually this parameter set is very large

ML Model to the user

- What is inside the ML model?
  - A set of real numbers representing parameters and hyper-parameters of several hyperplanes
  - Usually this parameter set is very large
- We also need a set of libraries that can read those parameters and can give the prediction

ML Model to the user

- What is inside the ML model?
    - A set of real numbers representing parameters and hyper-parameters of several hyperplanes
    - Usually this parameter set is very large
- We also need a set of libraries that can read those parameters and can give the prediction

ML Model to the user

- To make available the ML model we need

An interface that two computer systems use to exchange information securely over the Internet

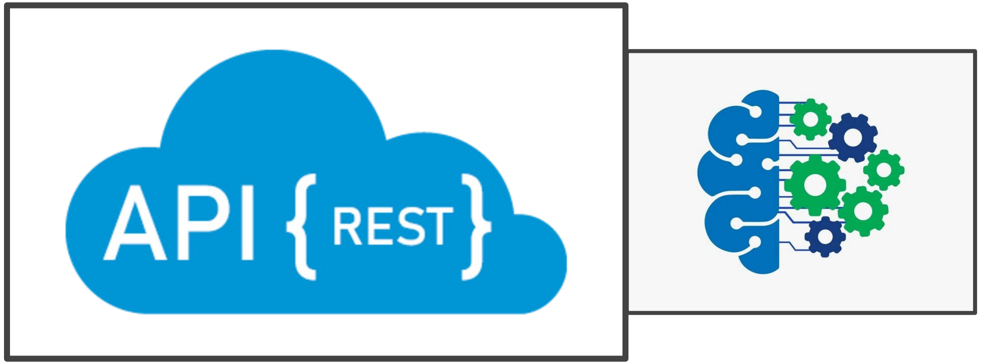
ML Model to the user

- To make available the ML model we need

An interface that two computer systems use to exchange information securely over the Internet

If such an interface exists, we can put our ML model in one computer and all the users will communicate to that computer to send the input data X to get the prediction y

ML Model to the user



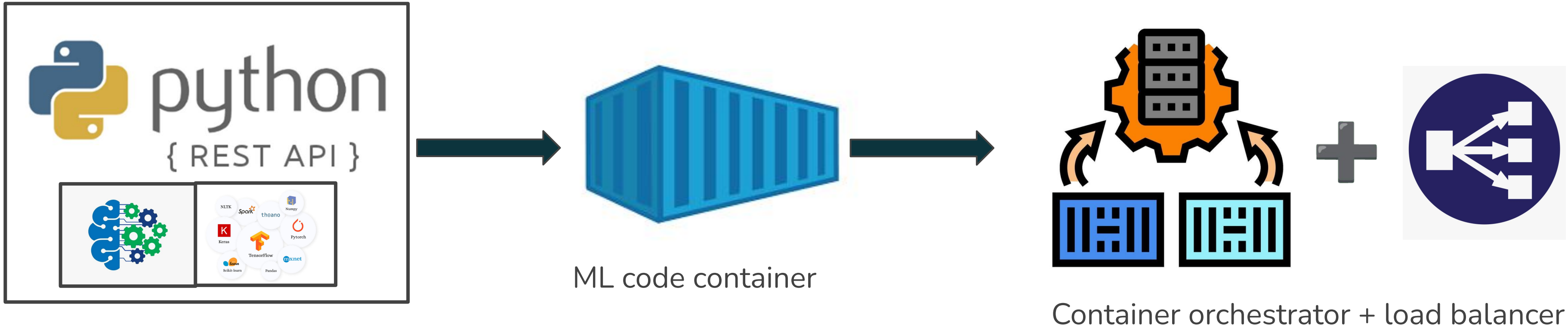
Thanks to the RESTful API, we are wrap the Ml model to make available to the users
ML serving

Code → Container → Container Orchestrator

ML code container

We *containerize* the code –> **a lightweight, standalone, executable package of software that bundles an application's code with all its dependencies, like libraries and configuration files.**

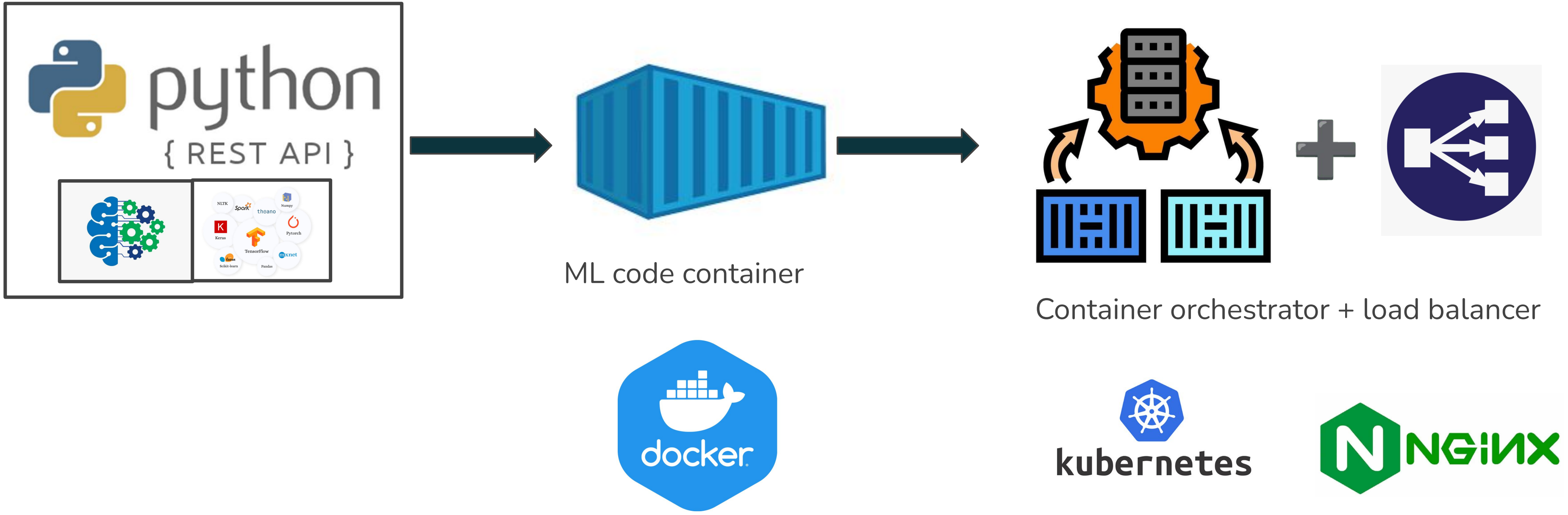
Code → Container → Container Orchestrator



We *deploy* the ML container inside a *container orchestrator tool* together with a *load balancer* –> **a system that automates the deployment, management, scaling, and networking of containerized applications**

Reliability

Code → Container → Container Orchestrator



Deployment

Deployment refers to the **process** of making a software application or update **available to end-users by installing and configuring** it on a **target environment** like servers, desktops, or mobile devices.

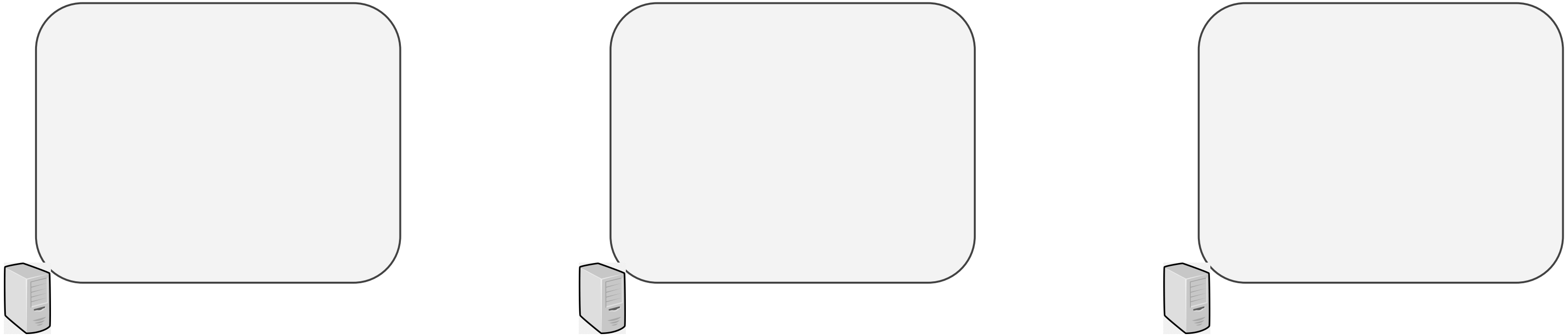
The goal is to **efficiently and reliably** make the software **ready for use**, which can involve a **combination of manual and automated processes** to handle **installation, configuration, and updates**.

Deployment ML serving

Let’s say that the ML model requires 6 CPU cores and 8 GB RAM and we have total 3 users

Deployment ML Serving

Let’s say that the ML model requires 6 CPU cores and 8 GB RAM and we have total 3 users



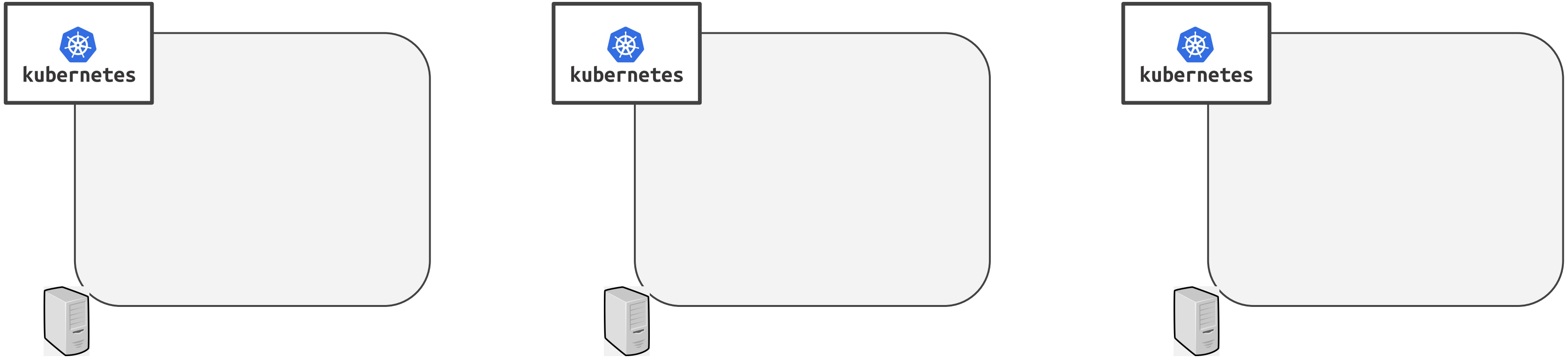
Node-1
(6 CPU, 8 GB RAM)

Node-2
(6 CPU, 8 GB RAM)

Node-3
(6 CPU, 8 GB RAM)

Deployment ML Serving

Let’s say that the ML model requires 6 CPU cores and 8 GB RAM and we have total 3 users



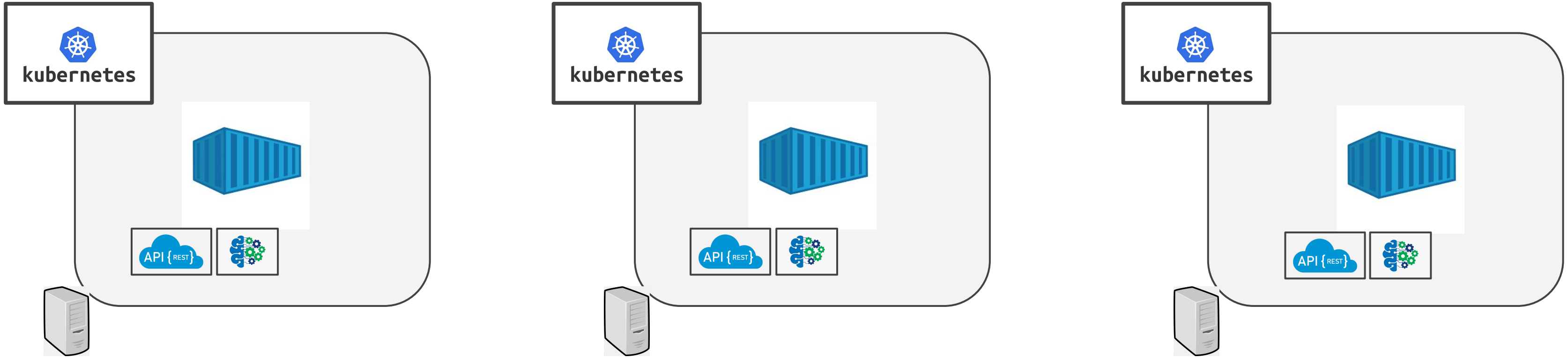
Node-1
(6 CPU, 8 GB RAM)

Node-2
(6 CPU, 8 GB RAM)

Node-3
(6 CPU, 8 GB RAM)

Deployment

Let’s say that the ML model requires 6 CPU cores and 8 GB RAM and we have total 3 users

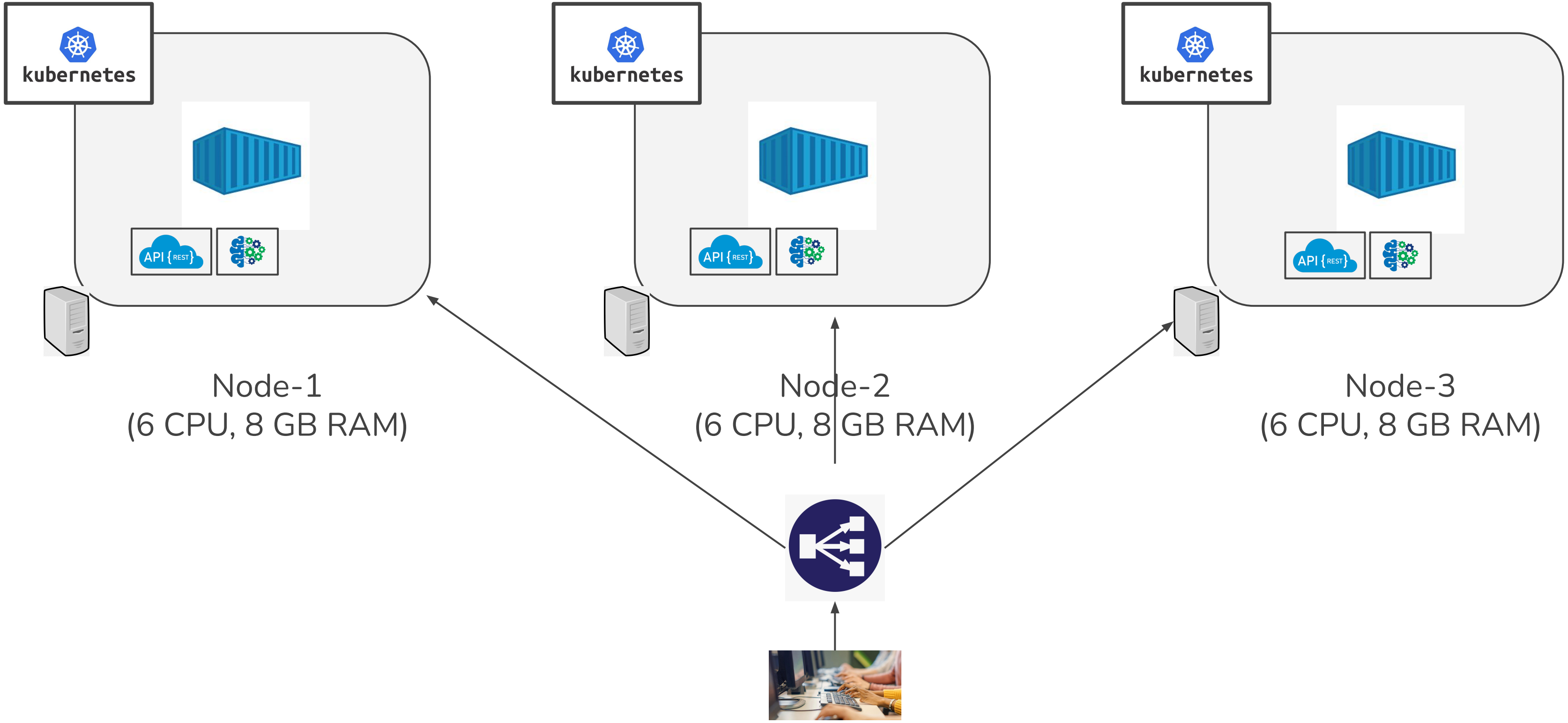


Node-1
(6 CPU, 8 GB RAM)

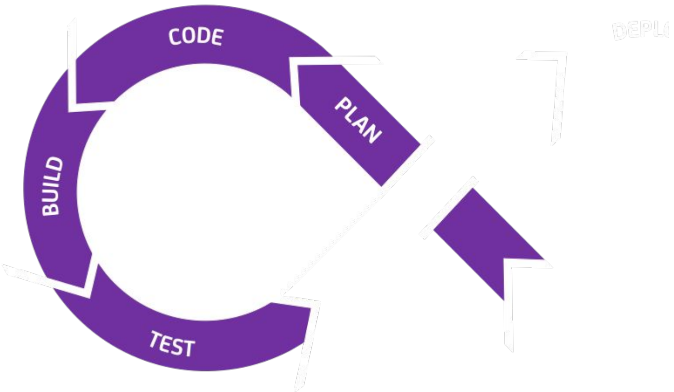
Node-2
(6 CPU, 8 GB RAM)

Node-3
(6 CPU, 8 GB RAM)

Deployment ML Serving

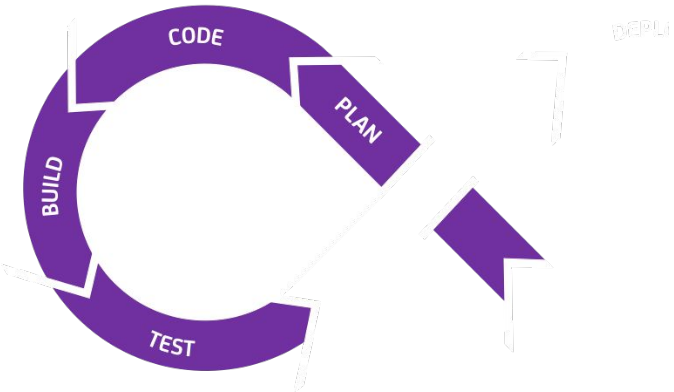


Translate to DevOps concepts



- ➢ Planning for the development of work (scope definition)
- ➢ Checkout the code version from a code hosting/collaboration tool (git)
    - Code quality checking
    - Running all the tests and check code coverage
- ➢ Container building
- ➢ Container scanning for security vulnerabilities
- ➢ Container image saving with version tracking

Translate to DevOps concepts

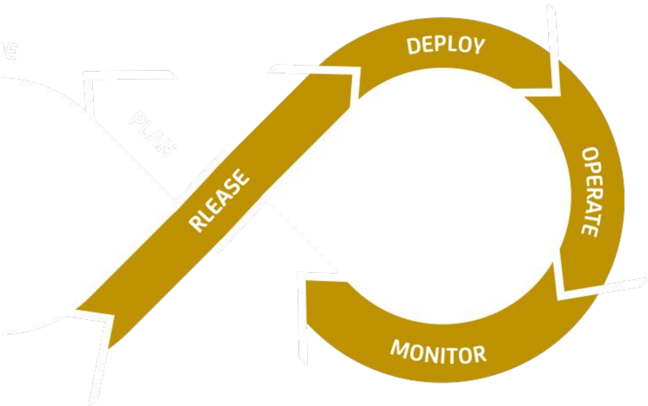


- ➢ Planning for the development of work (scope definition)
- ➢ Checkout the code version from a code hosting/collaboration tool (git)
  - ○ Code quality checking
  - ○ Running all the tests and check code coverage
- ➢ Container building
- ➢ Container scanning for security vulnerabilities
- ➢ Container image saving with version tracking

Continuous Integration (CI)

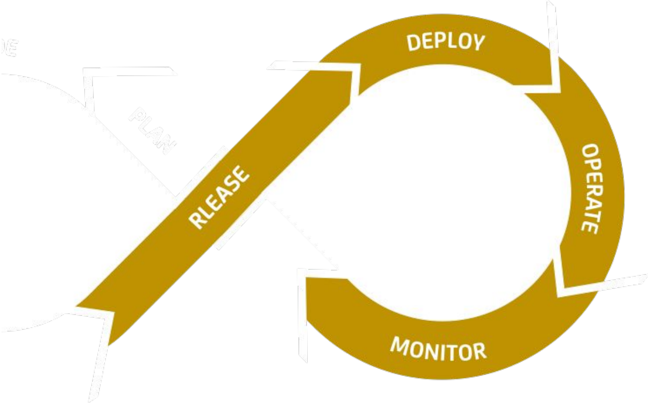
Translate to DevOps concepts

- ➢ Release
  - Save the image container in *container registry* → publish image
- ➢ Deploy
  - Combines published container image together with a set of user defined config parameters
  - Creates deployment unit
  - Actual deployment
  - Rollback on failure
- ➢ Production run
- ➢ Monitor the deployment



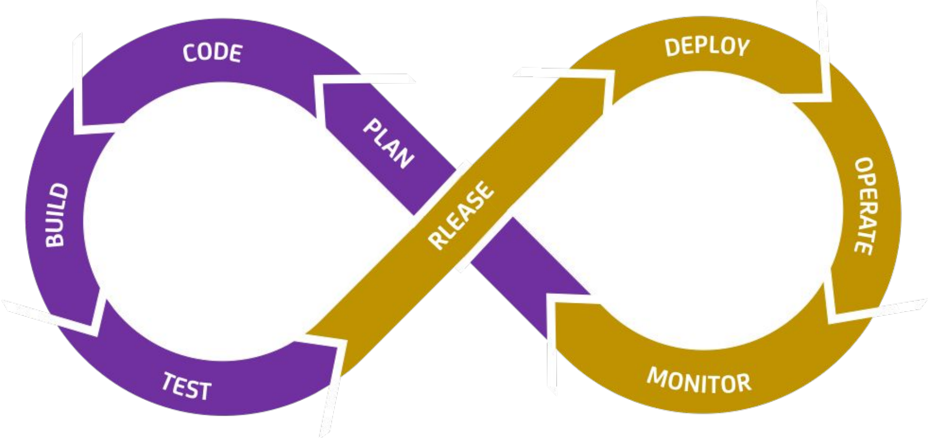
Translate to DevOps concepts

- Release
  - Save the image container in *container registry* → publish image
- Deploy
  - Combines published container image together with a set of user defined config parameters
  - Creates deployment unit
  - Actual deployment
  - Rollback on failure
- Production run
- Monitor the deployment



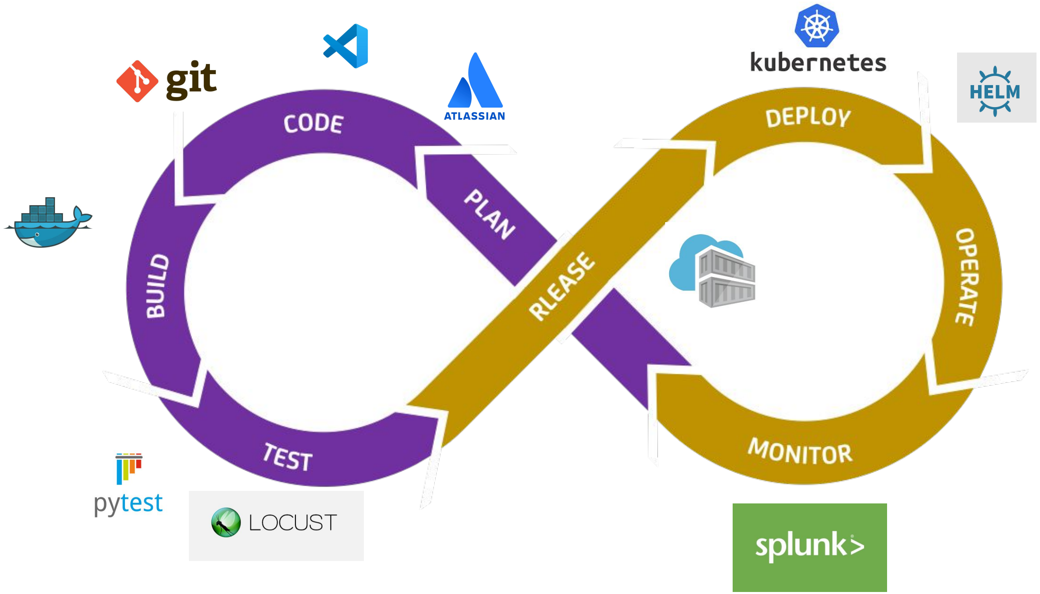
Continuous Delivery (CD)

CI/CD



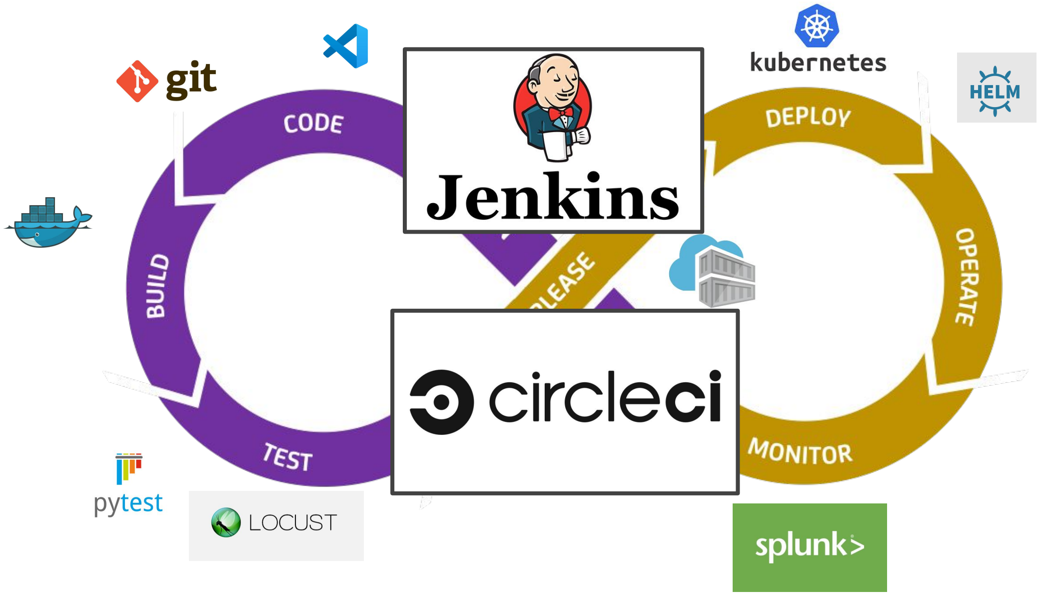
It's a DevOps practice that automates building, testing, and deploying code changes, enabling faster and more reliable software releases.

CI/CD - tools

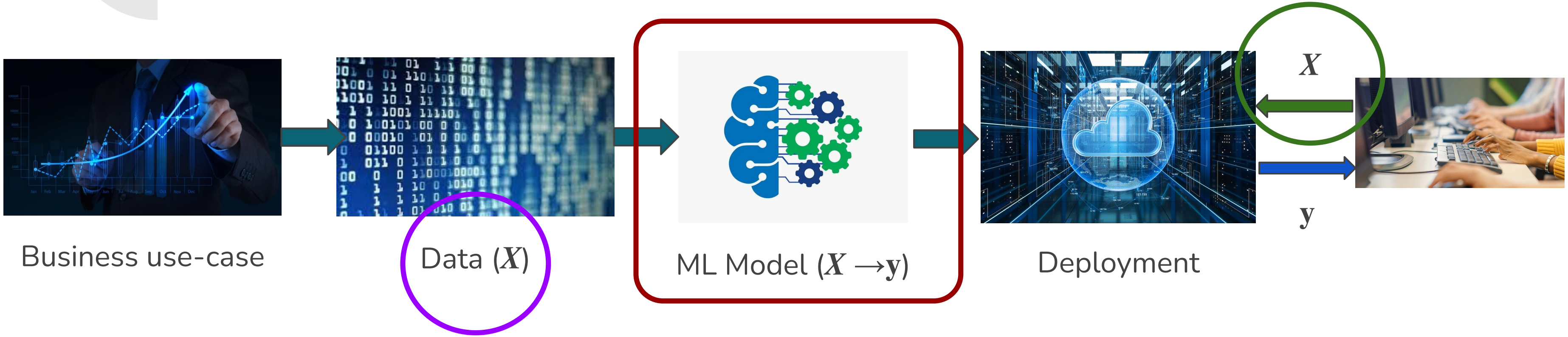


git
CODE
ATLASSIAN
PLAN
BUILD
TEST
RLEASE
kubernetes
HELM
DEPLOY
OPERATE
MONITOR
pytest
LOCUST
splunk>

CI/CD - tools

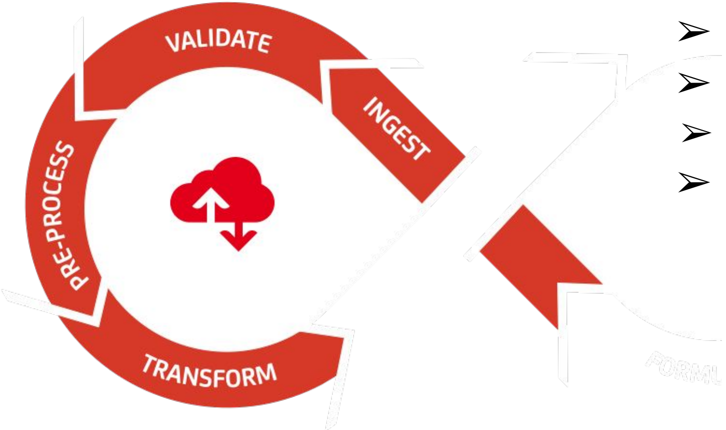

CODE
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
git
kubernetes
HELM
Jenkins
circleci
pytest
LOCUST
splunk>

Catch point: Model drift



- ➔ The data distribution in Production often varies from the data distribution used for the initial model training → **Model drift**
- ➔ How to identify the model drift? → **Human in the Loop**
- ➔ How to properly address this model drift? → **DataML/DataOps**

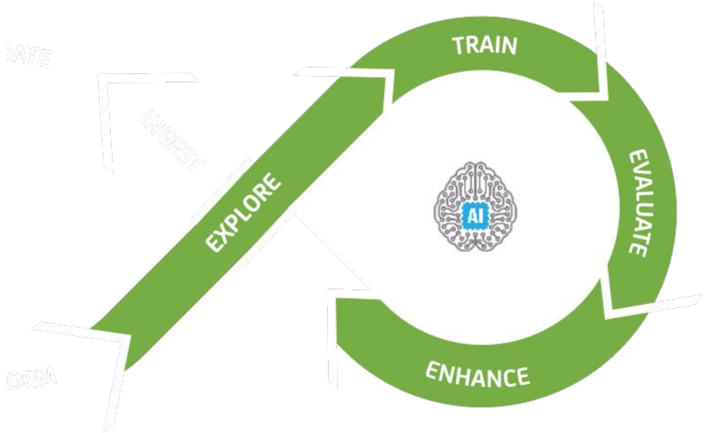
Data pipeline



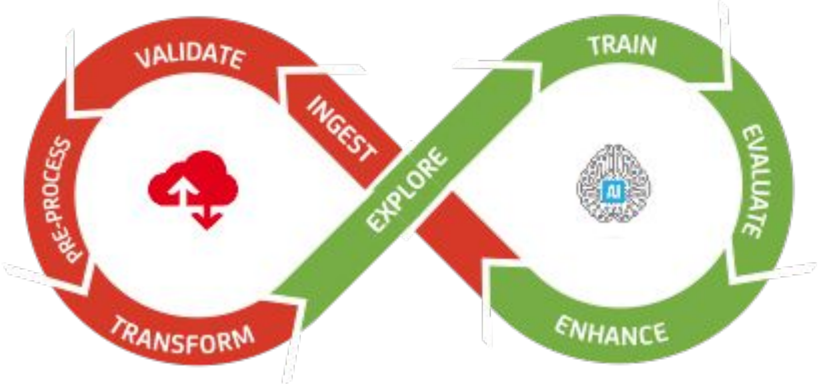
- ➢ Data ingestion with data versioning
- ➢ Data validation pipeline
- ➢ Data pre-processing
- ➢ Data transformation into suitable data object to facilitate the ML model dev ste

ML pipeline

- ➢ Data exploration with feature analysis
- ➢ ML algorithm selection and model training
- ➢ ML model evaluation
- ➢ Hyper-parameter tuning
- ➢ Model monitoring for enhancement

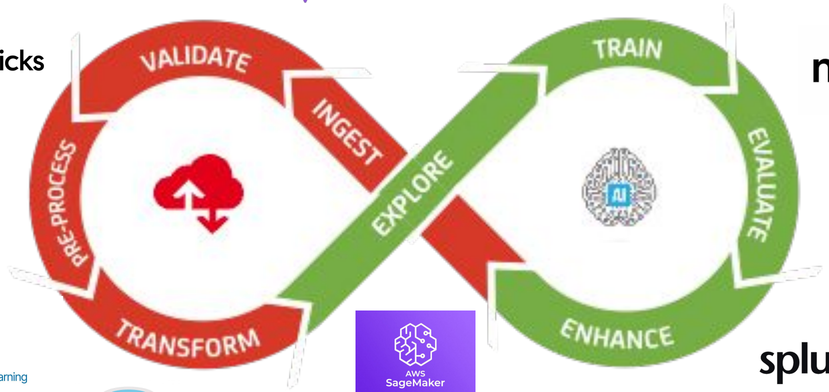


DataML = Data + ML

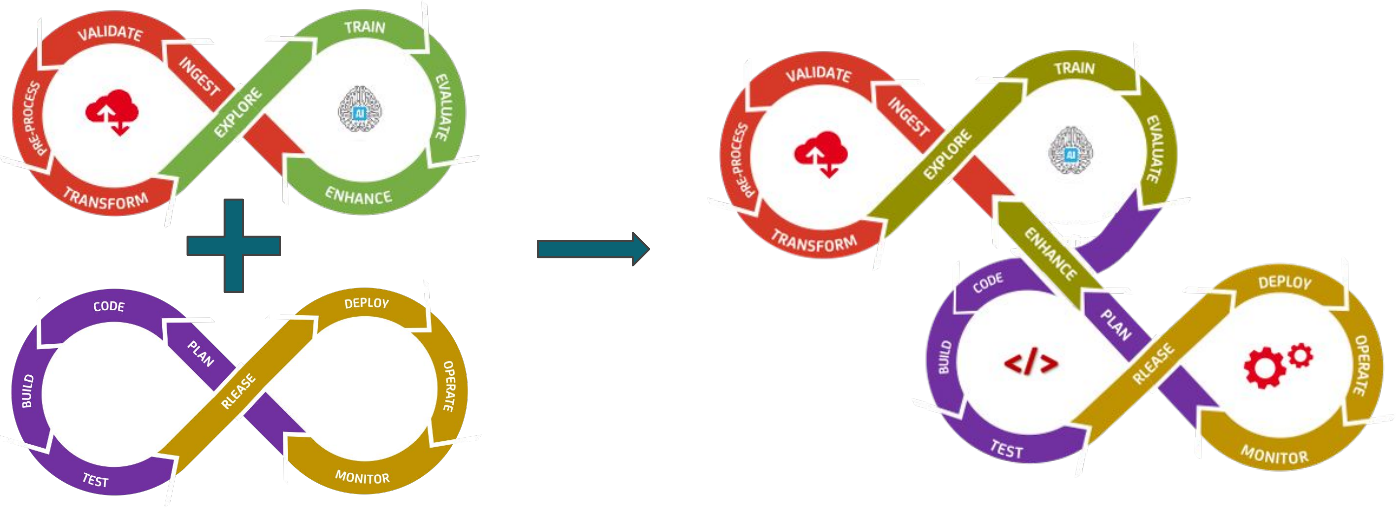


DataML is a **set of practices** that consolidates **the Data and ML pipeline** for a **robust data management and ML model training process**

DataML - tools



MLOps = DataML + DevOps



MLOps, or Machine Learning Operations, is a set of practices that **automate and streamline** the **machine learning (ML) lifecycle**, from **development to deployment** and **maintenance**. It ensures that the models are **built, tested, and deployed** in a **reliable, scalable, and consistent** way.

ML app lifecycle

- Starts with a business use-case
  - Completely new application or,
  - Replacement of an existing application
- Quick feasibility analysis
- Full business requirements and estimations →Code development, Infrastructure, Security/Compliance, Run management etc
- Consolidation of methodology → **ML model building**
- Industrialization → S/W engineering part for the end-to-end application
- Testing → **Integration test/Stress test**
- Monitoring setup with Human in the Loop
- User Acceptance Test (UAT)
- Go-live
- Post go-live support
- Run management

ML app lifecycle: Infrastructure env

- An infrastructure environment for software development is the **combination of underlying resources** like **hardware, networking, and software** that hosts and runs applications.
- Usually we have the following ***five*** environments in ML applications
  - **Dev** → where development, debugging happens
  - **Test** → Integration test, stress test etc happens
  - **Staging/UAT/Mute/QAT/Q0** → User Acceptance Test happens
  - **Production/C0** → Real application environment
  - **Disaster Recovery/DR** → In case of failure in Production environment
- **Test** environment is sometimes optional
- **UAT**, **Production** and **DR** should be identical

Types of ML app

- Completely new product with ML capabilities
- Total replacement of existing product with ML models
- Incremental replacement with ML models
- Multiple ML models for the same use-case

Based on the types of ML application use-cases, different deployment strategies can be used

Deployment strategies in ML prod

- Blue-Green deployment
  - **Two identical production environments** are maintained, **one active ("blue")** and **one inactive ("green")**. The **new model** is deployed to the green environment, and **when ready, traffic is switched** from **blue** to **green**.
  - **Minimizing downtime** and **quickly rolling back by simply switching traffic** back to the old environment.

- Canary deployment
  - The new model is released to **a small subset of users or traffic**. The **performance** of the **new model** is **monitored**, and if it's **stable, the rollout is gradually increased** to the rest of the users.
  - **Testing new models** in **a live environment with a limited blast radius**, allowing for a safe gradual rollout.

Deployment strategies in ML prod

- Rolling deployment
  - The **new model** is deployed **incrementally, gradually replacing** instances of the **old model**. Each new instance is brought up and the old one is taken down one by one or in small batches.
  - **Updating services** with **minimal downtime** by **gradually phasing out** the old version.

- A/B testing
  - **Two different versions** of **a model (A and B)** are deployed to **different user segments. A/B testing** compares **their performance on real-world data** to see which one is **better** based on **predefined metrics**.
  - **Comparing** different models **head-to-head** to **choose the superior one** before a full rollout.

Deployment strategies in ML prod

- Shadow deployment
    - A **new model** version runs in **parallel with the existing one**, receiving a **copy of the production traffic**. The new model's **results are recorded for analysis** but **are not used for actual responses**; the **live system continues to serve users based on the old model**.
    - **Evaluating** a new model's **performance, accuracy, and behavior under real-world load** without any risk to end-users.

Real ML production components

- API gateway → To receive the requests from backend
- Message broker → Fault tolerance
- Orchestrator → Contains business logic
- ML serving → RestAPI + ML model
- Output preparation block
- DB tables, Internal storages → To save app/data states
- Logging and monitoring unit
- Human in The Loop

Real MLOps production components

- Data lake → For data and features store with versioning
- ML model registry
- Data processing code container
- ML model training code container
- ML model evaluation code container
- MLOps pipeline framework
- DB table, internal storage

Compliance in ML applications

- Responsible AI → AI Act, AI ethics
    - **Responsible AI** is an approach to developing and deploying artificial intelligence from both an **ethical and legal standpoint**. The goal is to **employ AI** in a **safe, trustworthy and ethical way**.
    - The **AI Act** is the European Union's **comprehensive law** to regulate artificial intelligence (AI) systems, setting rules for their development and use to ensure they are **safe, trustworthy, and aligned with EU fundamental rights and values**
- **Transparency** and **Accountability** are the two main pillars of the AI Act
- **MLOps** helps organizations **comply with the EU AI Act** by providing **a framework** for **auditable, transparent, and reproducible AI lifecycles**, which includes **version control, automated testing, and continuous monitoring**

Demo: ML serving

- Pre-trained ML model from Kaggle which classifies an image having either a cat or a dog into cat/dog category (link)
- Wrap the model within RESTful API
  - Unittest
- CI pipeline → Docker
- CD pipeline →Local infrastructure

ML serving testing using python code and postman

Demo: MLOps

- Setup MLOps tools
  - MLFlow together with minio and mysql
- Run a ML model training code using Kaggle dataset (link) about property value using Random forest classifier
- Model run tracking in MlFlow

Quick recap

- Explained the concept of ML in prod
- Highlighted major catchpoints
  - ML serving
  - Model drift
- Blending with DevOps practice to obtain MLOps
- ML life cycle
- Types of deployment strategies
- Compliance in ML production application
- Demonstration of ML serving and MLOps in an approximated scope

Q&A

Thank you for your attention!