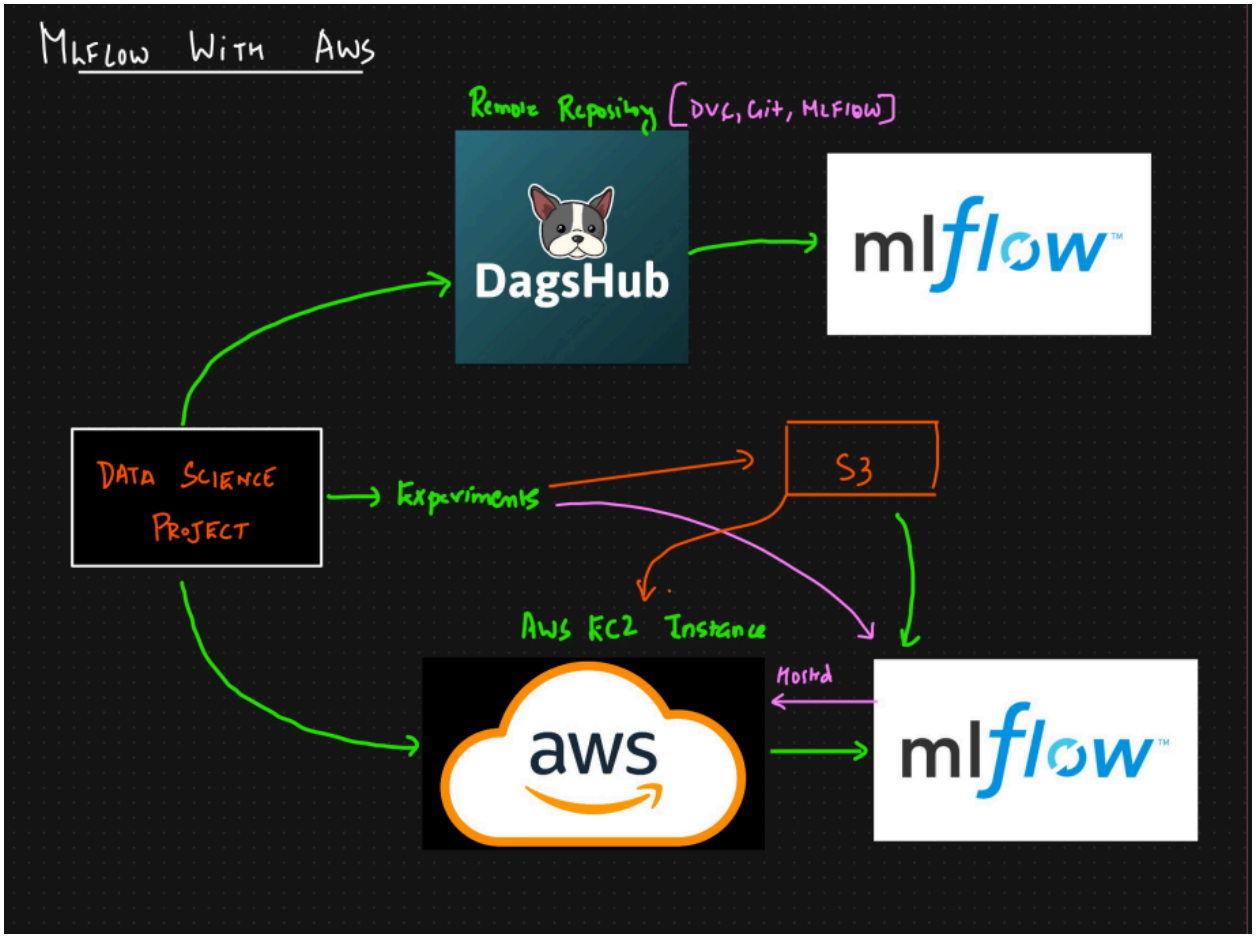
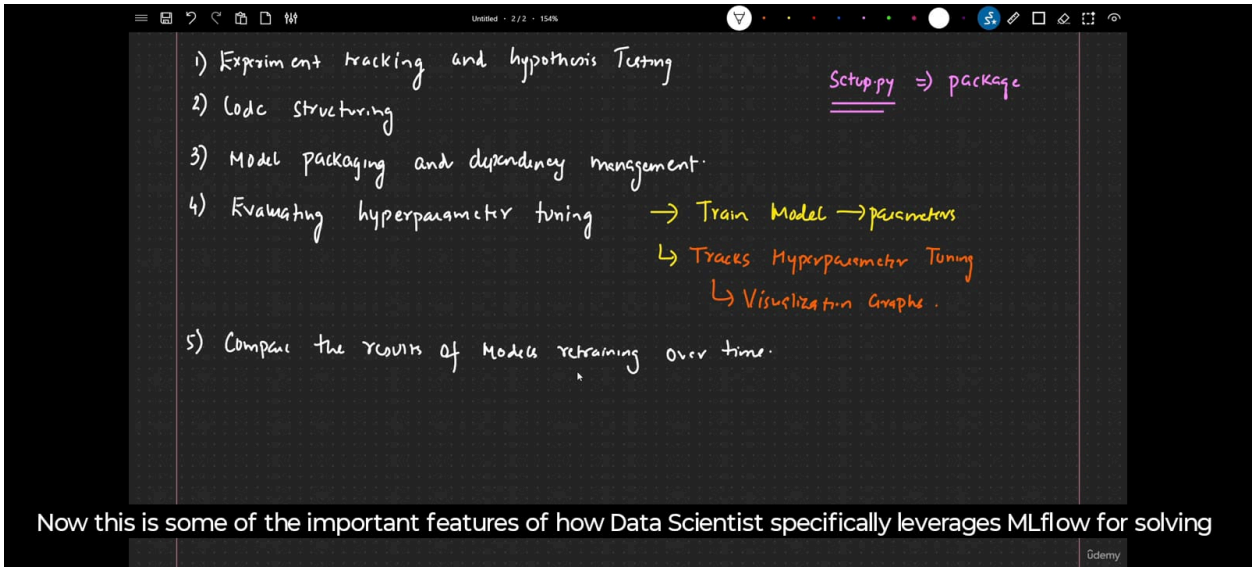




Now this is some of the important features of how Data Scientist specifically leverages MLflow for solving

1) Evaluate And Experiment with large language models
2) Create custom prompts and experiment.
3) Deciding on the best base model suitable for project Requirement.

Usecases

1) Experiment Tracking → Track parameters And metrics → UI →
2) Model Selection And Deployment
3) Model performance monitoring.
4) Collaborative Project :

If you're working with a team I think MLflow is the best MLOps platform, which will actually help you

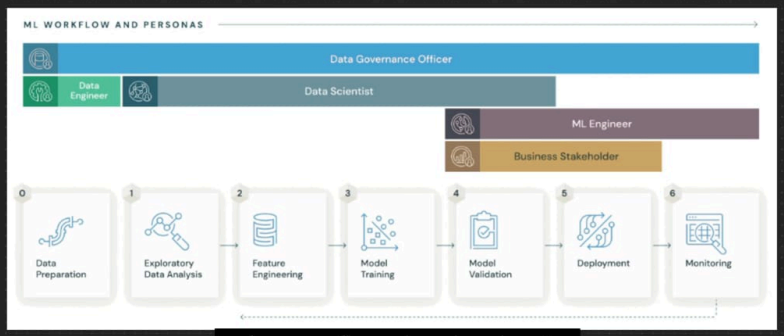
MLOps Professional

1) Manage the lifecycle of trained models, both pre and post deployment.
2) Deploy models securily to the production environment
3) Manage deployment dependencies.

Prompt Engineering Users

1) Evaluate And Experiment with large language models
2) Create custom prompts and experiment.
3) Deciding on the best base model suitable

3) Who uses MLFlow
4) Usecase of MLFlow

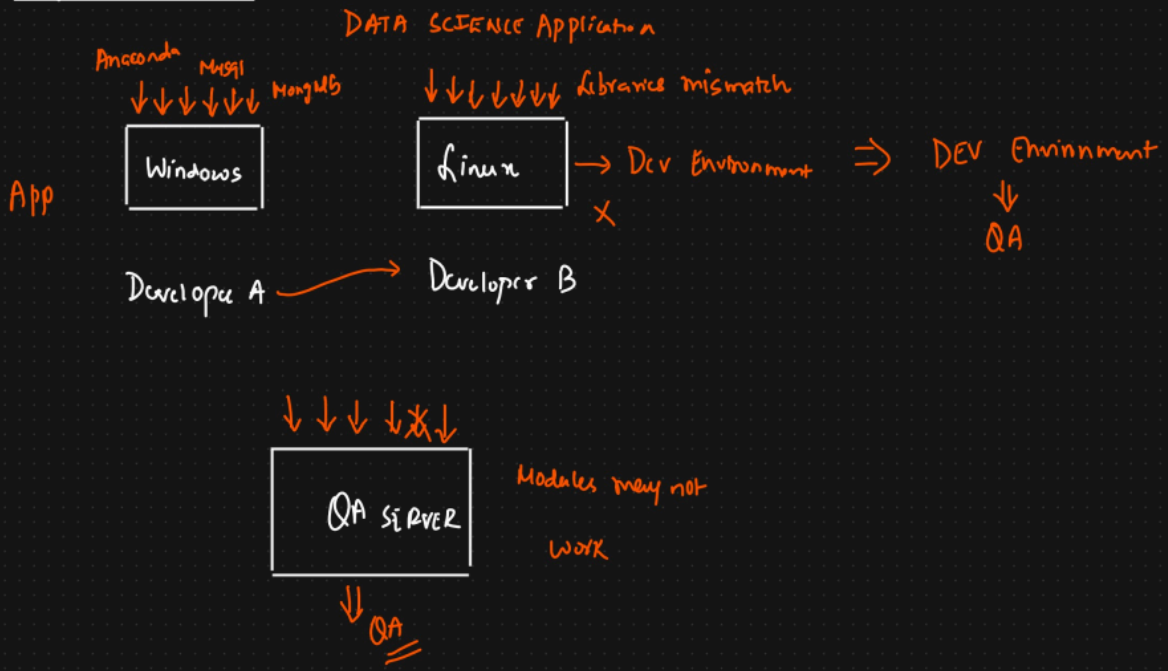


So here I will go ahead and write.

Docker Series.

1. What is Docker?
2. What are Containers?
3. Container Vs Virtual Machine
4. Docker Image Vs Container
5. Practial Implementation of Dockers.

Why Containers?

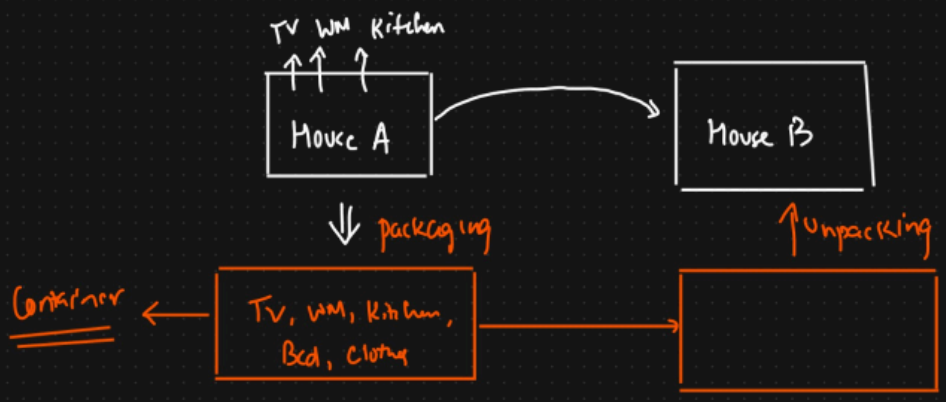


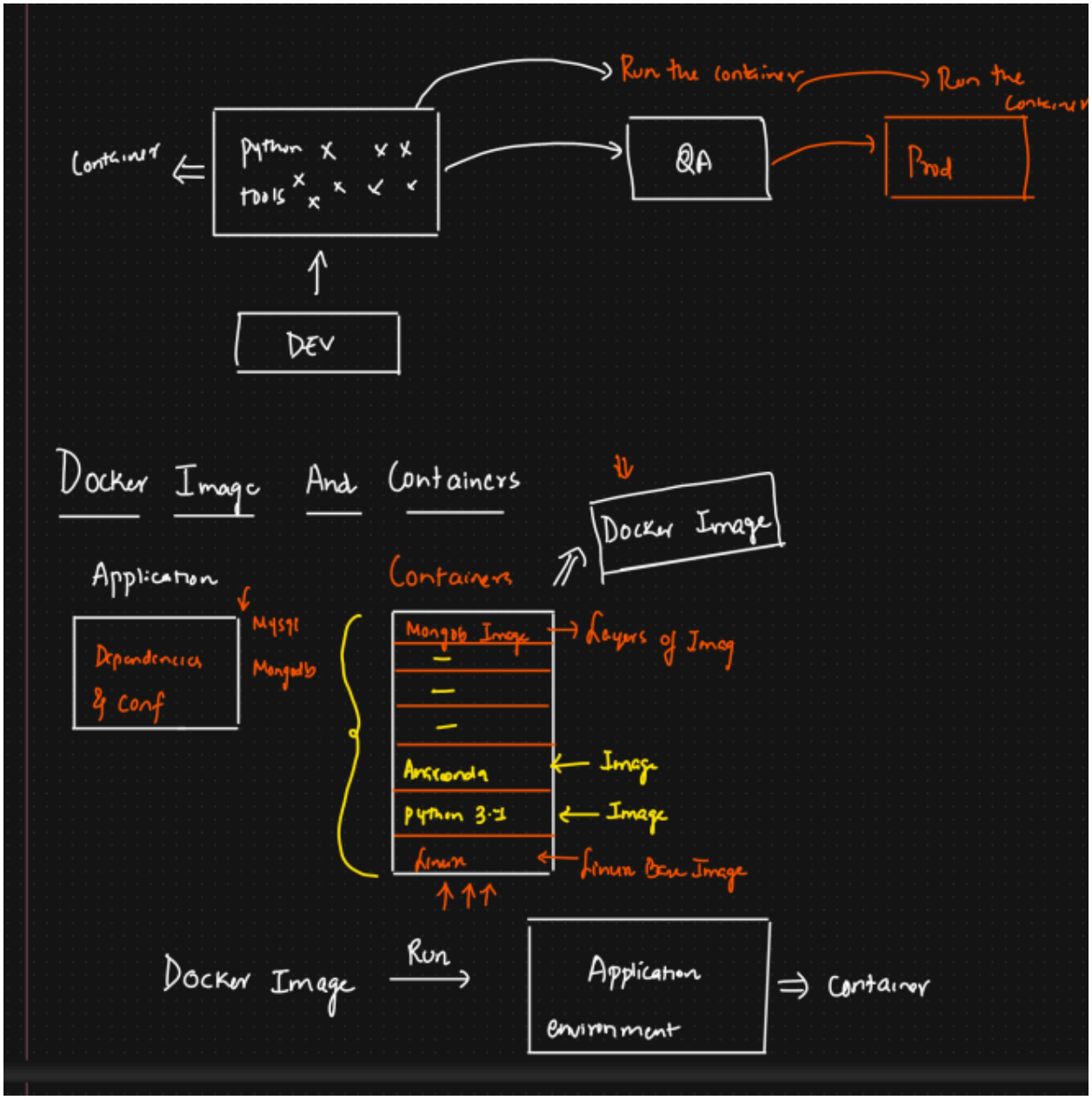
Containers

*) A way to package application with all the necessary dependencies And configuration

*) Portable artifact, easily share and move this package to any environ

*) Makes development and deployment more easy and efficient




Run the container
Run the container
Container
Python
Tools
QA
Prod
DEV
Docker Image And Containers
Docker Image
Application
Containers
Dependencies & Conf
Mysql
Mongodb
Mongodb Image
Layers of Image
Anaconda
Image
Python 3.7
Image
Linux
Linux Base Image
Docker Image
Run
Application Environment
Container

Docker Image

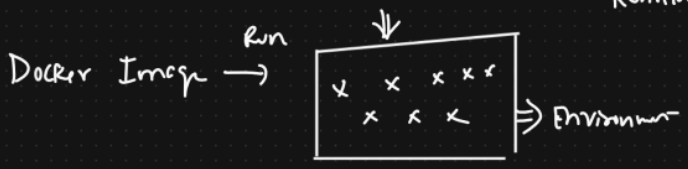
1. Package or Artifact
2. Move or share this Artifact

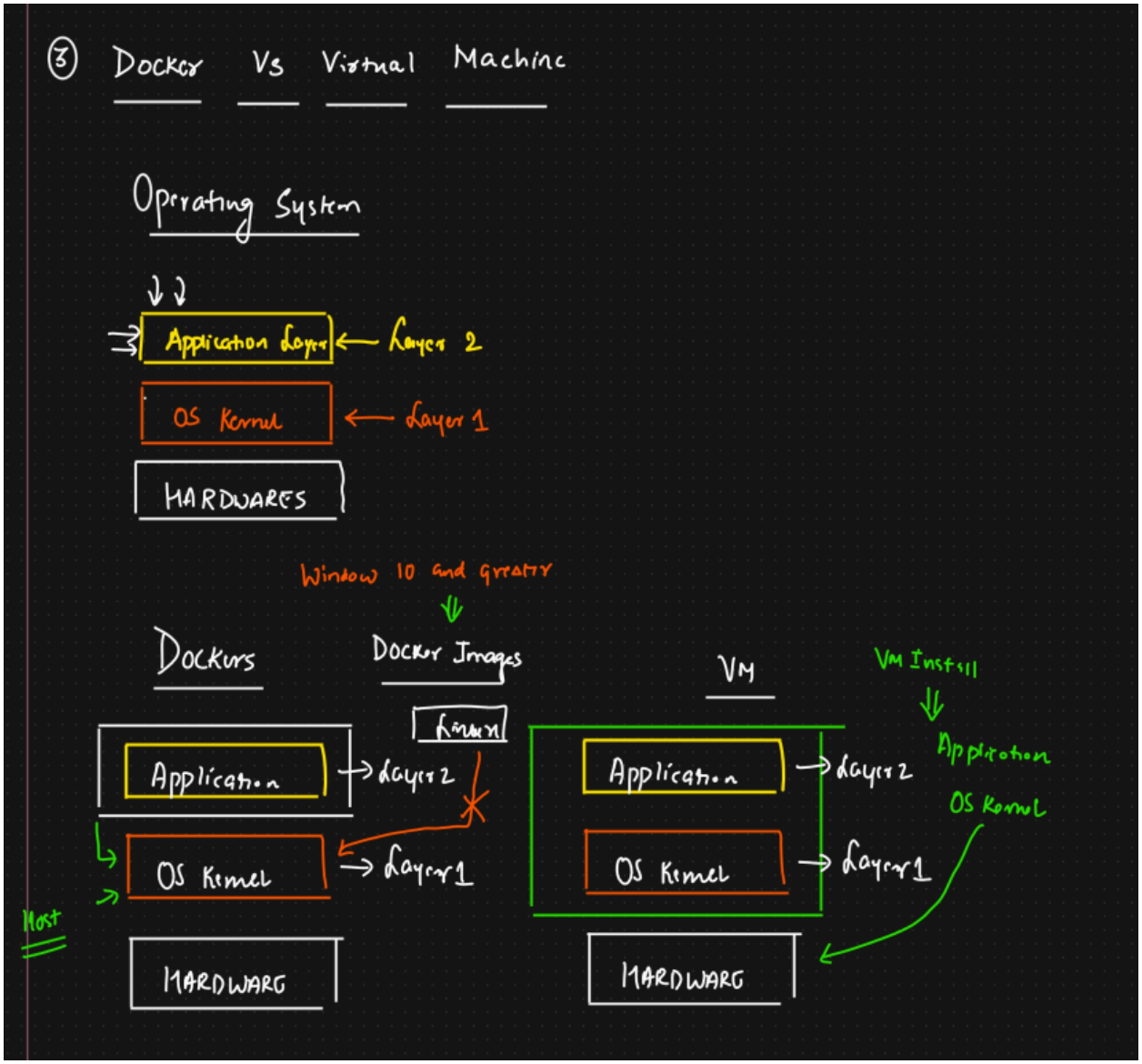
Container

1. Docker Image → Run
2. Start the Application
3. Create a container => Environment

⇓

Running




③ Docker Vs Virtual Machine
Operating System
Application Layer
Layer 2
OS Kernal
Layer 1
HARDWARES
Window 10 and greater
Dockers
Docker Images
VM
VM Install
Application
Layer 2
OS Kernel
Layer 1
Application
Layer 2
OS Kernel
Layer 1
Application
OS Kernel
Host
HARDWARE
HARDWARE

① Dockers :- Docker Image size is usually smaller. (MB)

VM :- Size will be huge. (GB)

② Dockers :- Dockers Containers start and run must faster

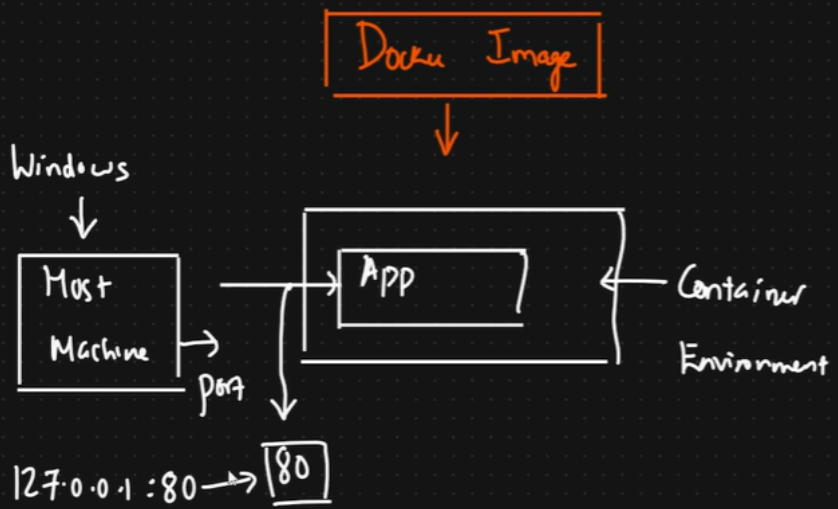
VM :- VM is comparatively slower

Compatility

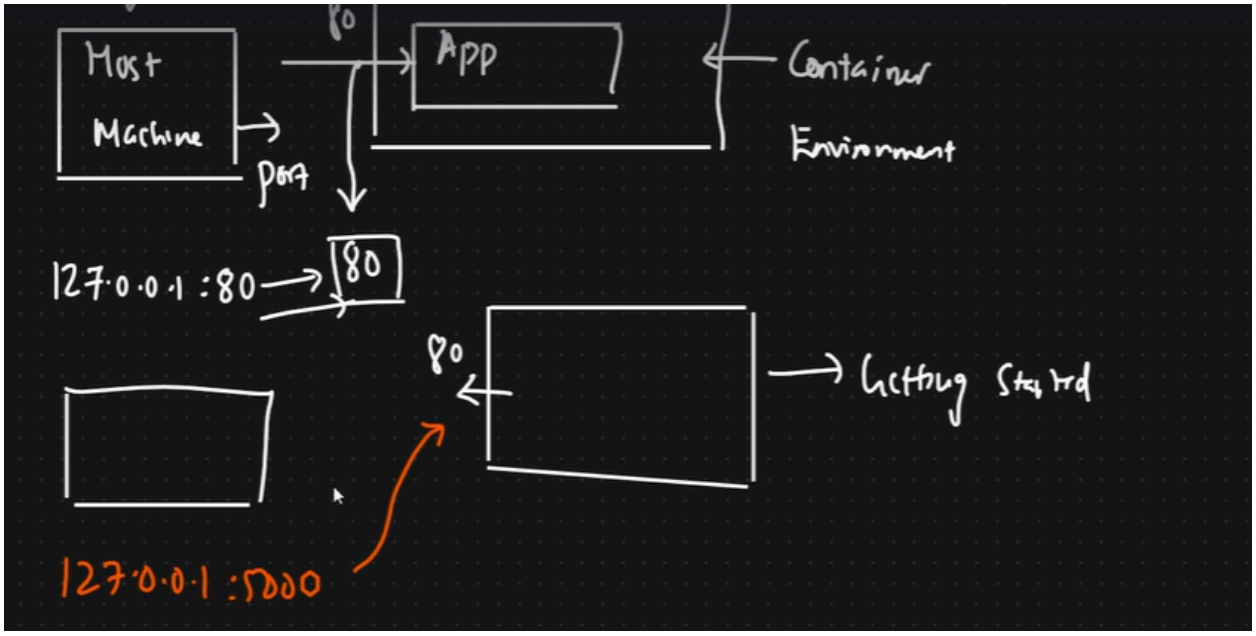
{ Install VM on any OS }

{ Docker Images → Compatility Issue }

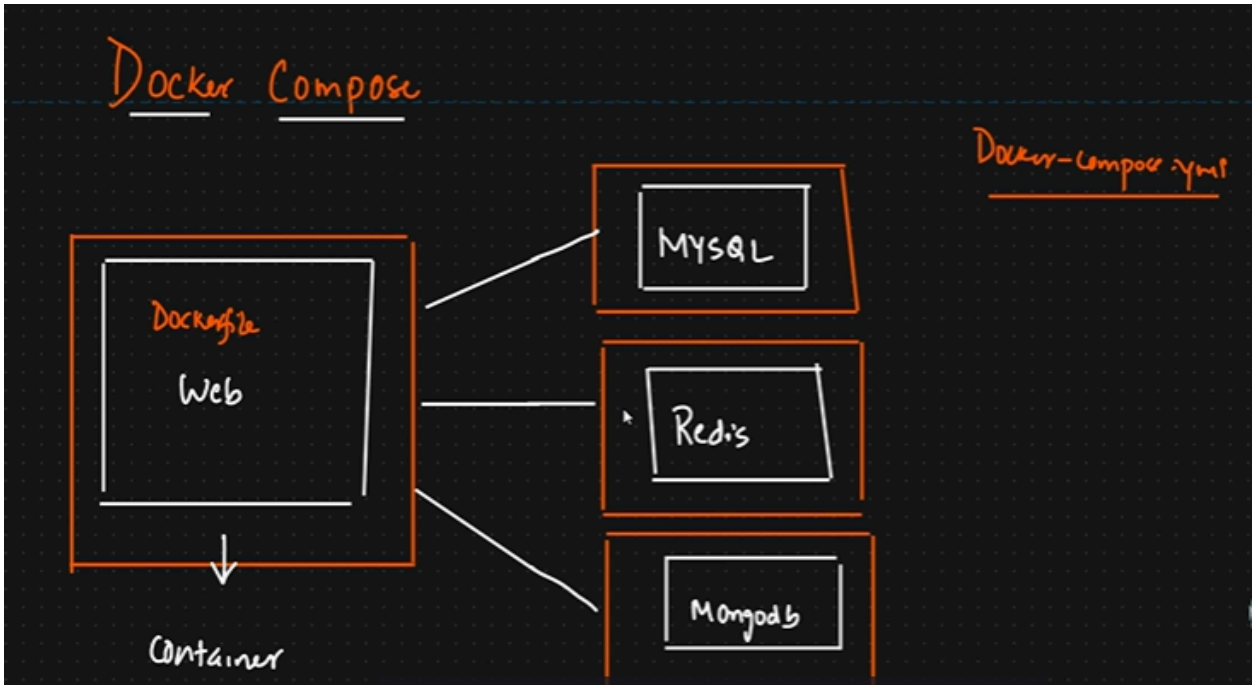
Container Port And Host Port



Container port can be same , host port needs to be different

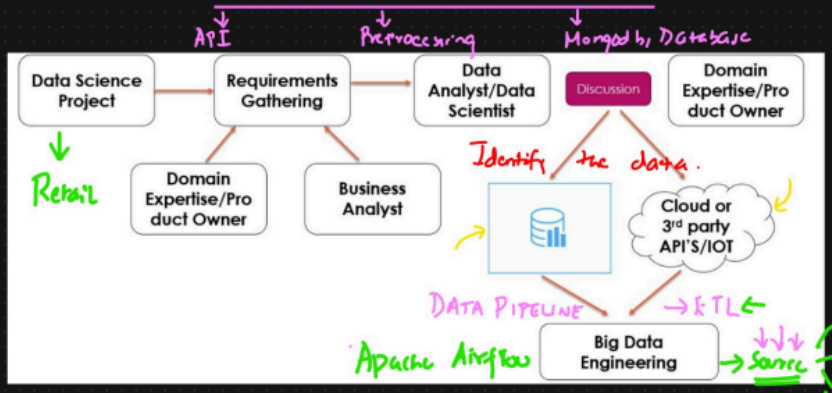


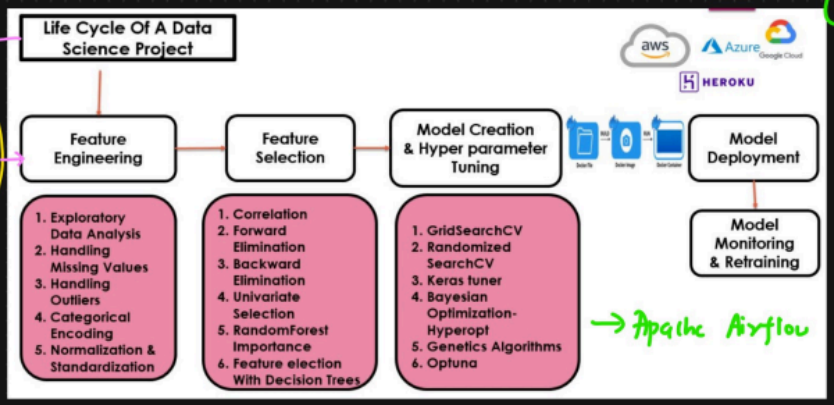
Docker compose used to run multiple docker containers.



Apache Airflow

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. It allows you to define complex workflows as code and manage their execution. Airflow is commonly used for data pipelines, where tasks like data extraction, transformation, and loading (ETL) are orchestrated across multiple systems.





Read DATA from a Source

Schedule

Airflow

Key Concepts In Apache Airflow

1) DAG (Directed Acyclic GRAPH) : Collection of tasks that you want to Schedule and run.

↓ Directed Graph

① Directed → Task must have a specific seq
② Acyclic
↳ No task should be depend on itself.

A<<B<<C

A>>B>>C

2) Tasks : Task represent the individual unit of work in a DAG.

1) python Function
2) Querying a Database
3) Sending an HTTP Request.

3) Dependencies :

Tasks in a DAG have dependencies, meaning one task might need to finish before another task can start. These dependencies allow you to control the order in which tasks are executed. Airflow provides mechanisms like set_upstream and set_downstream to define these dependencies between tasks.

③ Why Airflow For MLOPS

In MLOps (Machine Learning Operations), orchestrating ML workflows efficiently is crucial for ensuring that data pipelines, model training, and deployment tasks happen smoothly and in an automated manner. Airflow is well-suited for this purpose because it allows you to define, automate, and monitor every step in an ML pipeline.

1) Orchestrating ML Pipelines And ETh Pipelines

DAG → TASKS, Dependencies

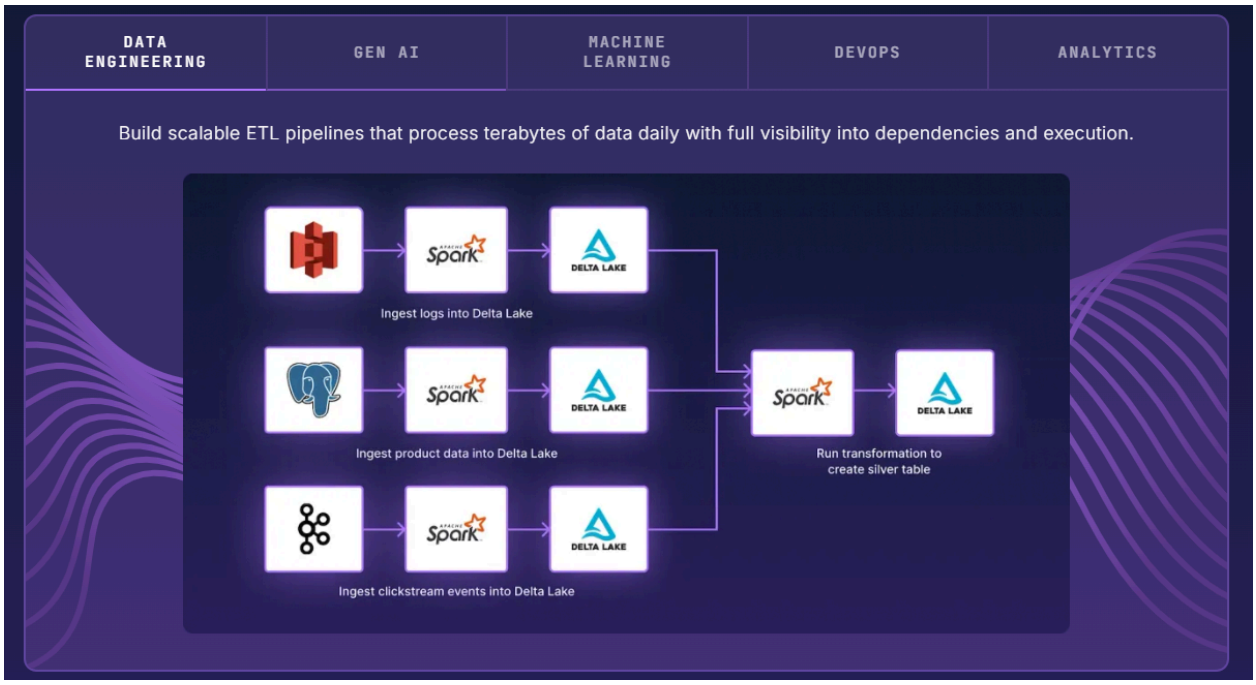
2) TASK Automation

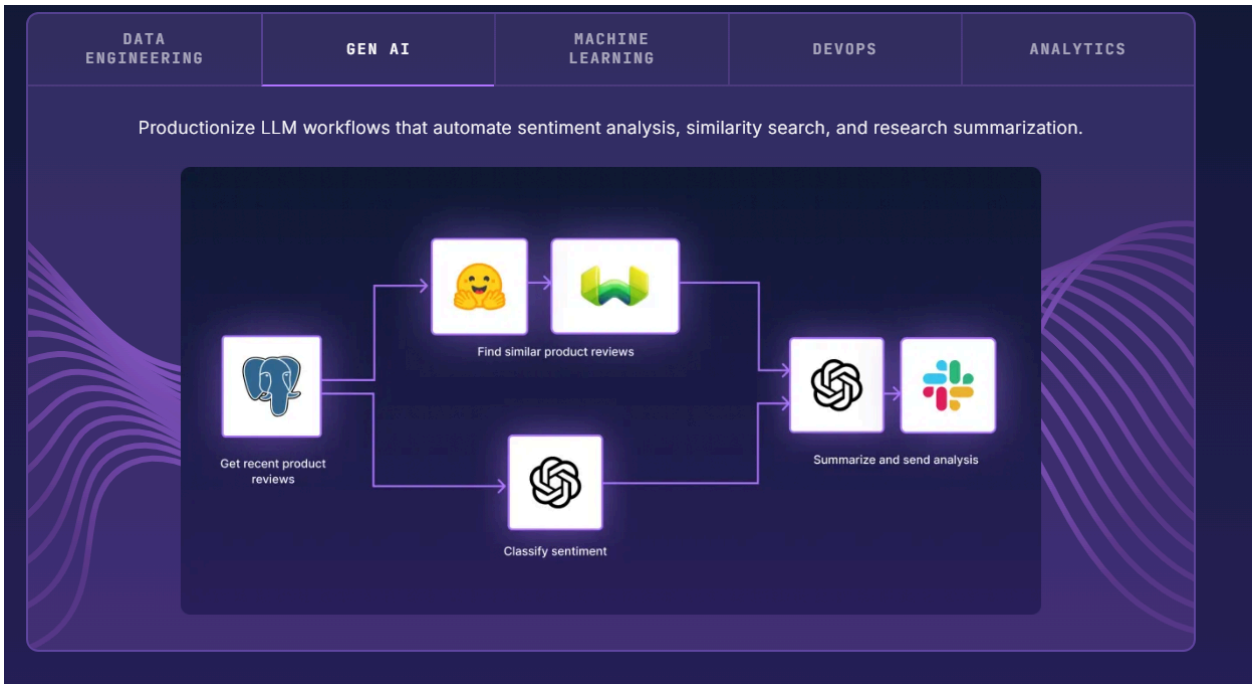
3) Monitoring And Alerts

3) Monitoring And Alerts

1) Real Time Monitoring → Airflow UI
2) Tasks logs
3) Alerts And Notifications → Emails
4) Retry Mechanism → Retry Task → Pre defined Rules.

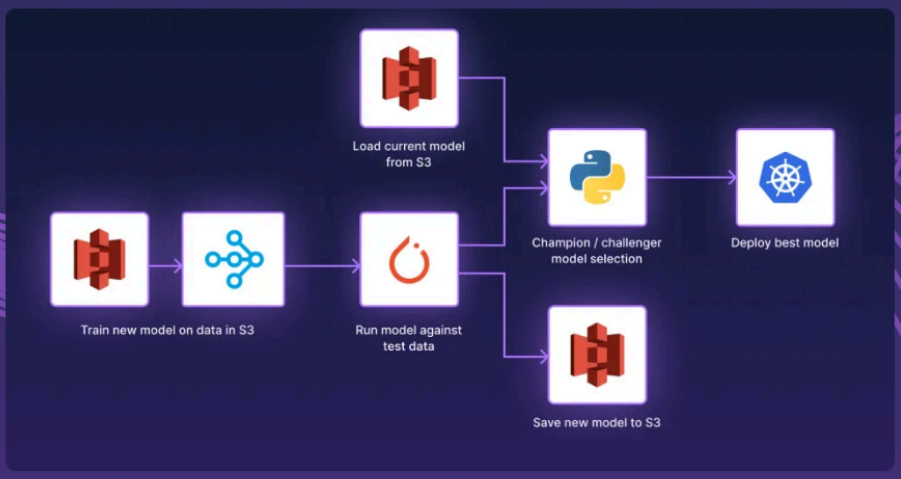
Astronomer with Airflow





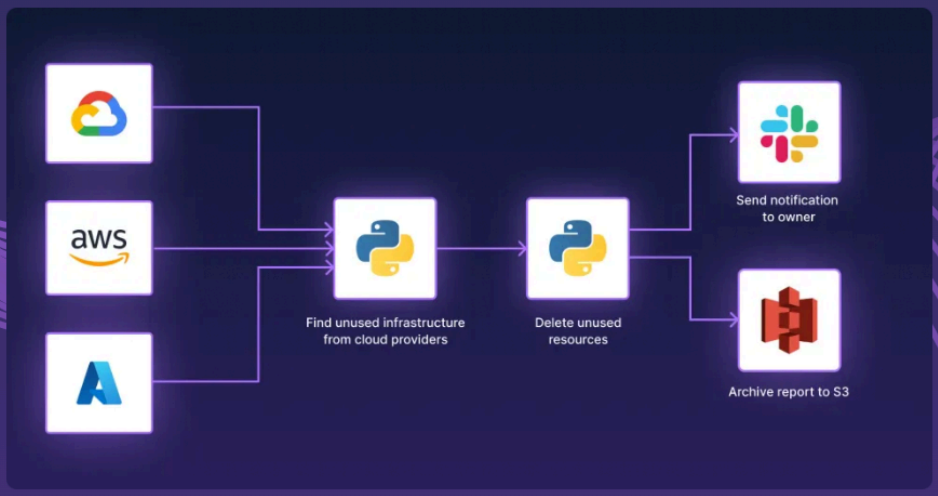
DATA ENGINEERING | GEN AI | **MACHINE LEARNING** | DEVOPS | ANALYTICS

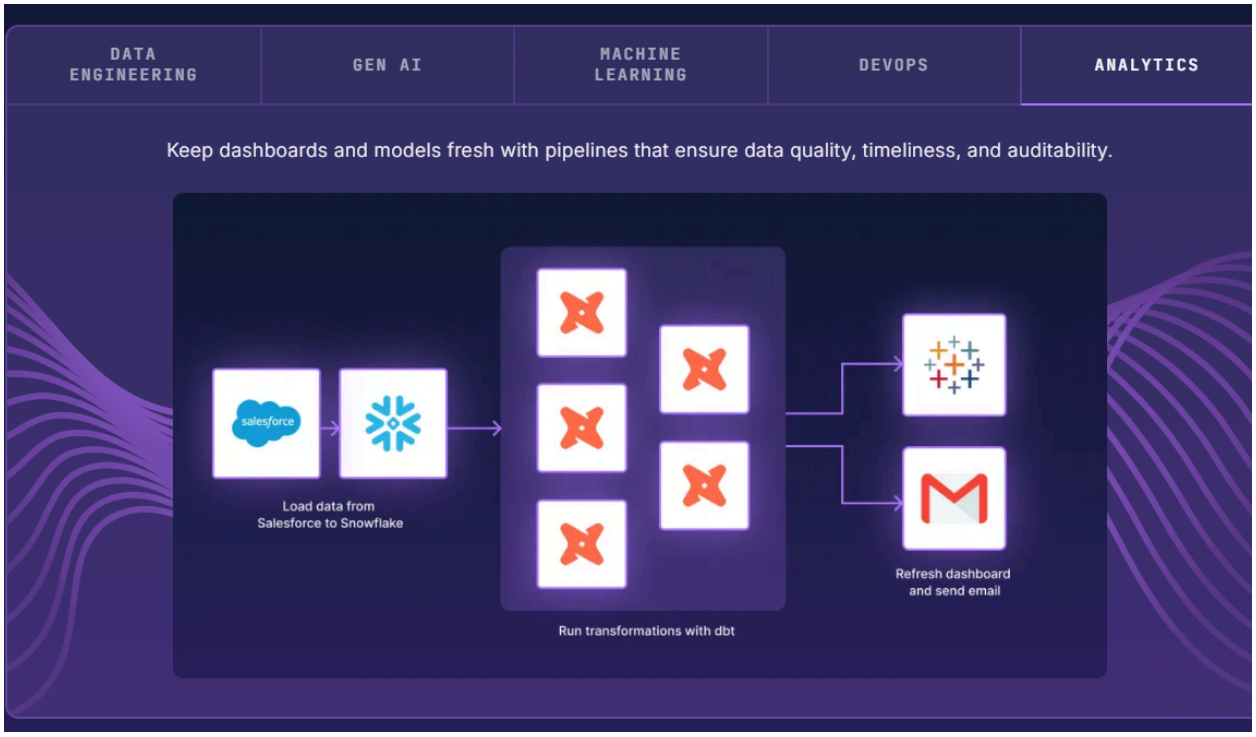
Operationalize complex ML workflows—from model training to inference—with reliability and reproducibility.

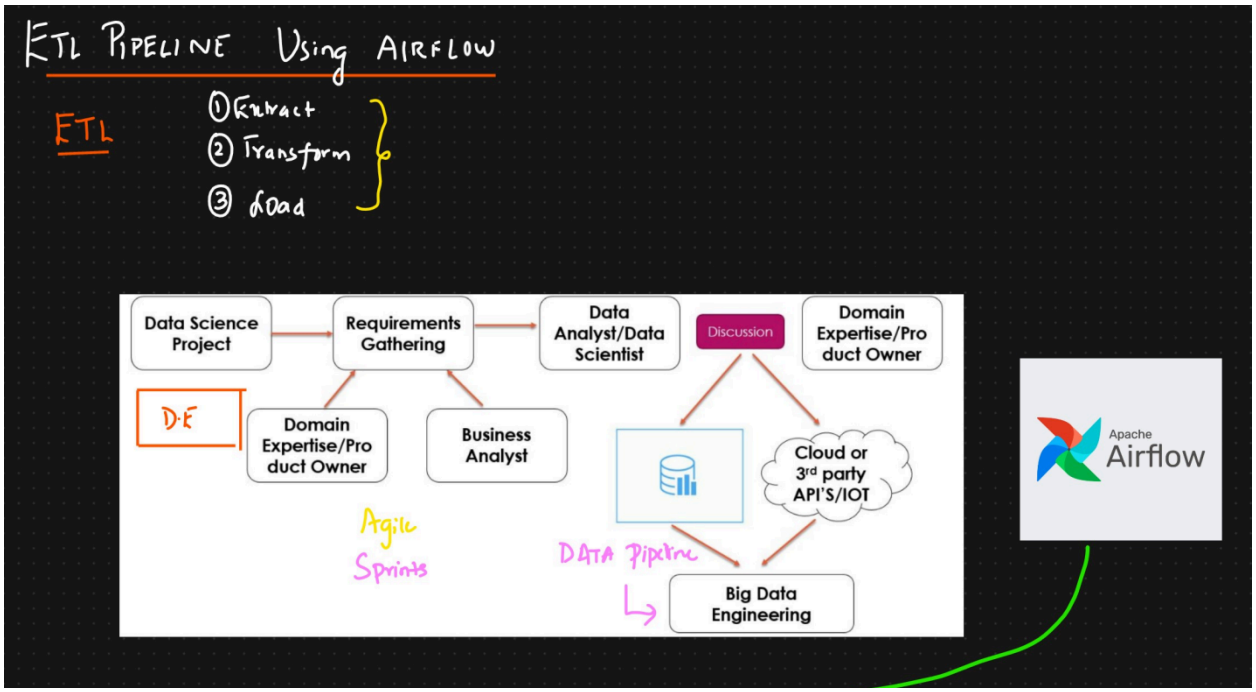


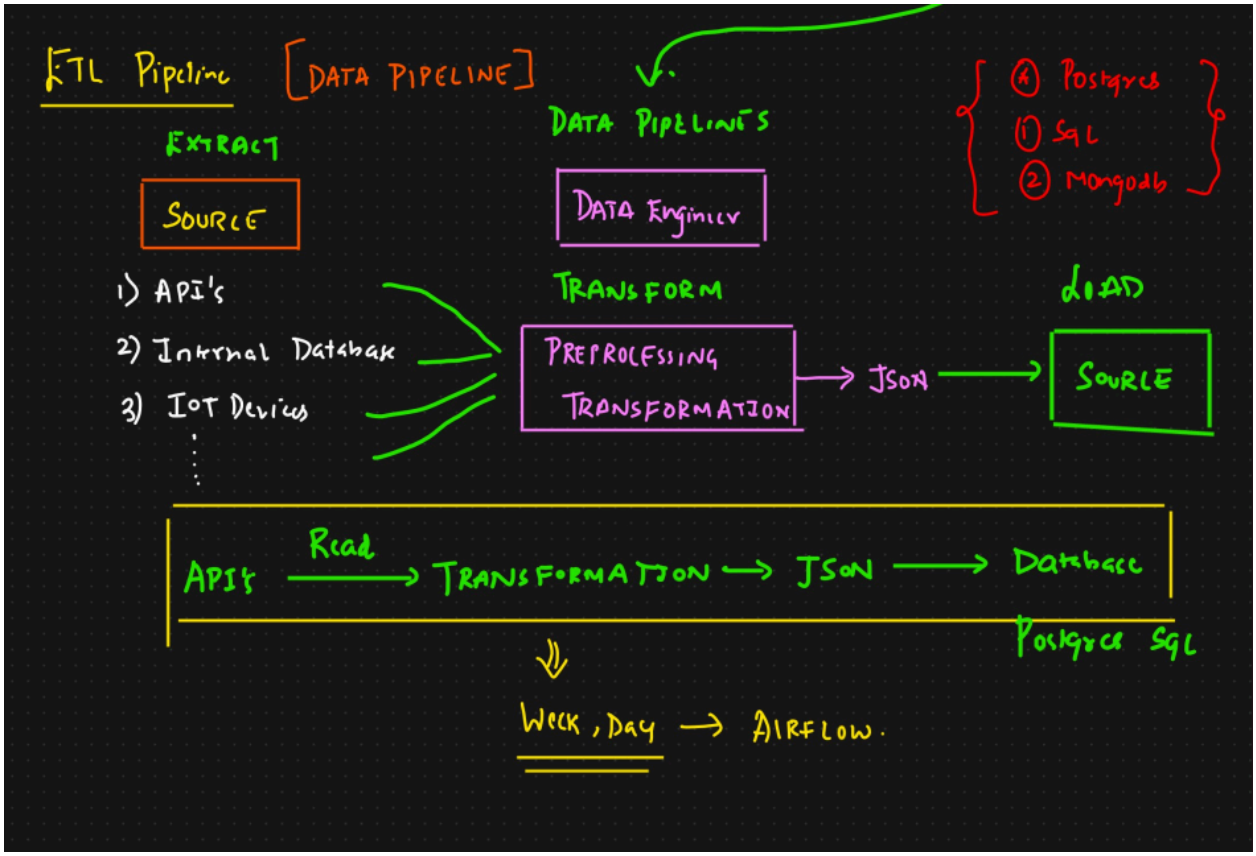
DATA ENGINEERING | GEN AI | MACHINE LEARNING | **DEVOPS** | ANALYTICS

Build infrastructure automation and reporting.




DATA ENGINEERING
GEN AI
MACHINE LEARNING
DEVOPS
ANALYTICS
Keep dashboards and models fresh with pipelines that ensure data quality, timeliness, and auditability.
salesforce
Load data from Salesforce to Snowflake
Run transformations with dbt
Refresh dashboard and send email

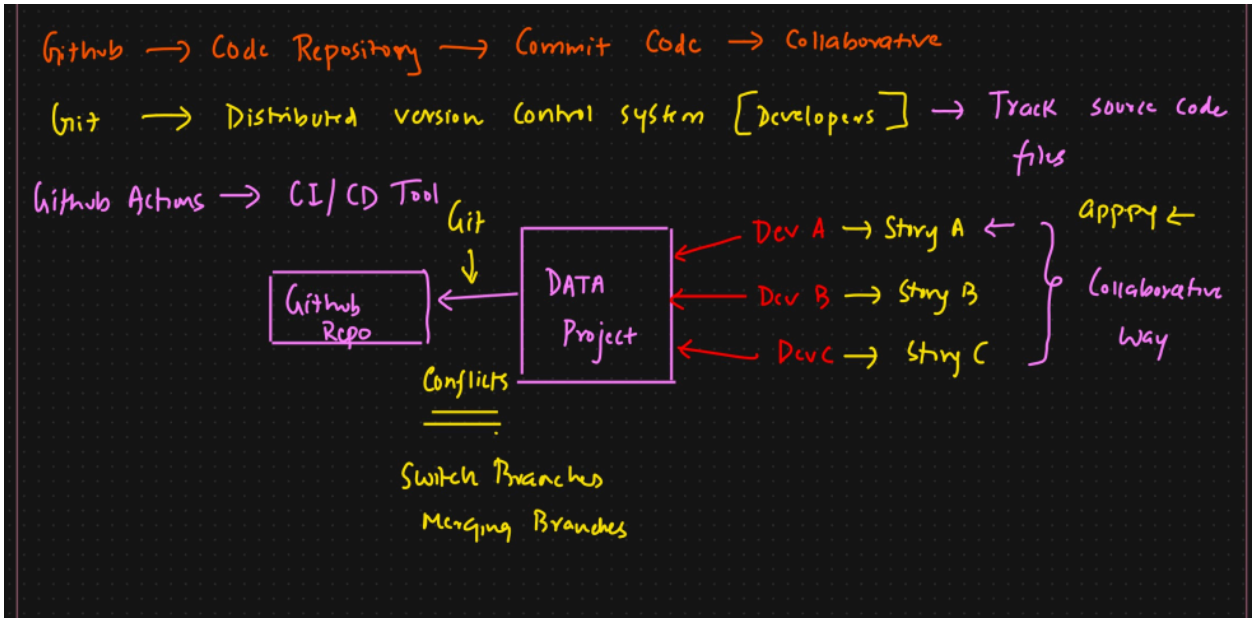

ETL Pipeline Using Airflow
ETL
1 Extract
2 Transform
3 Load
Data Science Project
Requirements Gathering
Data Analyst/Data Scientist
Discussion
Domain Expertise/Product Owner
DE
Domain Expertise/Product Owner
Business Analyst
Agile Sprints
Cloud or 3rd party API'S/IOT
DATA Pipeline
Big Data Engineering
Apache Airflow



Github Actions – CI/CD Pipelines

GitHub Actions is a powerful automation tool provided by GitHub that enables developers to automate tasks related to their software development workflows. It allows for the creation of custom workflows, which can be triggered by various events, such as pushes to a repository, pull requests, or scheduled events. These workflows can be used for various purposes, including Continuous Integration (CI), Continuous Deployment (CD), testing, and more.

Github Action → CI/CD Tool
↓
Github.

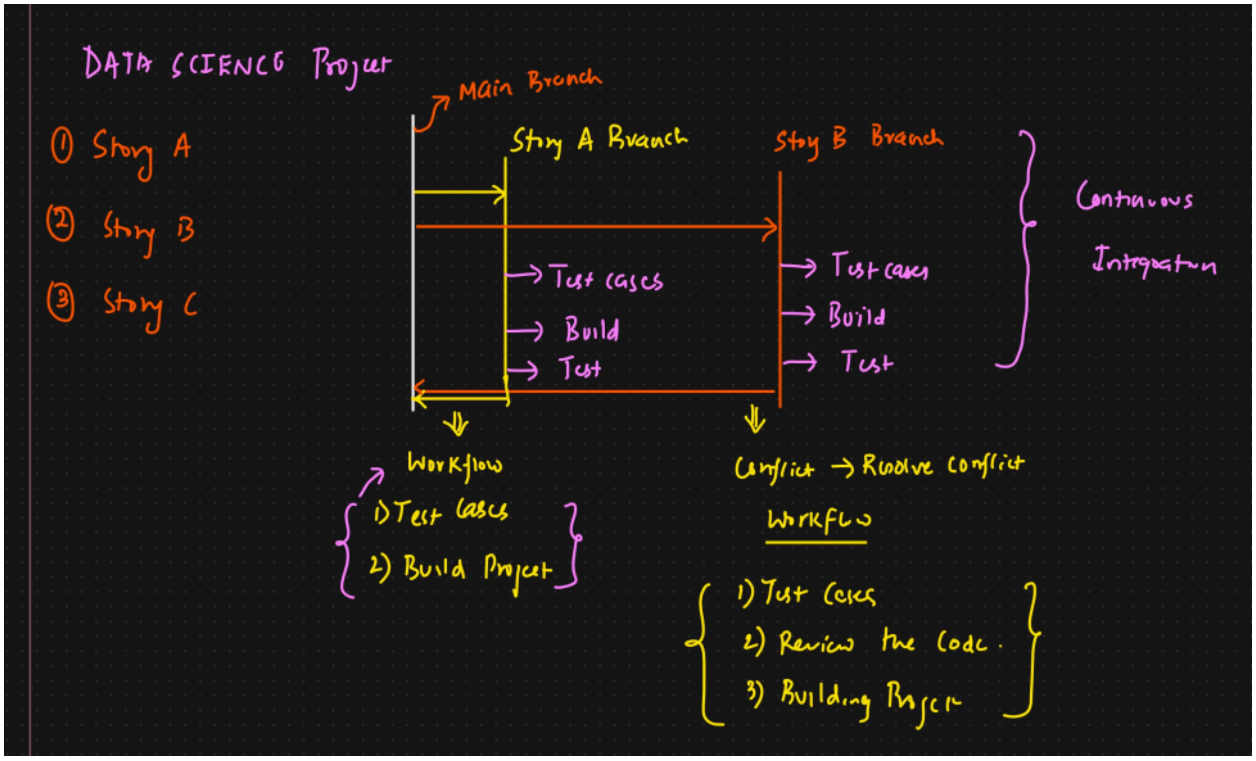


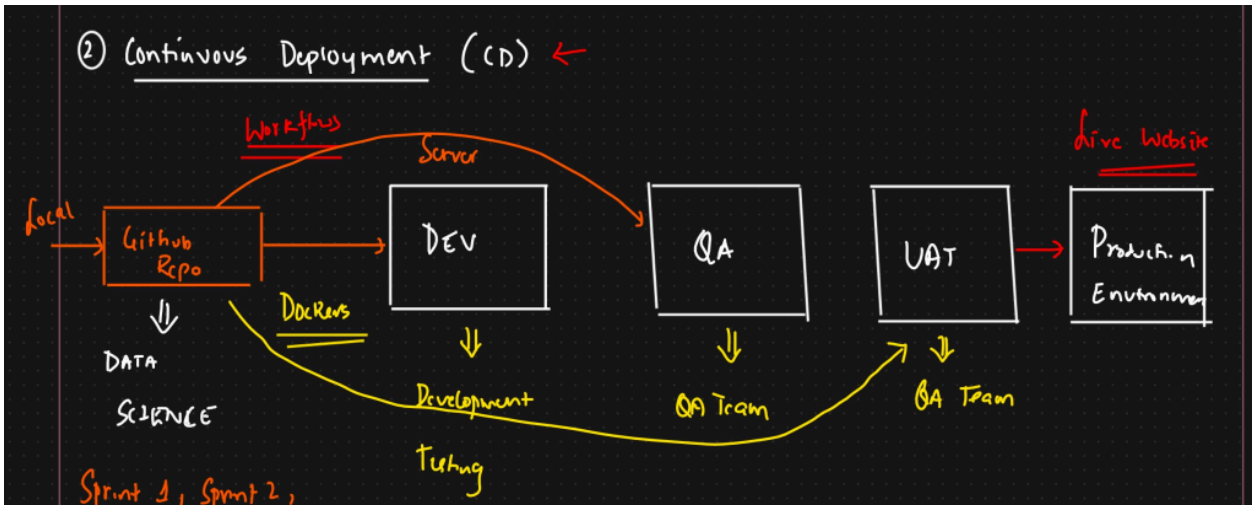
Continuous Integration And Continuous Deployment

Continuous Integration (CI) and Continuous Deployment (CD) are two key practices in modern software development, and GitHub Actions can facilitate both:

Continuous Integration (CI): CI is a practice where developers frequently merge their code changes into a shared repository. Each merge triggers an automated build and testing process to ensure the changes do not break the existing functionality. With GitHub Actions, developers can set up workflows to automatically build and test their code every time they push changes to the repository or create a pull request.

Continuous Deployment (CD): CD extends the concept of CI by automating the deployment of code to production environments after it passes all required tests. GitHub Actions can be configured to automatically deploy applications to various environments, such as staging or production, once the code passes CI checks. This practice reduces the time between development and deployment, allowing for faster delivery of features and updates.





2. Workflow - Developers Workflow

A developer's workflow refers to the series of steps, practices, and tools that a developer or a team of developers follows to write, test, collaborate on, and deploy code effectively. This workflow is designed to streamline the software development process, enhance productivity, reduce errors, and ensure high-quality code output. A well-defined developer's workflow incorporates various stages of development, from coding to deployment, and often integrates tools for version control, code review, testing, and continuous integration/continuous deployment (CI/CD).

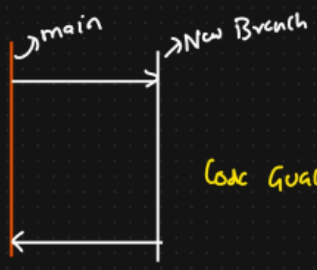
Key Stages

1) Coding → IDE (VS code) → Python, javascript → Coding standards, Best practise

2) Version Control : Git → Manage the codebase . Multiple developer to Collaborate

- Commit
- Branches
- push
- pull
- Resolve conflict

3) Code Review



Peer Code Review → Best Practices

4) Testing :

Automated Testing

1) Unit Testing
2) Integrating Testing
3) End to End Test Cases

5) Continuous Integration (CI)

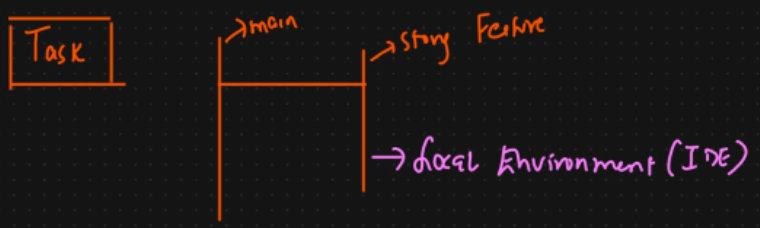
Building, Testing → Notified → Fixing the Issues

6) CD : Deploy Different Server (Dev, QA, UAT, Prod)

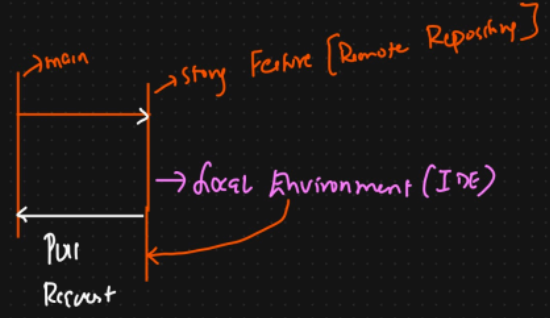
Developer Workflow

Developer A Team → Data Science Project

1) Feature Development



② Push And Pull Request (PR)



CI

Other Team member Review the PR for code quality, style and any other issue

4) Automated CI Pipeline :

PR → Event → Workflow is Triggered

1) Once the PR is opened, A CI pipeline is automatically triggered.
2) This pipeline builds the application and runs all test cases.
3) If pipeline passes, the PR is approved and merged in the main branch

5) Continuous Deployment

1) Upon merging the PR, a CD pipeline is triggered

2) The application is automatically deployed to Dev or Staging Environment.

3) Production Environment

Benefits of a Developer's Workflow

Improved Collaboration: Clear workflows and processes facilitate better collaboration among team members, making it easier to understand who is responsible for what.

Higher Code Quality: Code reviews, automated tests, and CI/CD practices help maintain a high standard of code quality.

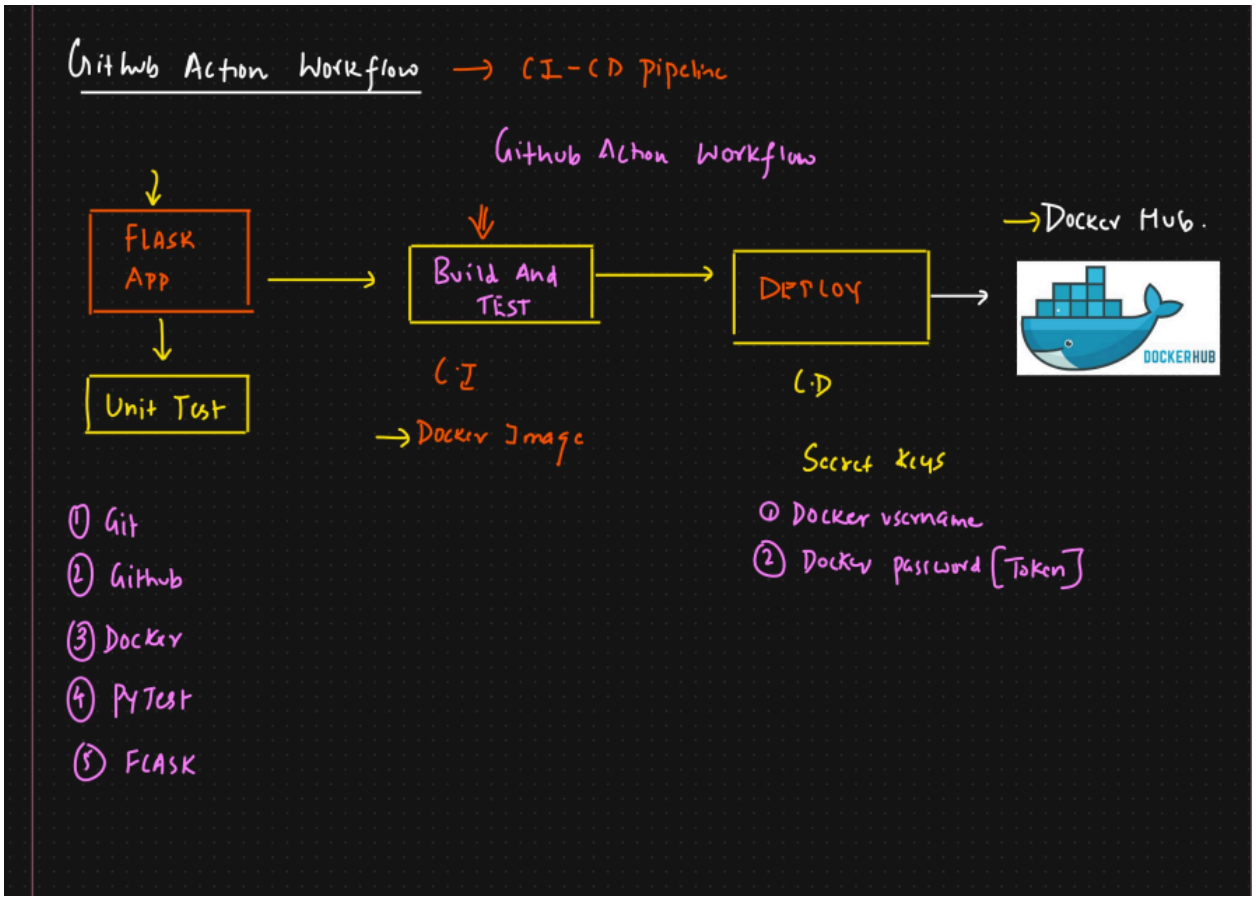
Reduced Errors: Automated testing and deployment reduce the likelihood of human error, ensuring more reliable releases.

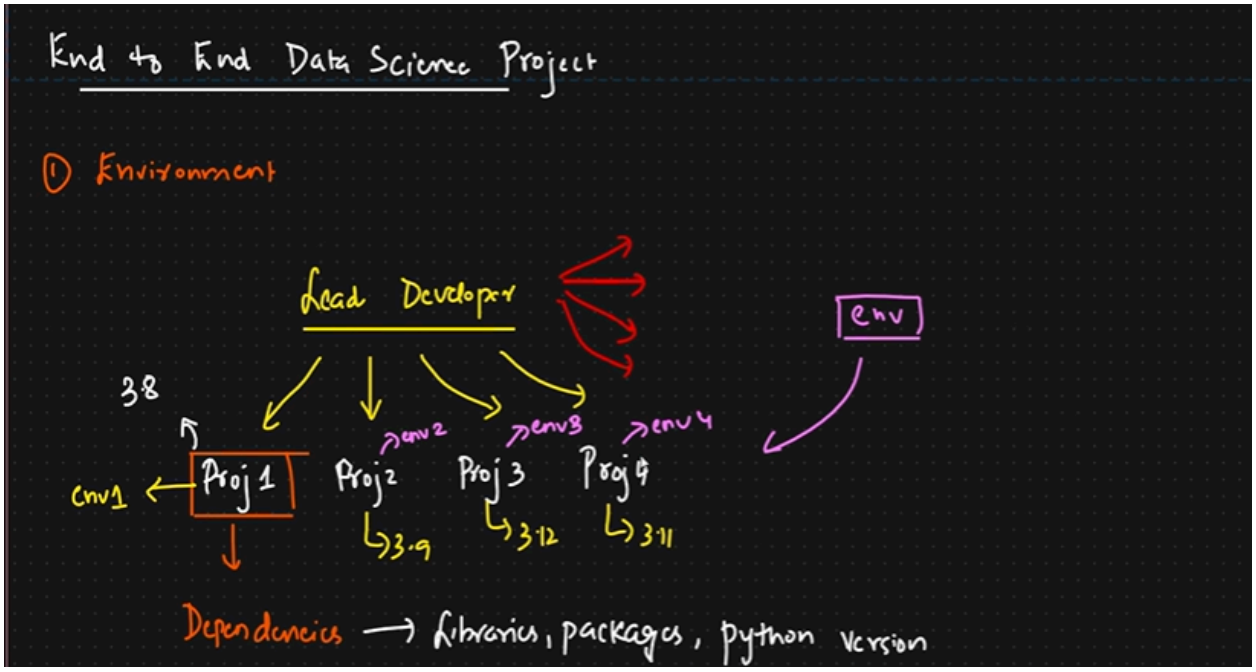
Faster Delivery: Streamlined workflows enable faster development and release cycles, allowing teams to deliver new features and fixes more quickly.

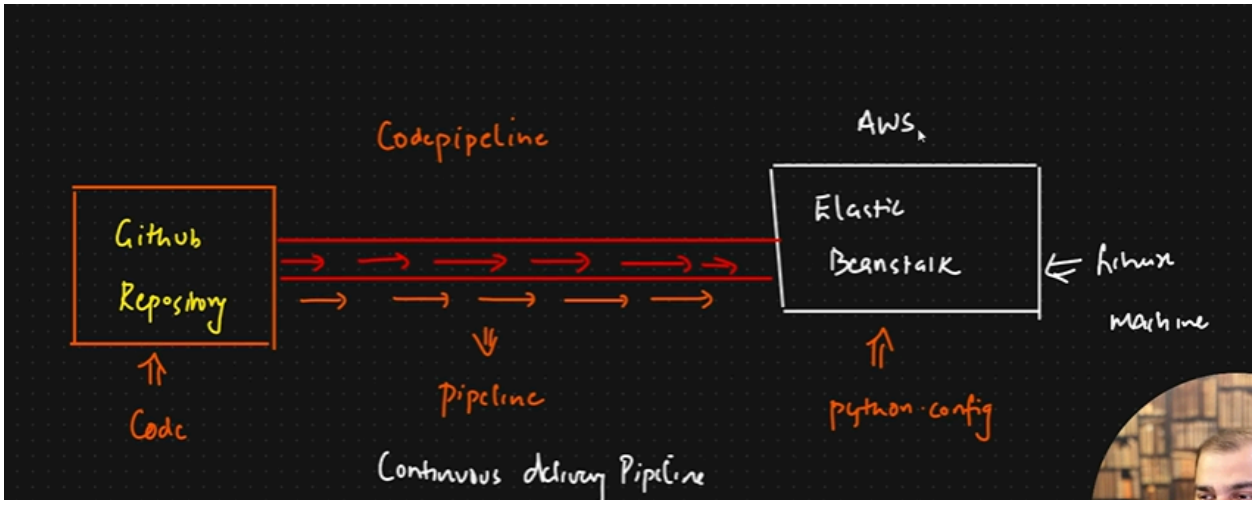
Continuous Feedback Loop: Regular monitoring and feedback help teams quickly identify and address issues, continuously improving the product.

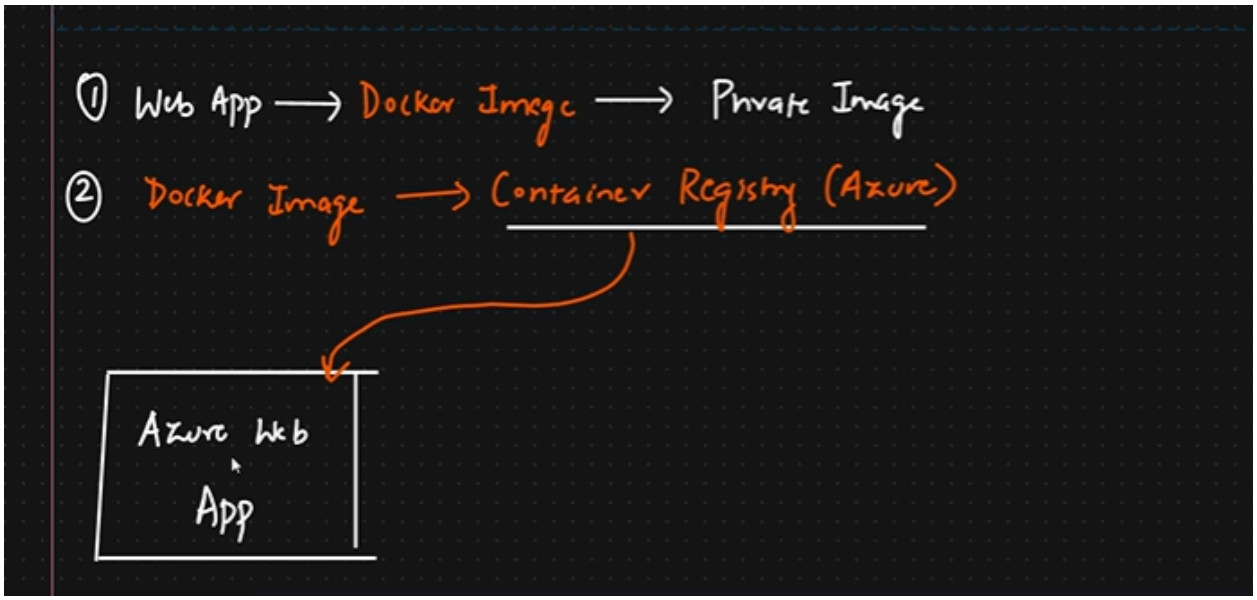
Github Action Workflow

Project : Automate Testing for a Python Project.


Github Action Workflow → CI-CD pipeline
Github Action Workflow
FLASK APP
Unit Test
Build And TEST
C.I
→ Docker Image
DEPLOY
C.D
→ Docker Hub.
DOCKERHUB
Secret keys
① Docker username
② Docker password [Token]
① Git
② Github
③ Docker
④ PyTest
⑤ FLASK

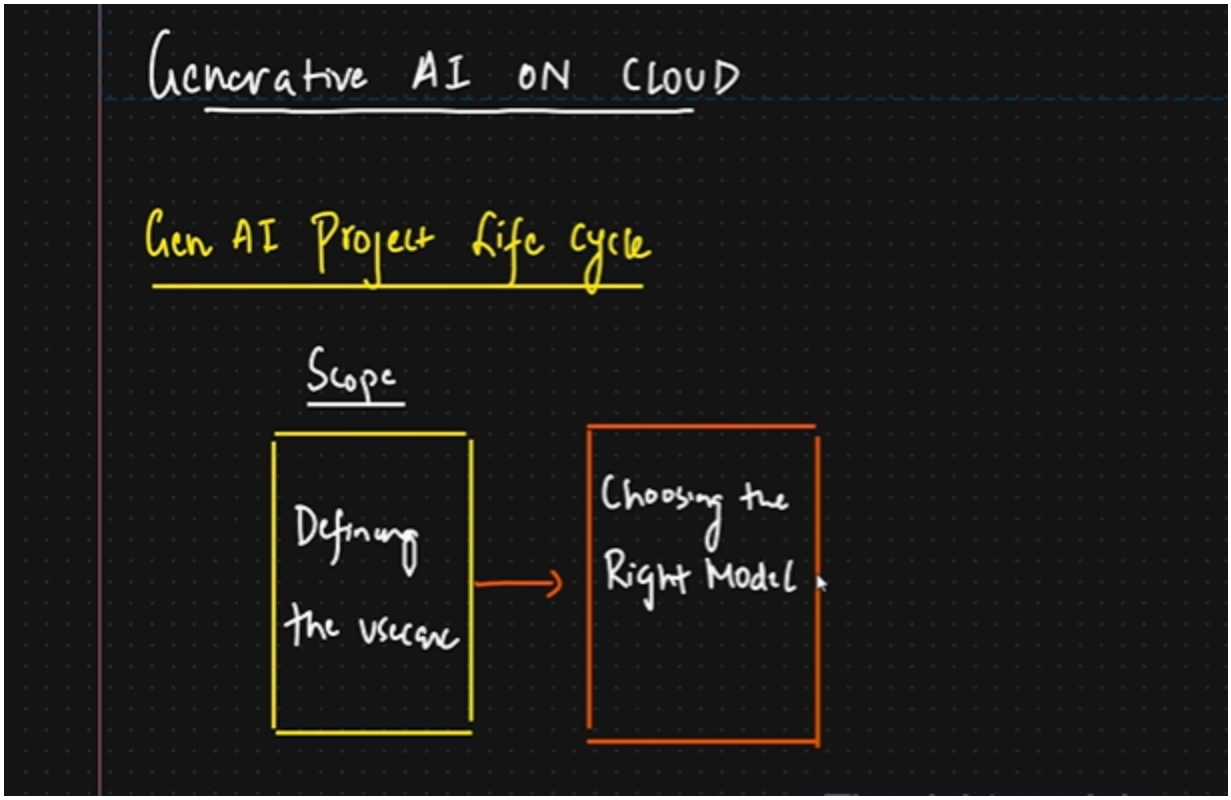

End to End Data Science Project
① Environment
Lead Developer
env
3.8
Proj 1
env1
Proj 2
env2
3.9
Proj 3
env3
3.12
Proj 4
env4
3.11
Dependencies → libraries, packages, python version

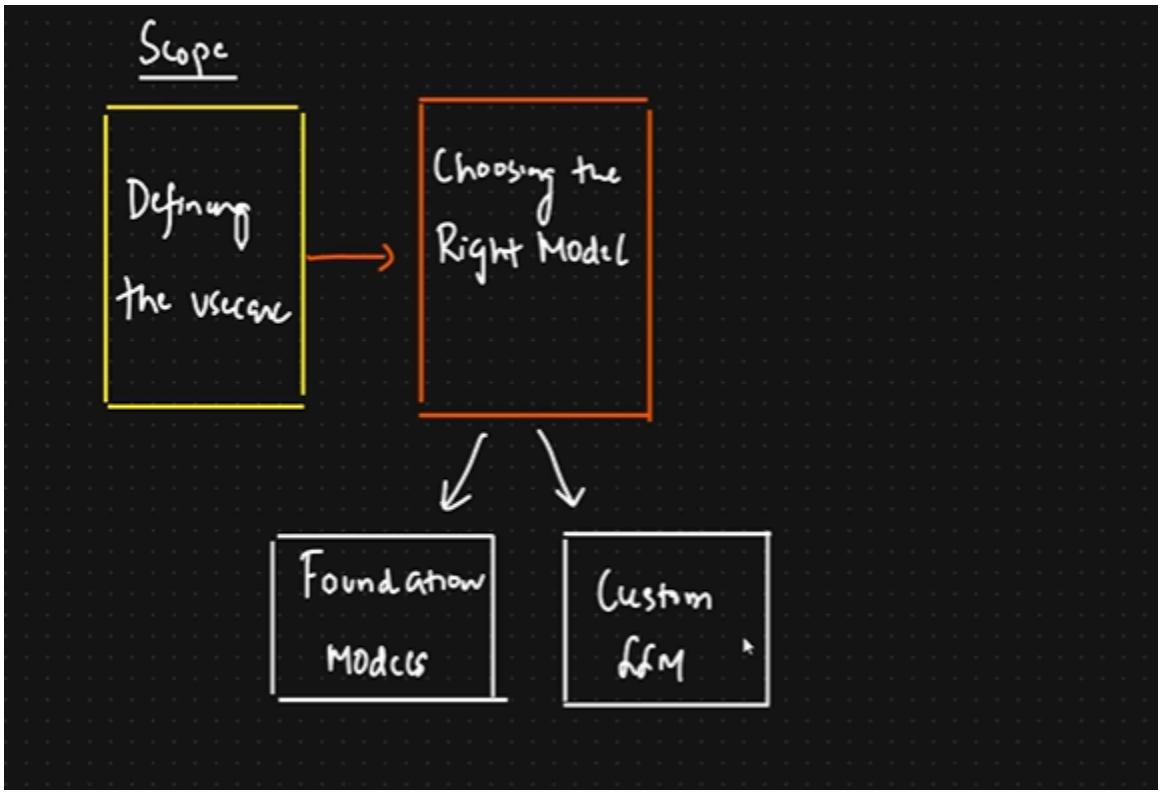

Codepipeline
AWS
Github
Repository
Elastic
Beanstalk
Linux
machine
Pipeline
Codc
python-config
Continuous delivery Pipeline

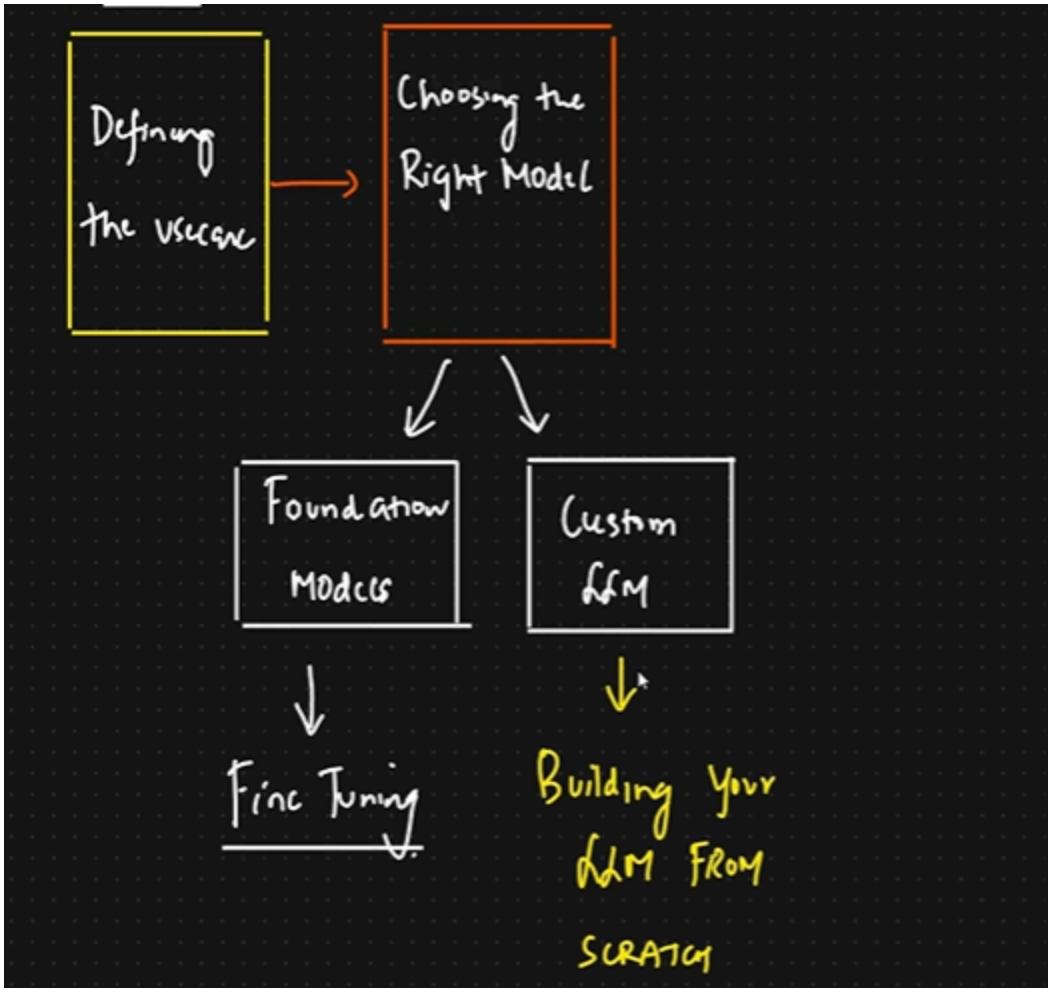

① Web App → Docker Image → Private Image
② Docker Image → Container Registry (Azure)
Azure Web
App

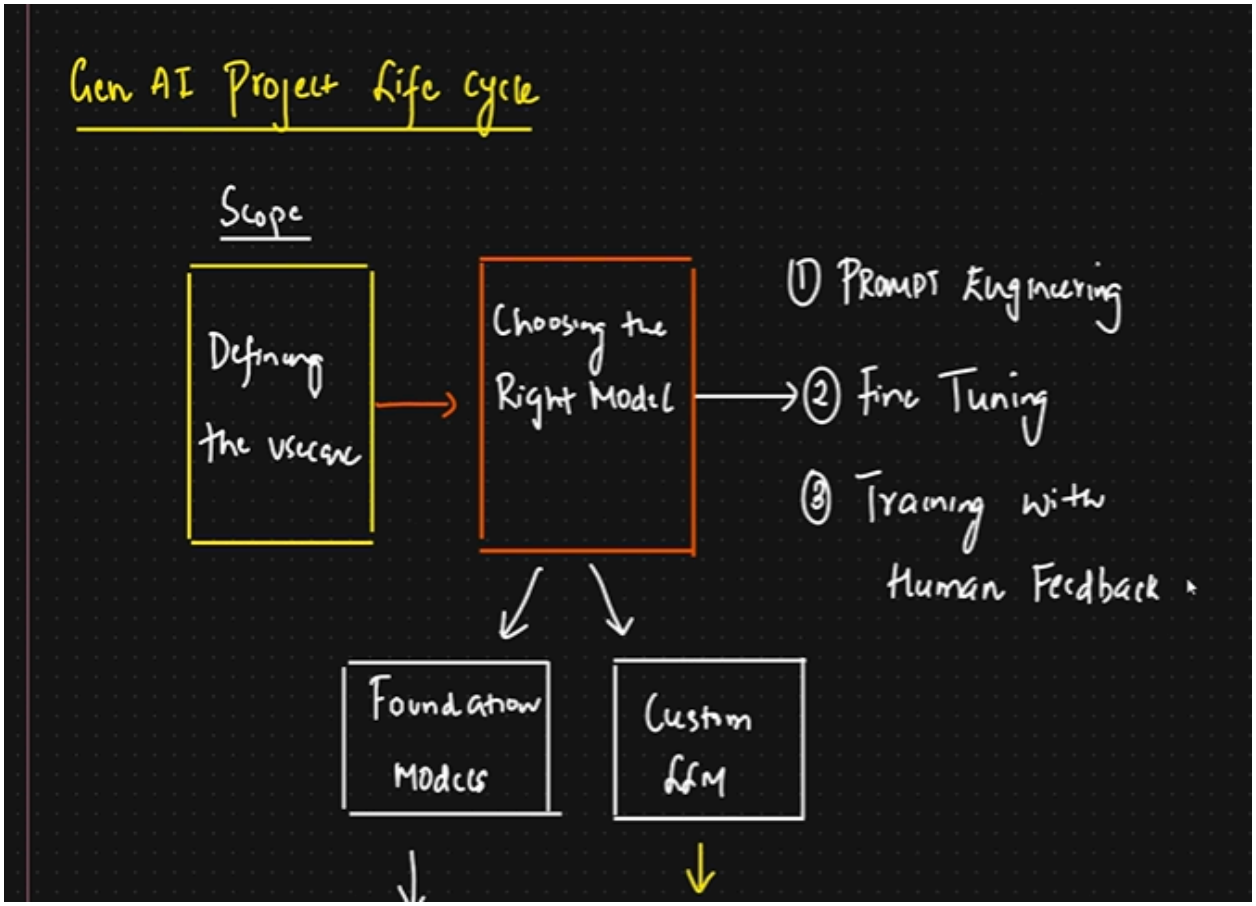
Generative AI on Cloud

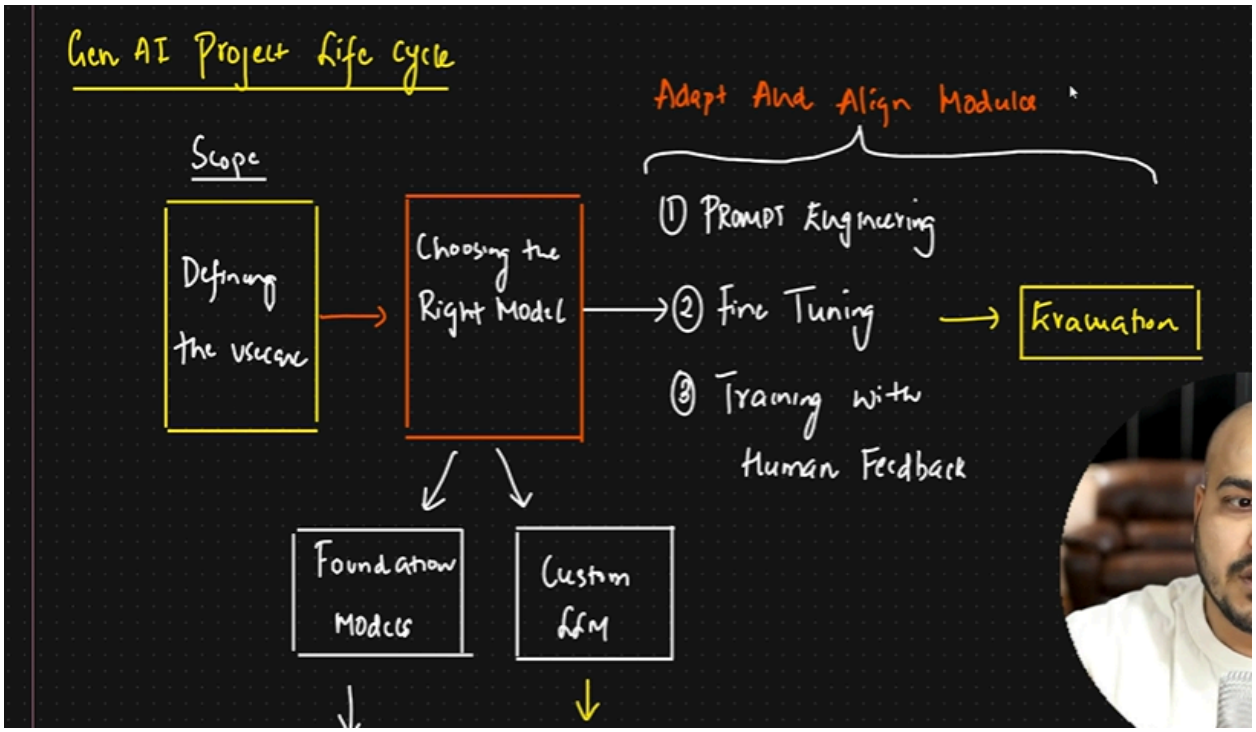
Gen AI Project Life cycle

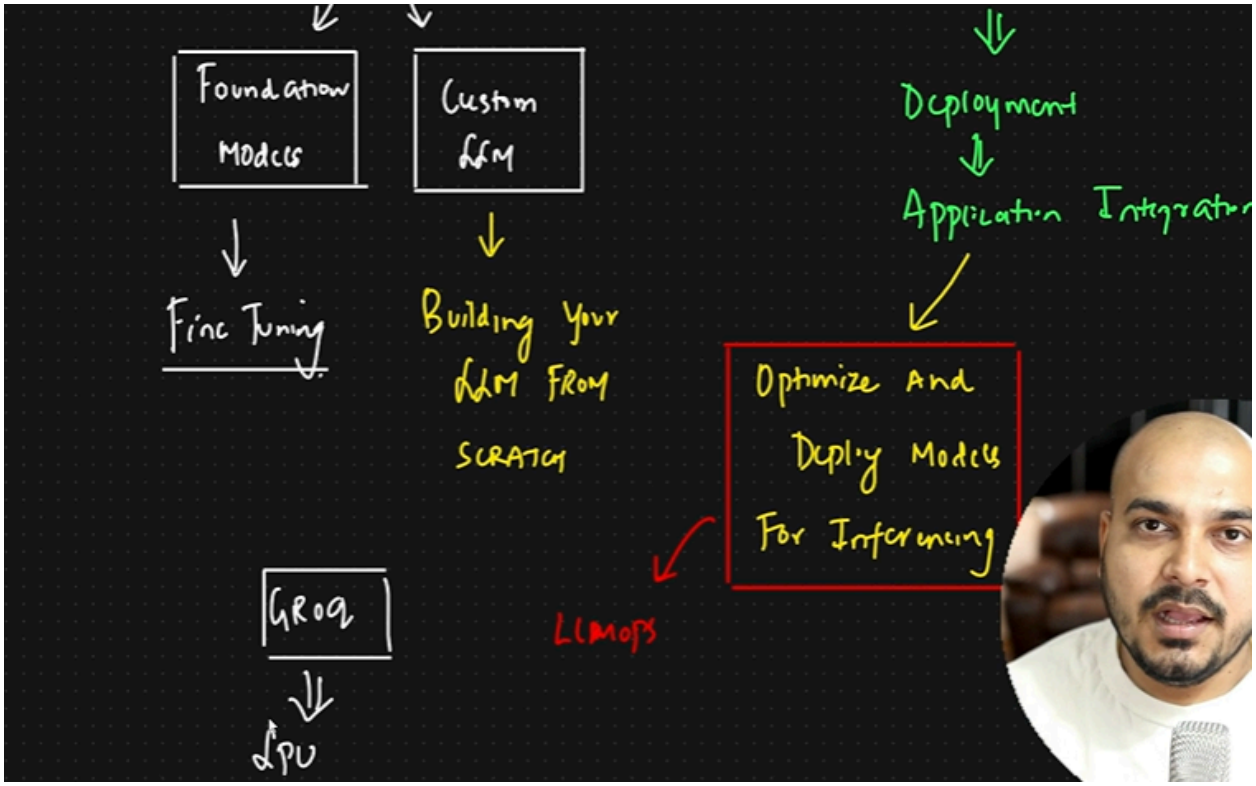


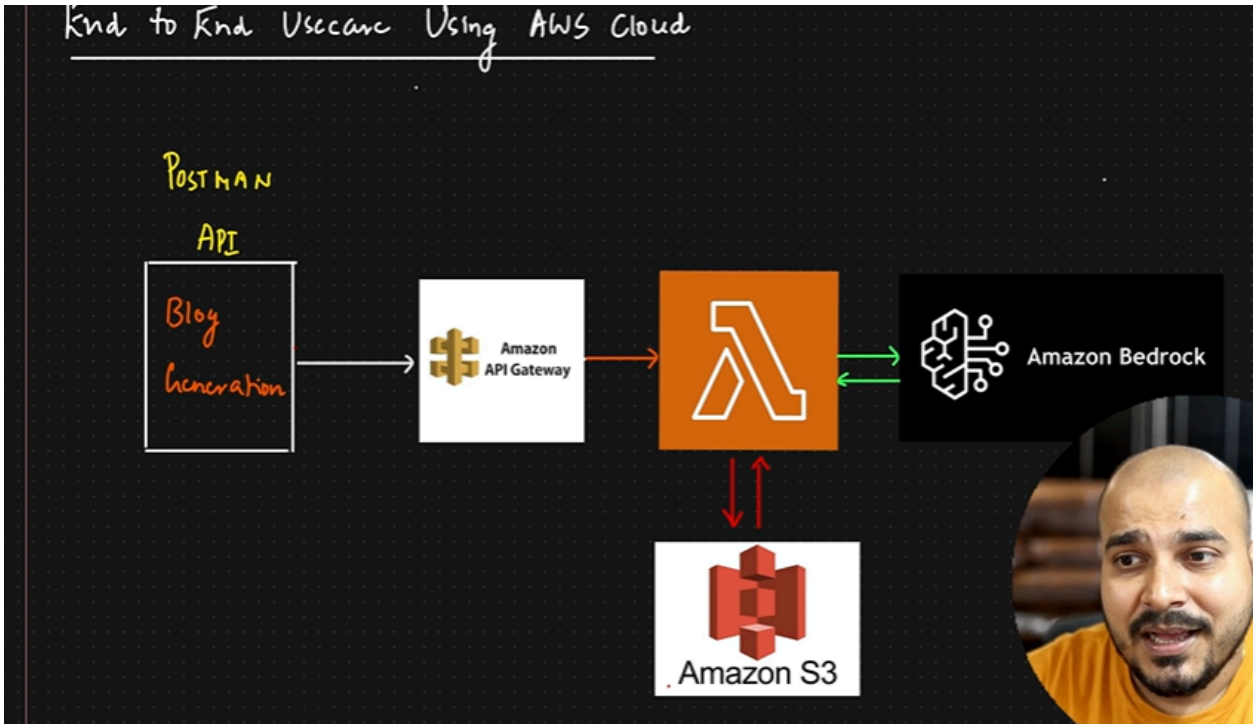



Defining the usecase
Choosing the Right Model
Foundation Models
Custom LLM
Fine Tuning
Building Your LLM FROM SCRATCH


Gen AI Project Life cycle
Scope
Defining the usecase
Choosing the Right Model
① Prompt Engineering
② Fine Tuning
③ Training with Human Feedback
Foundation Models
Custom LLM


Gen AI Project Life cycle
Adapt And Align Modules
Scope
Defining the usecase
Choosing the Right Model
① Prompt Engineering
② Fine Tuning
Evaluation
③ Training with Human Feedback
Foundation Models
Custom LLM


Foundation Models
Custom LLM
Deployment
Application Integration
Fine Tuning
Building Your LLM From SCRATCH
Optimize And Deploy Models For Inferencing
LLMOps
GROQ
LPU


End to End Usecase Using AWS Cloud
POSTMAN
API
Blog Generation
Amazon API Gateway
Amazon Bedrock
Amazon S3