2024

Production ML System Design

From System Design to Deployment

Alexander Guschin
Mikhail Rozhkov

OUTLINE

1. System Architecture
2. Deployment and Serving
3. Data Engineering
4. Model Development
5. Testing, Monitoring, and Scalability
6. Infrastructure

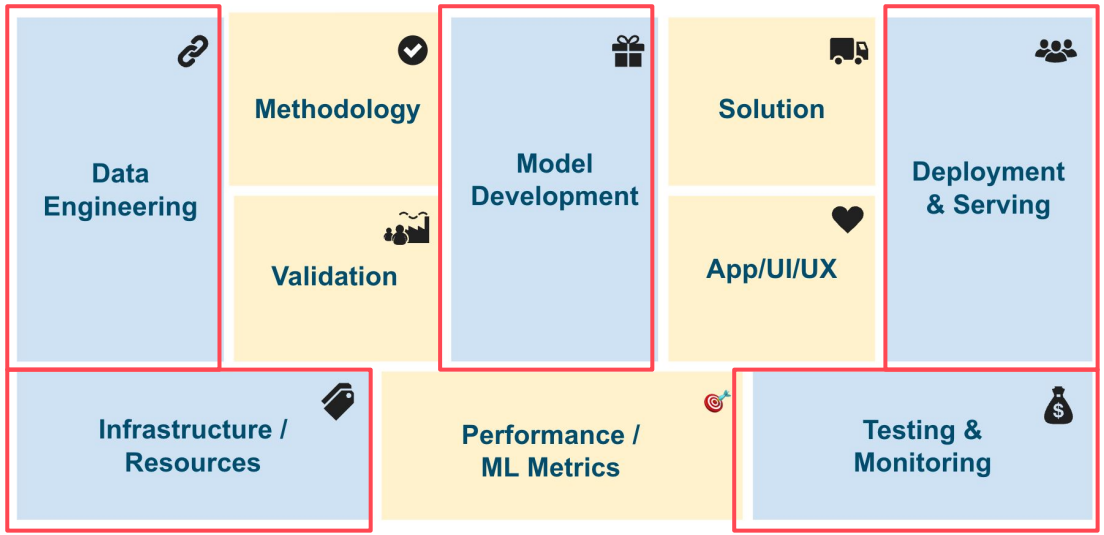
5.x - Production ML System Design

Purpose

- Frame the ML solution

Guiding questions:

- How does the ML could help?



MODULE DETAILS

Goals

- Learn to design scalable and maintainable ML systems
- Understand the challenges of deploying ML models in production
- Develop skills to monitor and maintain ML systems over time

Learning Outcomes

- Design production-ready ML architectures
- Develop deployment and serving strategies
- Implement effective data engineering pipelines
- Create comprehensive testing and monitoring plans

System Architecture

Production ML System Design

> Guide: 5.1 - High-level design

5.1 - High-level design

What are the key components of the ML System?

Purpose

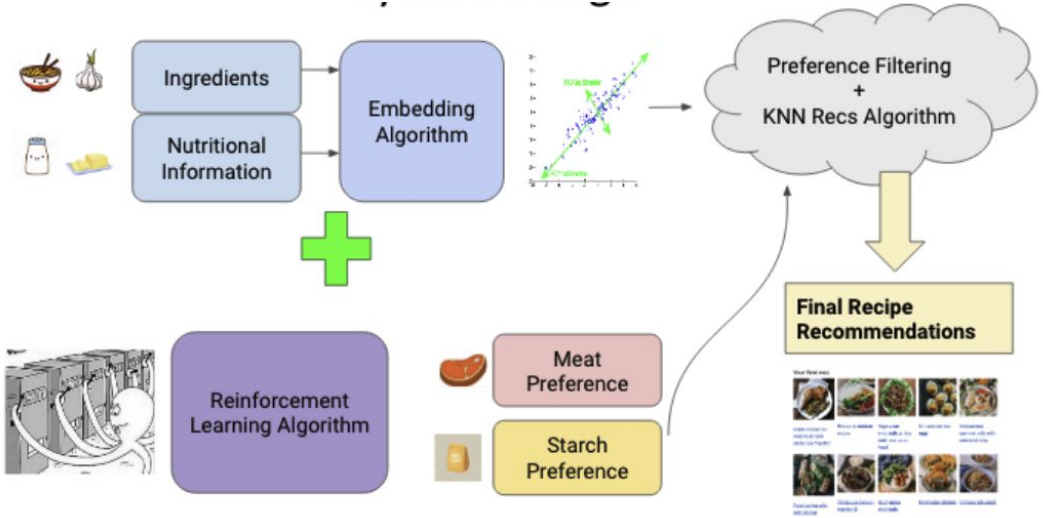
- To ensure all technical stakeholders understand how the system will be built and how data will flow through it

Guiding questions:

- How will the ML system integrate with our current infrastructure?
- What are the key components of our data processing pipeline?
- How will we ensure efficient data flow through the system?

Case: recipes recommendation

[screenshot of our evaluation page]

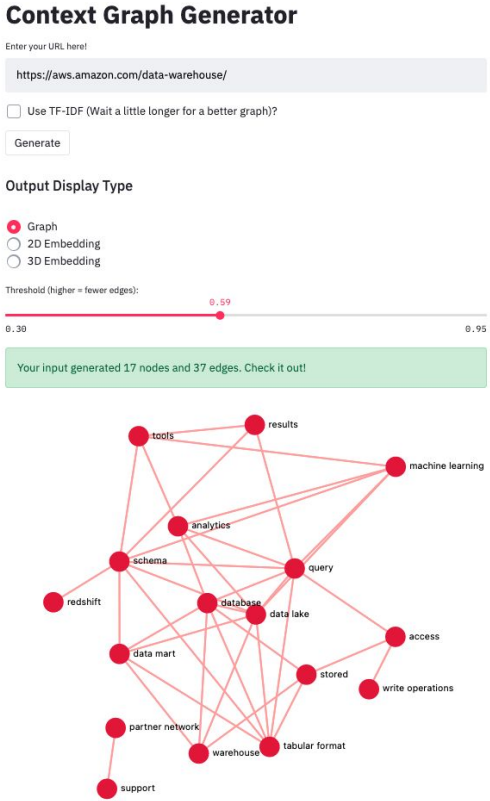


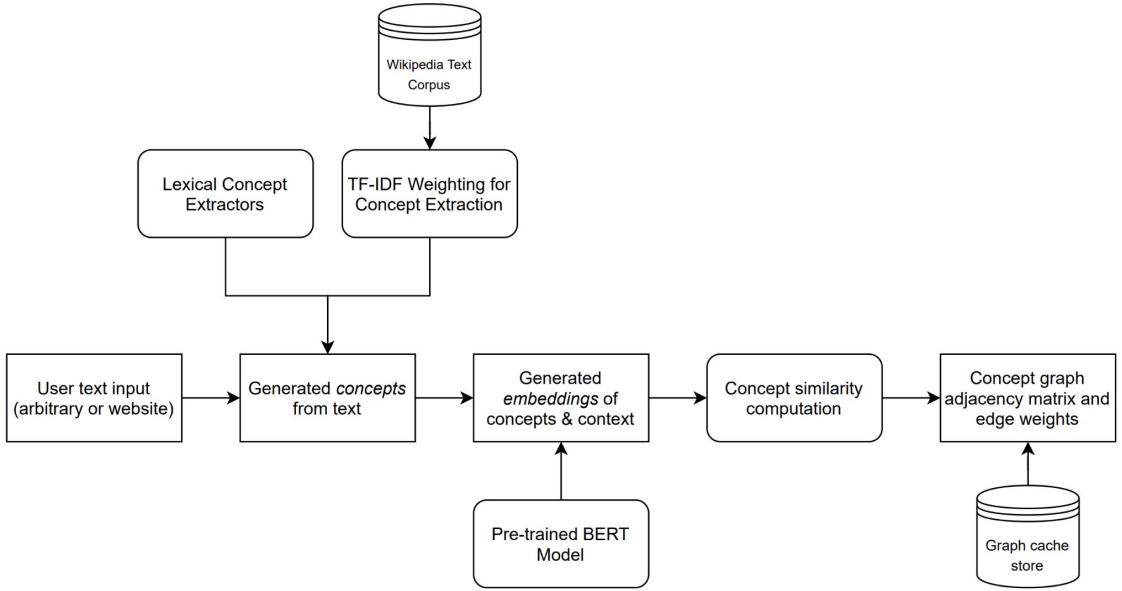
[System diagram to show the moving components of our application]

Project: Tender Matching People to Recipes

Sourel: https://stanford-cs329s.github.io/reports/tender-recipe-recommendations/

Case: article graph summarization





Project: Building a Context Graph Generator

Sourel: https://stanford-cs329s.github.io/reports/context-graph-generator/

Exercise: High-level design

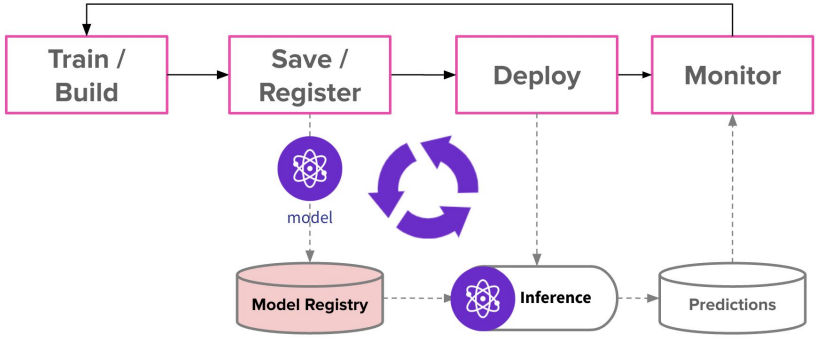
How to ensure generalization and robustness?

Group task:

- Create a draft of the ML System Architecture. Complete the Production ML System Design section:
  - 5.1 - High-level design
- 5 min

Key points:

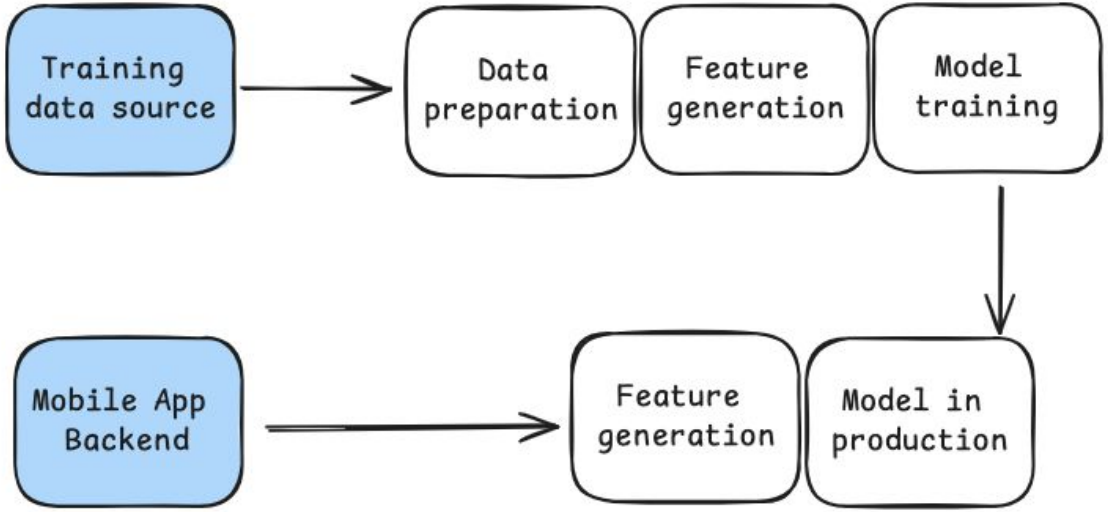
- System architecture diagram regardless of tools/services you use
- Data flow and processing pipeline
- Integration with existing infrastructure



Practice

5.1 - High-level design

What are the key components of the ML System?

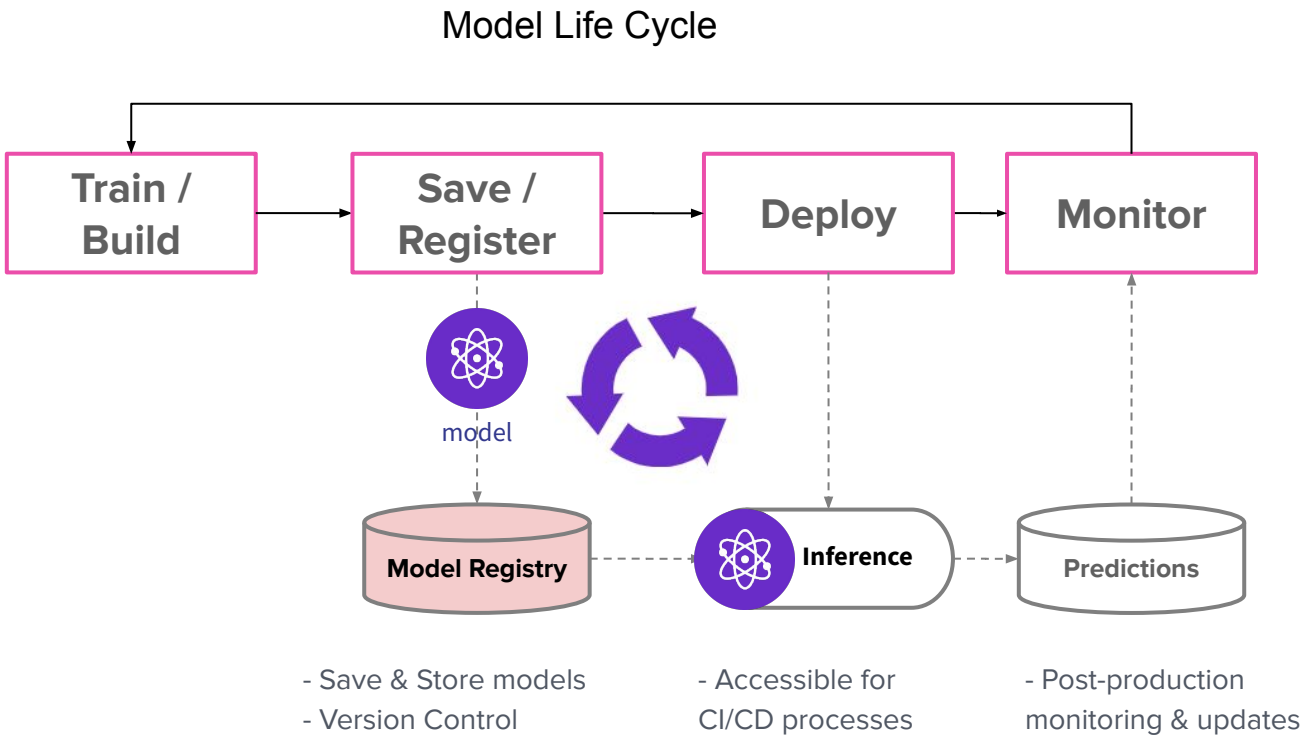


Example

Deployment and Serving

Production ML System Design
> Guide: 5.2 - Deployment and Serving

Deployment defines details of your system

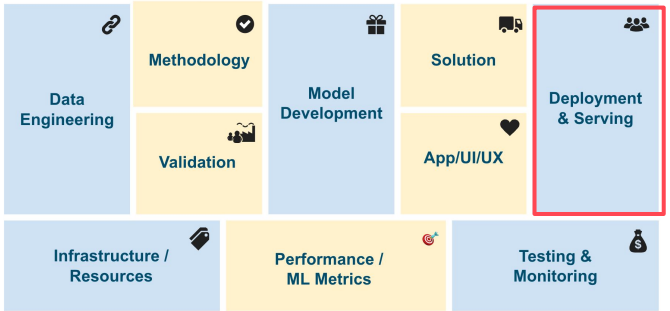


5.2 – Deployment and Serving

How do we deploy and serve the model?

Purpose

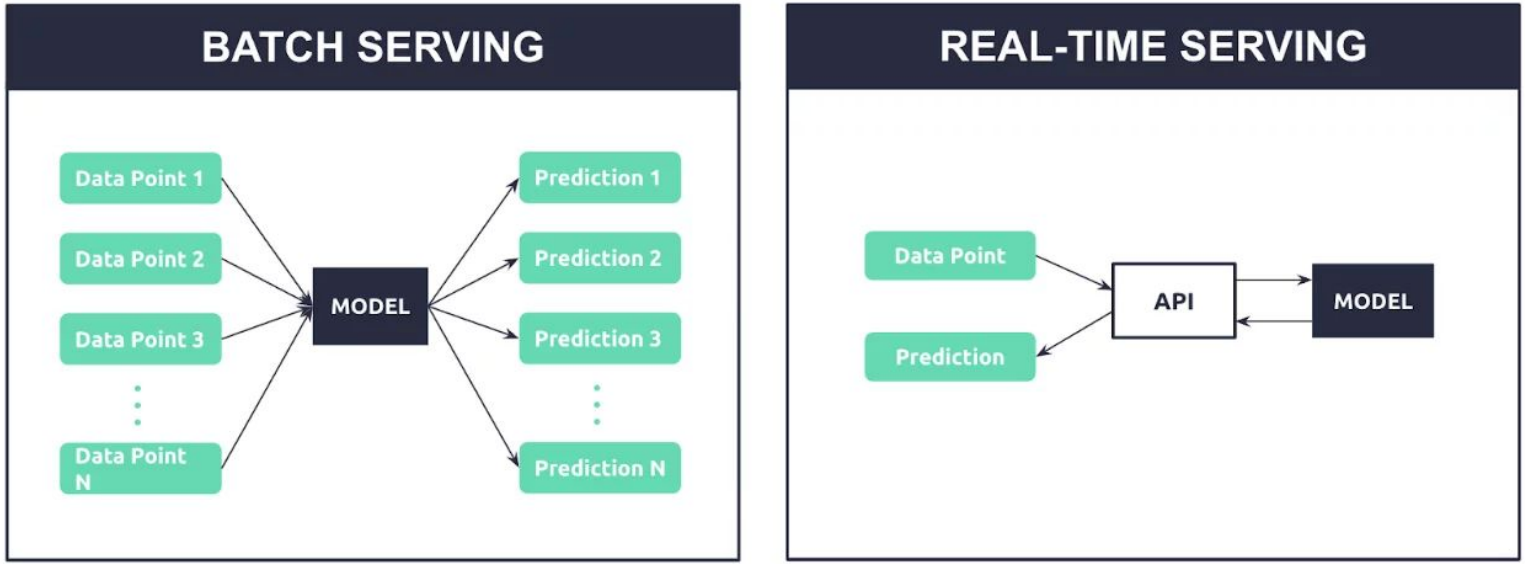
- To ensure smooth deployment and reliable serving of model predictions.



Guiding questions:

- Do we want to perform batch (offline) or real-time (online) inference? What tools should we use?
- What infrastructure is needed to serve the ML service?
- How will we deploy the model with minimal disruption? Do we need human checks? How do we test the models before deployment to be sure they doesn't break prod?
- Do we need to support online A/B testing, canary deployment, etc?

ML inference: batch vs. online



Source: https://www.iguazio.com/glossary/model-serving-pipeline/

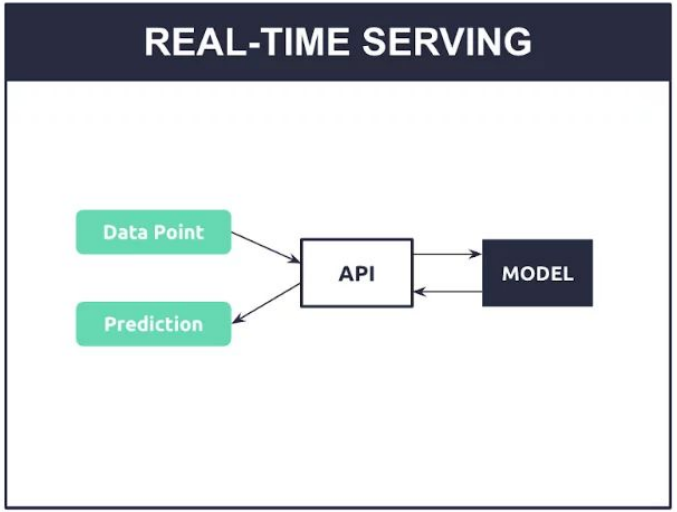
Online Serving

Typical requirements:

- Get predictions on demand
- Latency < 1 sec
- REST API

Applications:

- Chat bot Assistants
- eCommerce Recommendation System



FastAPI Flask mlflow

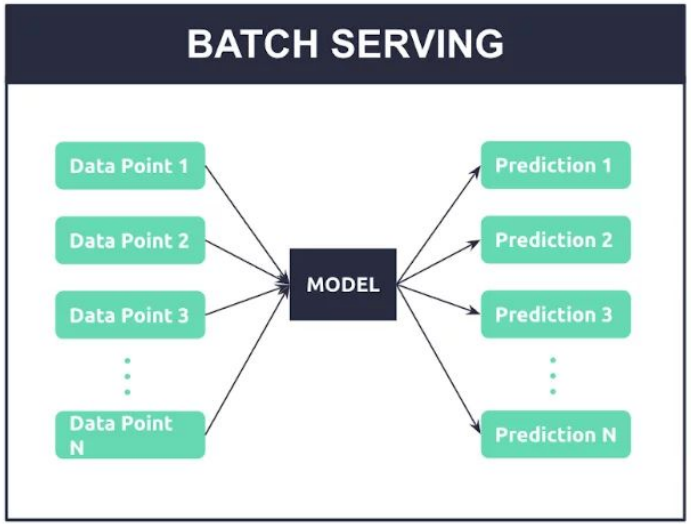
Batch Serving

Typical requirements:

- Get predictions for a large dataset at once
- Latency mins-hours
- Scheduled
- HDFS, S3, Hive, files

Applications:

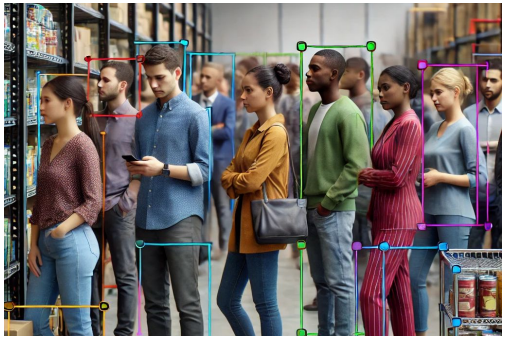
- Sales forecast
- Churn prediction



Apache Airflow PREFECT DVC

Exercise: Select correct deployment method?

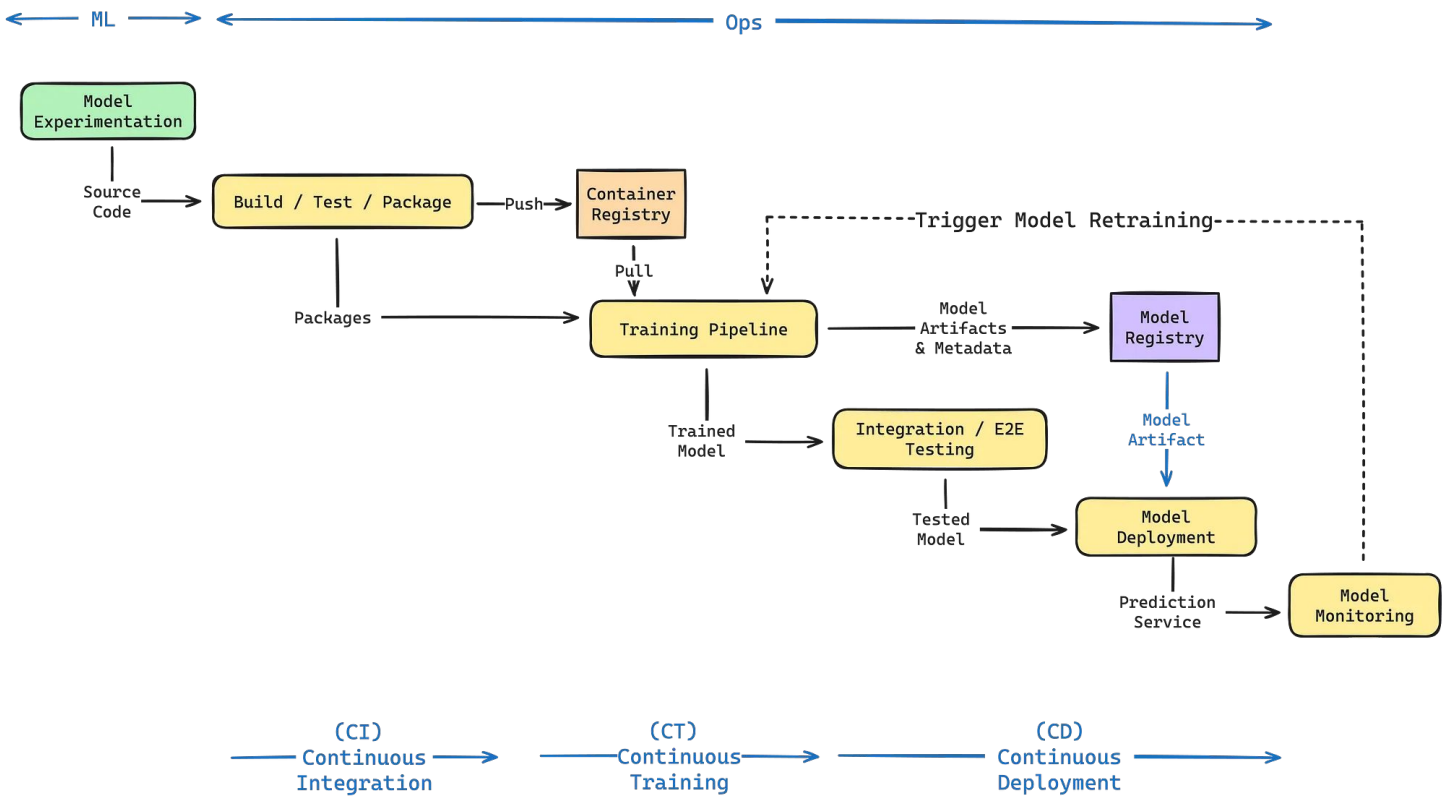
New pizza waiting time prediction



Shop queue detection [CV]

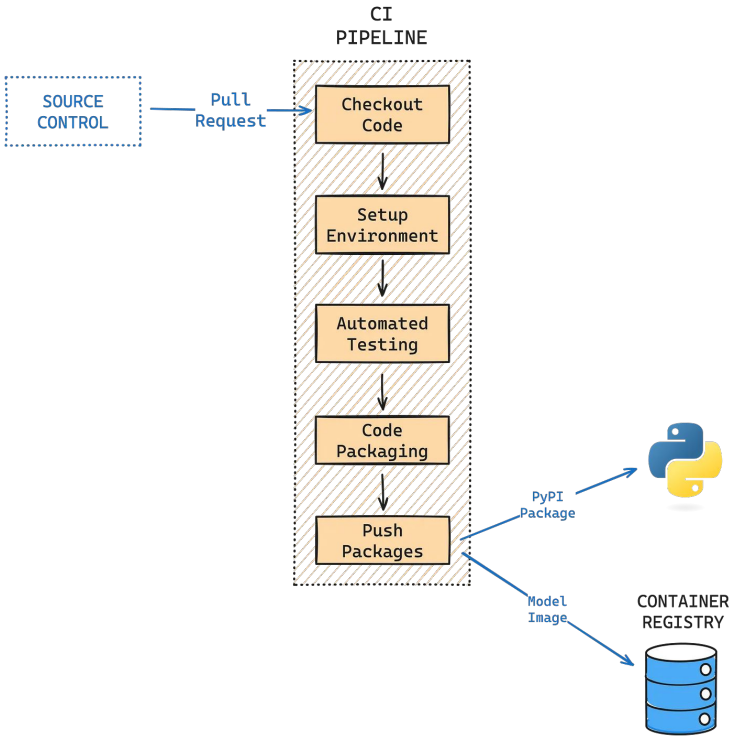
NY Taxi trip duration prediction

CI/CD automates process of Model Deployment



Source: [CI/CD for Machine Learning in 2024: Best Practices to Build, Train, and Deploy]

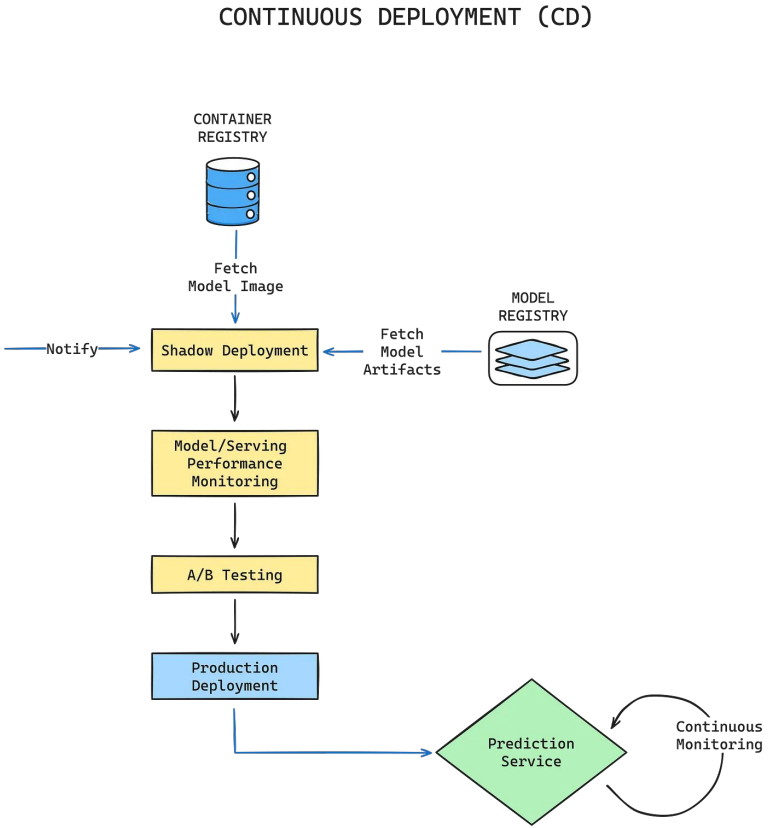
Continuous Integration (CI)



- "Integrates" a new code into the existing code base
- Ensures the code quality and functionality, preparing the codebase for subsequent phases of the ML lifecycle

Source: CI/CD for Machine Learning in 2024: Best Practices to Build, Train, and Deploy

Continuous Deployment (CD)



- the systematic and automated retraining of ML modelsc

Source: CI/CD for Machine Learning in 2024: Best Practices to Build, Train, and Deploy

Exercise: Deployment and Serving

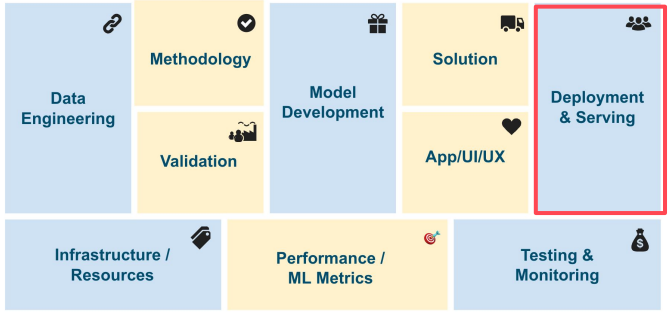
How do we deploy and serve the model?

Group task:

- Complete the Production ML System Design section:
  - 5.2 - Deployment and Serving
- 5 min

Key points:

- Inference Type
- CI/CD approach
- Serving Infrastructure: tools your are going to use for deployment and serving
- Deployment Methodology: important things to consider about deploy



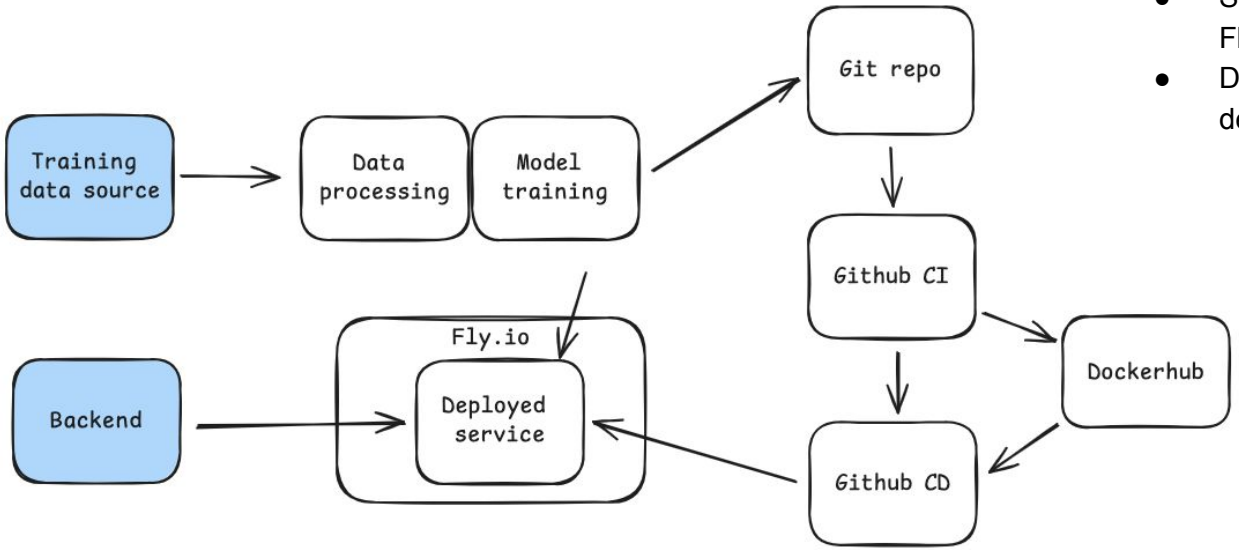
Practice

5.2 – Deployment and Serving

How do we deploy and serve the model?

Overview:

- The deployment involves a CI/CD pipeline, Docker Compose for containerization, and a shadow deployment mode for testing.



Key points:

- Inference Type: Real-time.
- CI/CD: Automated deployment pipeline in GitHub Actions (GHA)
- Serving Infrastructure: FastAPI, Docker, Fly.io
- Deployment Methodology: Shadow deployment.

Example

ML System Design - Production: EasyRide Taxi

Data Engineering

-
-

Methodology

- Problem Type: Regression.
- Approach: Supervised learning.
- Metric: MAPE.
- Baseline: Current predictions.

Validation

- Validation Methodology: Shadow deployment, A/B testing.
- Pilot Scope: 1 week
- Success Criteria: Lower MAPE, higher booking rates.

Model Development

-
-

Solution

- Target: trip duration
- Format: Duration in minutes
- Components: Taxi App, Data Ingestion, ML Solution, Backend

App/UI/UX

- UI: ETA, trip duration and cost/earnings.
- UX: Requesting a ride, viewing the predicted cost, booking the ride.

Deployment & Serving

Inference Type: Real-time.

CI/CD: Automated deployment pipeline in GitHub CI

Serving Infrastructure: FastAPI, Docker, Fly.io, PosgreSQL

Deployment Methodology: Shadow deployment, canary releases.

Infrastructure / Resources

-
-

Performance / ML Metrics

- Prediction accuracy (MAPE) <= 15%
- Prediction latency < 500ms
- Business Metrics: Booking rate, Revenue Increase
- Timeline: Daily metric evaluation.

Testing & Monitoring

-
-

Data Engineering

Production ML System Design
> Guide: 5.3 - Data Engineering

5.3 - Data Engineering

How do we ingest and store data?

Purpose

- To ensure reliable, efficient data processing and feature engineering in production.

Guiding questions:

- How will we handle data ingestion and storage?
- What ETL processes are needed to prepare data for the model?
- How will we track data versions and lineage?

Exercise: Data Engineering

How do we ingest and store data?

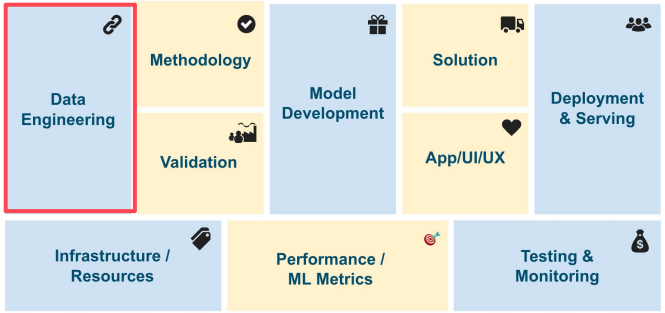
Group task:

- Complete the Production ML System Design section:
  - 5.3 - Data Engineering
- 5 min

Key points:

- Data ingestion and storage solutions
- ETL processes (data pipelines)
- Data versioning and lineage tracking
- Feature Stores (Data Marts)

Practice



5.3 - Data Engineering

How do we ingest and store data?

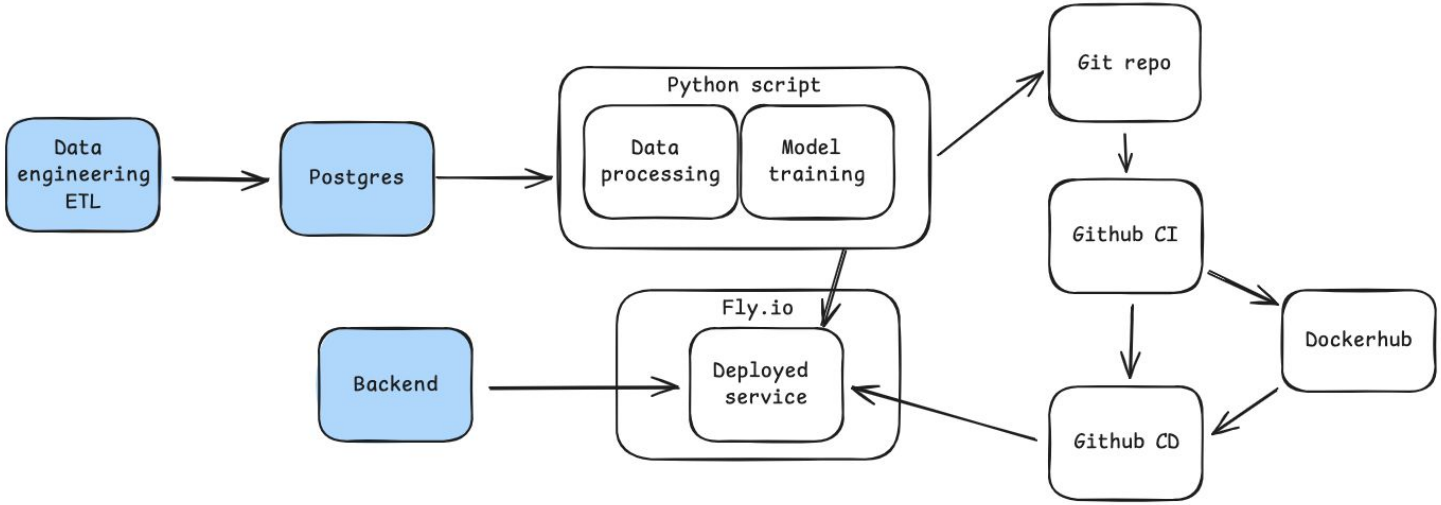
Overview:

- Data is ingested from Postgres, processed via ETL pipelines, and features are stored in a feature store. Data lineage and versioning are tracked.

Key points:

- Data Ingestion: From Postgres every 30 minutes.
- Data Versioning: Ensuring reproducibility.

Example



ML System Design - Production: EasyRide Taxi

Data Engineering

Data Ingestion: From Postgres every 30 minutes.

ETL Processes: Data cleaning, feature engineering.

Data Versioning: Ensuring reproducibility.

Methodology

Problem Type: Regression.

Approach: Supervised learning.

Metric: MAPE.

Baseline: Current predictions.

Validation

Validation Methodology: Shadow deployment, A/B testing.

Pilot Scope: 1 week

Success Criteria: Lower MAPE, higher booking rates.

Model Development

Solution

Target: trip duration

Format: Duration in minutes

Components: Taxi App, Data Ingestion, ML Solution, Backend

App/UI/UX

UI: ETA, trip duration and cost/earnings.

UX: Requesting a ride, viewing the predicted cost, booking the ride.

Deployment & Serving

Inference Type: Real-time.

CI/CD: Automated deployment pipeline in GitHub CI

Serving Infrastructure: FastAPI, Docker, Fly.io, PosgreSQL

Deployment Methodology: Shadow deployment, canary releases.

Infrastructure / Resources

Performance / ML Metrics

Prediction accuracy (MAPE) <= 15%

Prediction latency < 500ms

Business Metrics: Booking rate, Revenue Increase

Timeline: Daily metric evaluation.

Testing & Monitoring

Model Development

Production ML System Design
> Guide: 5.4 - Model Development Lifecycle

5.4 – Model Development Lifecycle

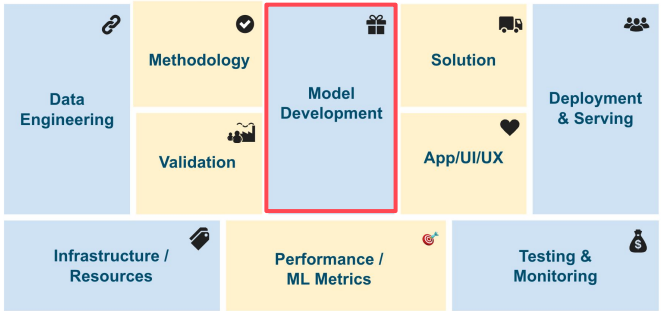
How to ensure reproducibility and reliability?

Purpose

- To ensure continuous improvement and reliable updates to the ML system..

Guiding questions:

- How to organize training in a pipeline?
- How often to re-train? When to do that?
- How will we manage model versions? Where we store them? How do we start a deployment process?



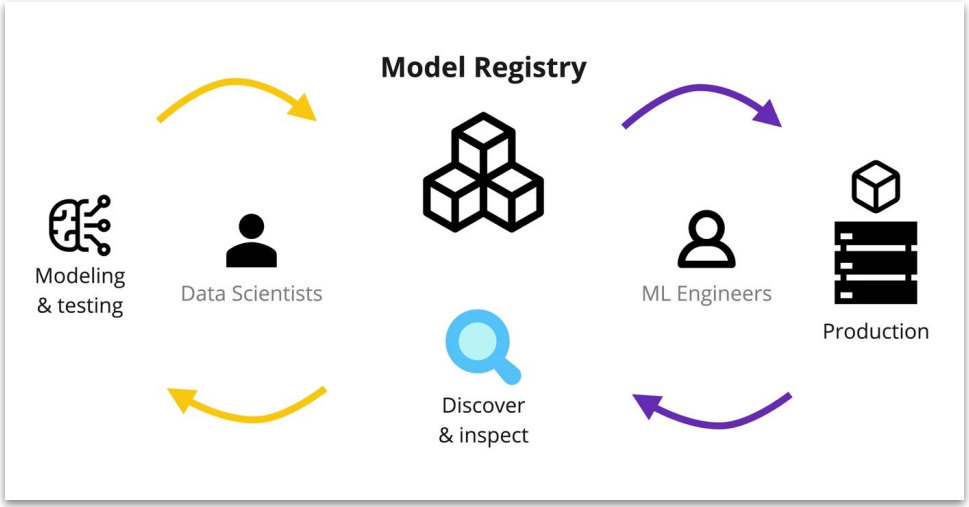
Requirements for efficient Model Development

- Model versioning & registering
- Pipelines Automation
- Reproducibility
- Metrics Tracking
- Continuous Training (CT)

What is a Model Registry?

Model Registry is a centralized repository that manages and stores machine learning models:

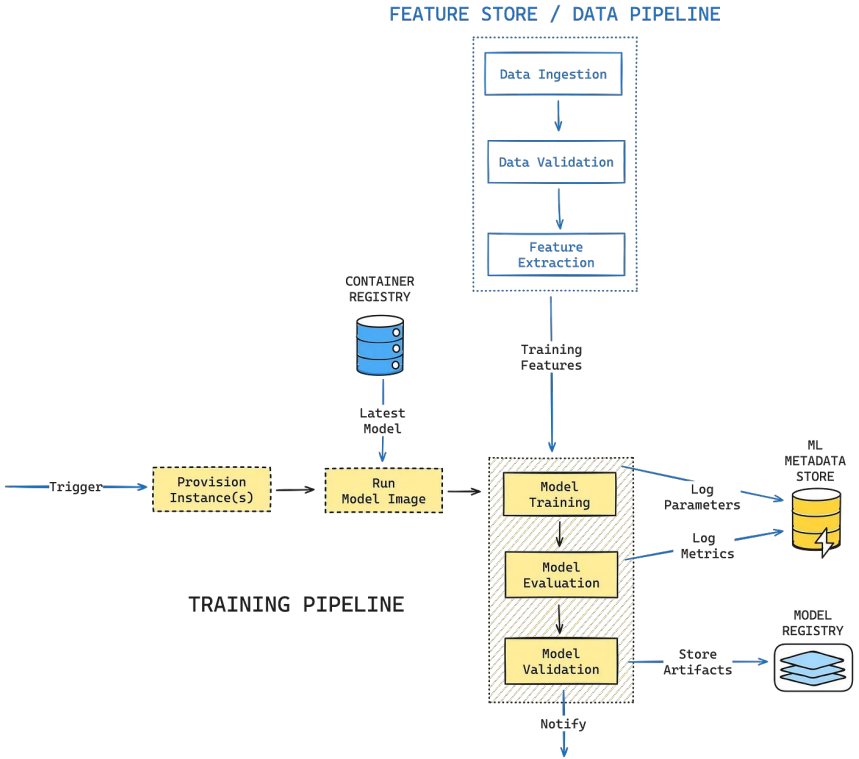
- to discover
- to track changes to models
- to version
- to deploy
- to share & collaborate



MLOps from modeling to production
Source: https://dvc.org/doc/use-cases/model-registry

mlflow Model Registry DVC

Continuous Training (CT)

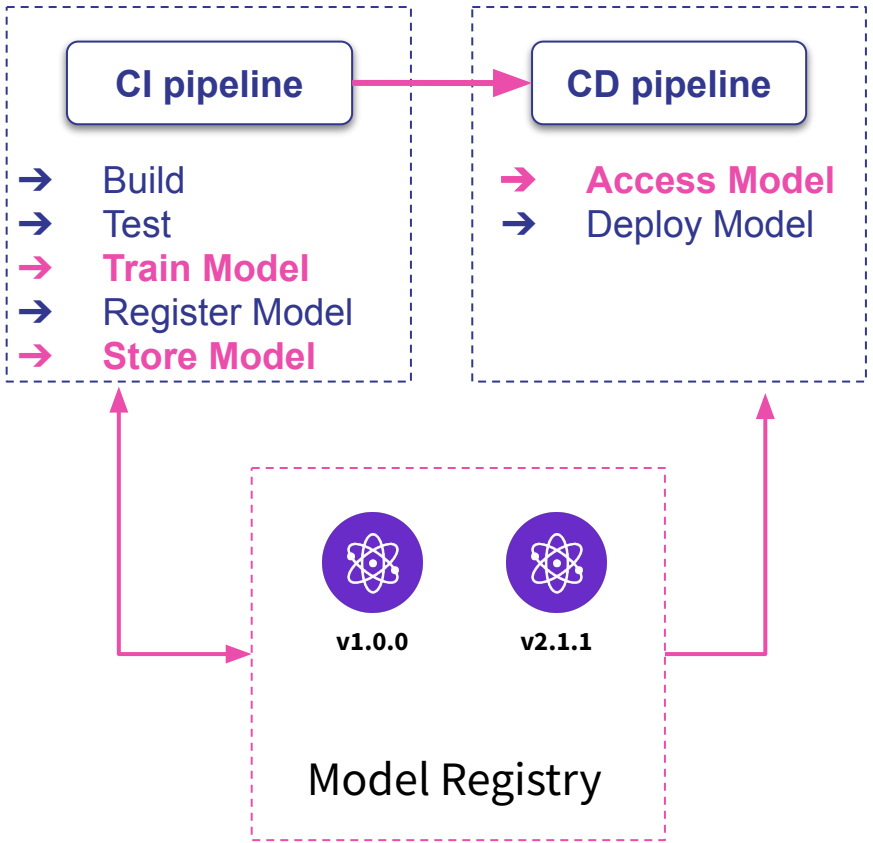


- the systematic and automated retraining of ML modelsc

Source: [CI/CD for Machine Learning in 2024: Best Practices to Build, Train, and Deploy]

Automated CI/CD pipelines: train & register

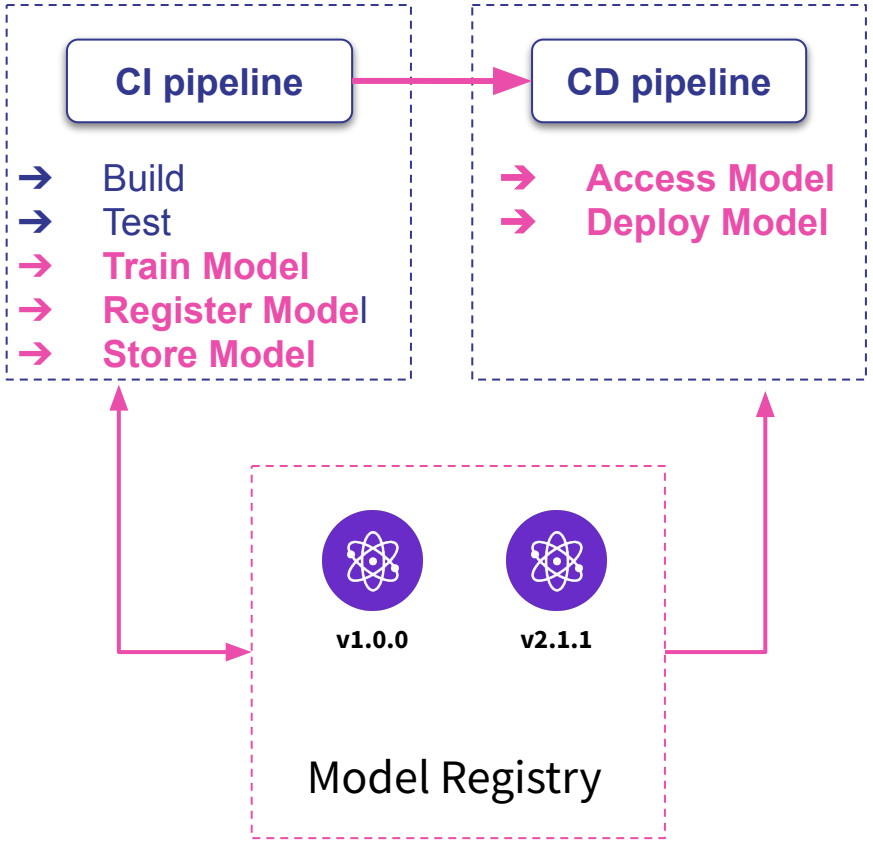
```yaml
name: Train
...
jobs:
  deploy-runner:
    runs-on: ubuntu-latest
    steps:
        ...
        run: |
          cml runner \
              --cloud-spot \
              --cloud=aws \
              --cloud-region=us-east-1 \
              --cloud-type=g4dn.xlarge \
              --labels=cml-runner
  train-model:
    needs: deploy-runner
    runs-on: [self-hosted, cml-runner]
    container:
      image: iterativeai/cml:0-dvc2-base1
      options: --gpus all
    environment: cloud
    steps:
      - name: dvc-repro-cml
        run: |
          dvc pull
          dvc exp run
          dvc push
          ...
          # Create CML report
          echo "## Metrics" >> report.md
          dvc metrics show --md >> report.md
          cml send-comment --pr --update report.md
```



Automated CI/CD pipelines: deploy

```yaml
name: Act on artifact registrations and promotions
on:
  push:
    tags:
      - "*"

jobs:
  act:
    name: Figure out what was registered/promoted and act on it
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: "Model Registry scenarios"
      id: gto
      uses: iterative/gto-action@v1
    # ...

    - name: "Publish: on registering a new version"
      if: steps.gto.outputs.event == 'registration'
      run: |
        echo "Version '${{ steps.gto.outputs.version }}' "
        echo "Model '${{ steps.gto.outputs.name }}' "
        echo "Run command to publish it!"

    - name: "Deploy: on assigning a new stage)"
      if: steps.gto.outputs.event == 'assignment'
      run: |
        echo "Model '${{ steps.gto.outputs.name }}' of version '${{
        steps.gto.outputs.version }}' promoted to stage '${{ steps.
        gto.outputs.stage }}'"

        echo "Run command to deploy it!"
```

Example: https://github.com/iterative/example-gto/tree/main



Exercise: Model Development

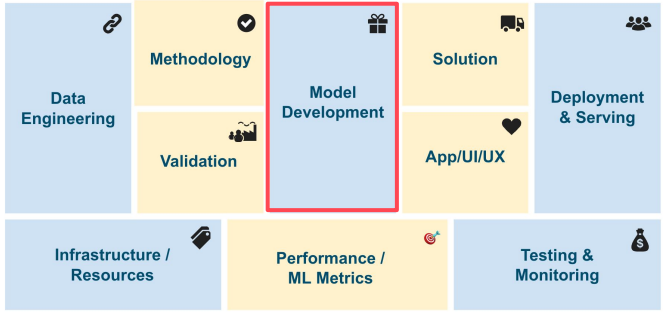
How do we ingest and store data?

Group task:

- Complete the Production ML System Design section:
  - 5.4 - Model Development Lifecycle
- 5 min

Key points:

- Requirements for automation, reproducibility, reliability
- Model versioning
- Model updating and retraining process
- Model Registry



Practice

5.4 – Model Development Lifecycle

How do we ingest and store data?

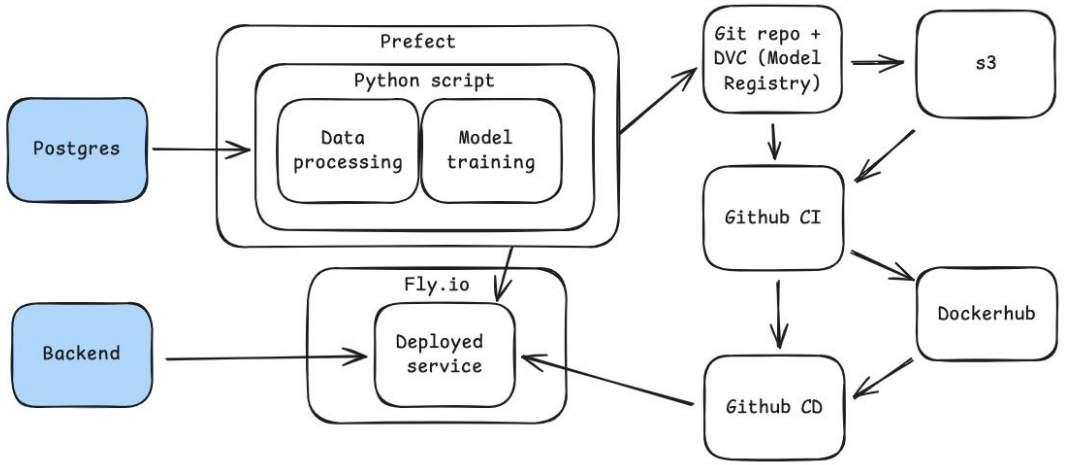
Overview:

- The model development follows a CI/CD pipeline with automated training, versioning using DVC, and deployment via Docker.

Key points:

- Automation: Training, deployment automation.
- Versioning: Model Registry with DVC.
- CI/CD: Continuous integration and deployment pipeline.

Example



ML System Design - Production: EasyRide Taxi

Data Engineering

Data Ingestion: From Postgres every 30 minutes.

ETL Processes: Data cleaning, feature engineering.

Data Versioning: Ensuring reproducibility.

Methodology

Problem Type: Regression.

Approach: Supervised learning.

Metric: MAPE.

Baseline: Current predictions.

Validation

Validation Methodology: Shadow deployment, A/B testing.

Pilot Scope: 1 week

Success Criteria: Lower MAPE, higher booking rates.

Model Development

Automation: Training, deployment automation.

Versioning: Model Registry with DVC.

CI/CD: Continuous integration and deployment pipeline.

Solution

Target: trip duration

Format: Duration in minutes

Components: Taxi App, Data Ingestion, ML Solution, Backend

App/UI/UX

UI: ETA, trip duration and cost/earnings.

UX: Requesting a ride, viewing the predicted cost, booking the ride.

Deployment & Serving

Inference Type: Real-time.

CI/CD: Automated deployment pipeline in GitHub CI

Serving Infrastructure: FastAPI, Docker, Fly.io, PosgreSQL

Deployment Methodology: Shadow deployment, canary releases.

Infrastructure / Resources

Performance / ML Metrics

Prediction accuracy (MAPE) <= 15%

Prediction latency < 500ms

Business Metrics: Booking rate, Revenue Increase

Timeline: Daily metric evaluation.

Testing & Monitoring

Testing and Monitoring

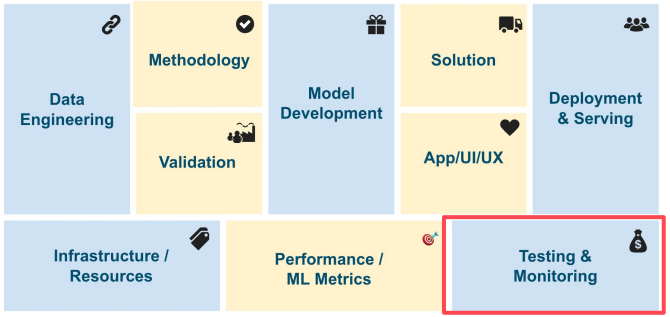
Production ML System Design
> Guide: 5.5 - Testing and Monitoring

5.5 - Testing and Monitoring

What is the post-training quality assurance processes?

Purpose

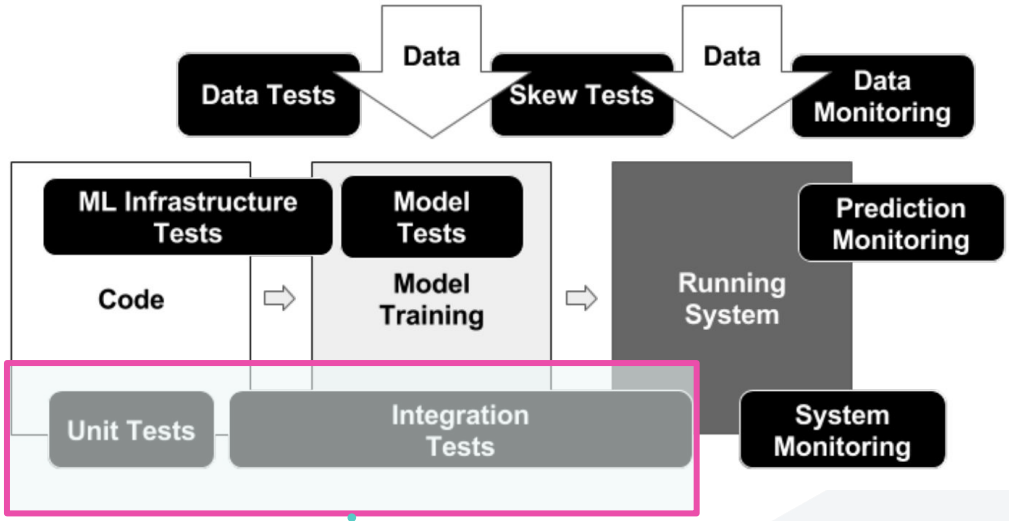
- To maintain system reliability and catch issues before they impact business operations.



Guiding questions:

- How will we monitor the system's performance in production?
- What testing procedures will ensure system reliability?
- How will we maintain quality as the system evolves?
- How we understand the model performs well?
- How will you log events in your system? What metrics will you monitor and how? Will you have alarms if a metric breaches a threshold or something else goes wrong?
- What are model and data metrics to track?

ML testing VS. traditional software testing



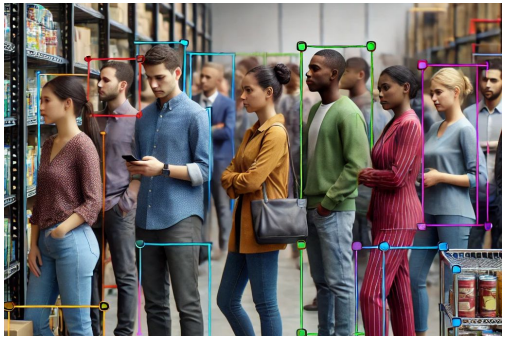
Traditional tests

Image source: The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction" Breck et al. (2017)
https://research.google/pubs/pub46555/

Exercise: What should we car about?

Brainstorming on testing and monitoring needs

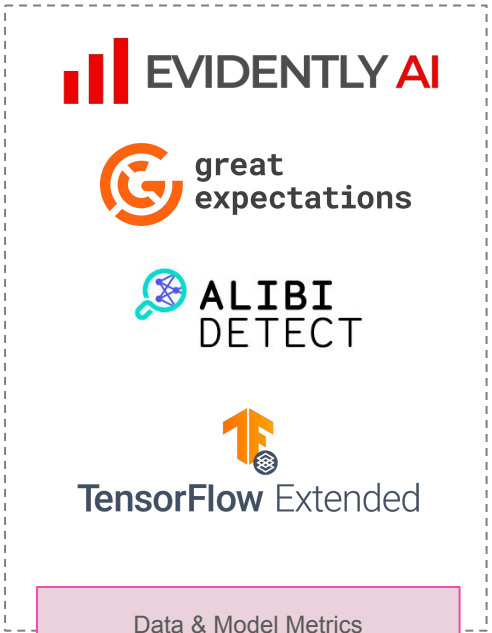
New pizza waiting time prediction

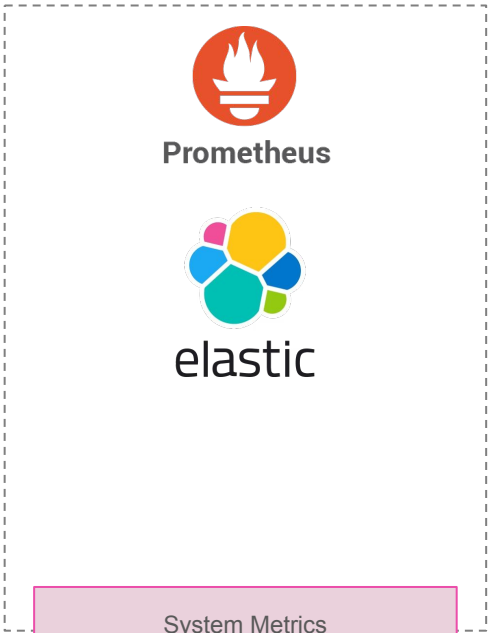


Shop queue detection [CV]

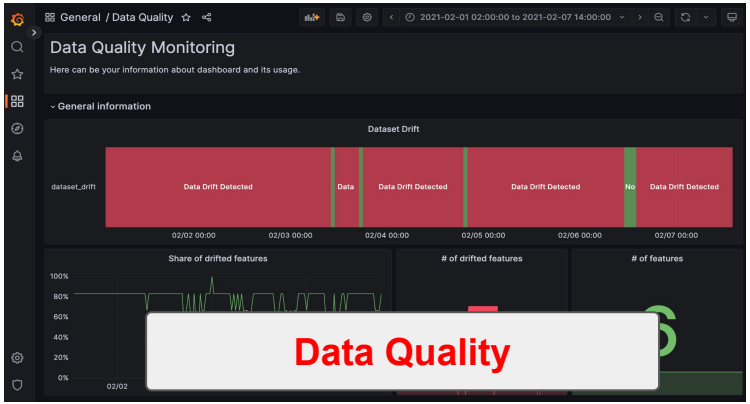
NY Taxi trip duration prediction

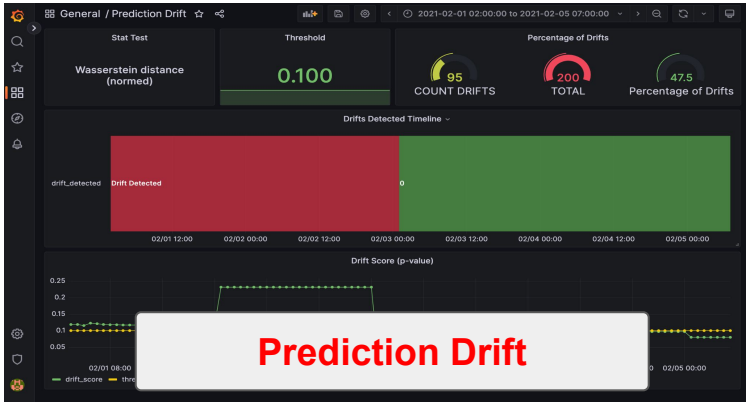
Monitoring tools

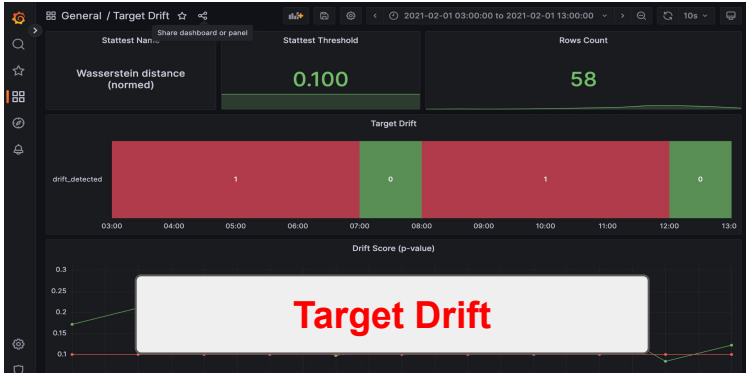




Model Monitoring: Evidently + Grafana







Exercise: Testing and Monitoring

How do we ingest and store data?

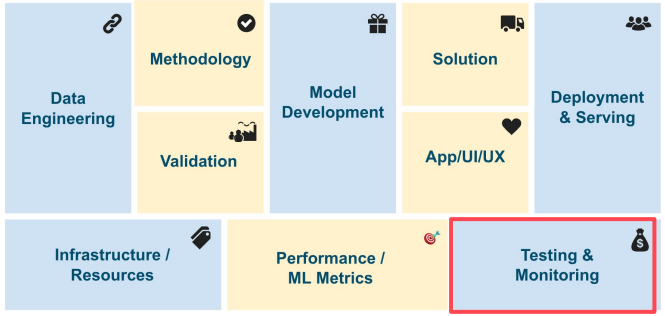
Group task:

- Complete the Production ML System Design section:
  - 5.5 - Testing and Monitoring
- 5 min

Key points:

- Testing strategy (unit, integration, system tests) and important tests
- Monitoring system design
- Monitoring metrics
- Biases and misuses of your model (if applicable)

Practice



5.5 – Testing and Monitoring

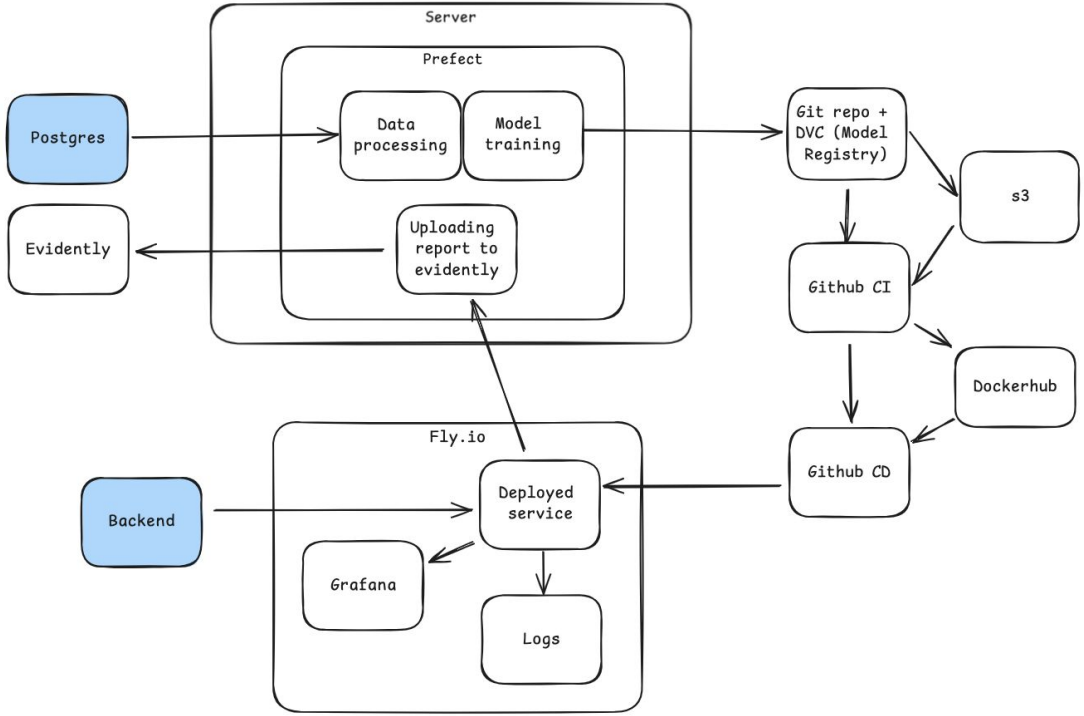
How do we ingest and store data?

Overview:

- System quality is maintained through rigorous testing and monitoring, including unit tests, integration tests, and performance monitoring using Evidently.

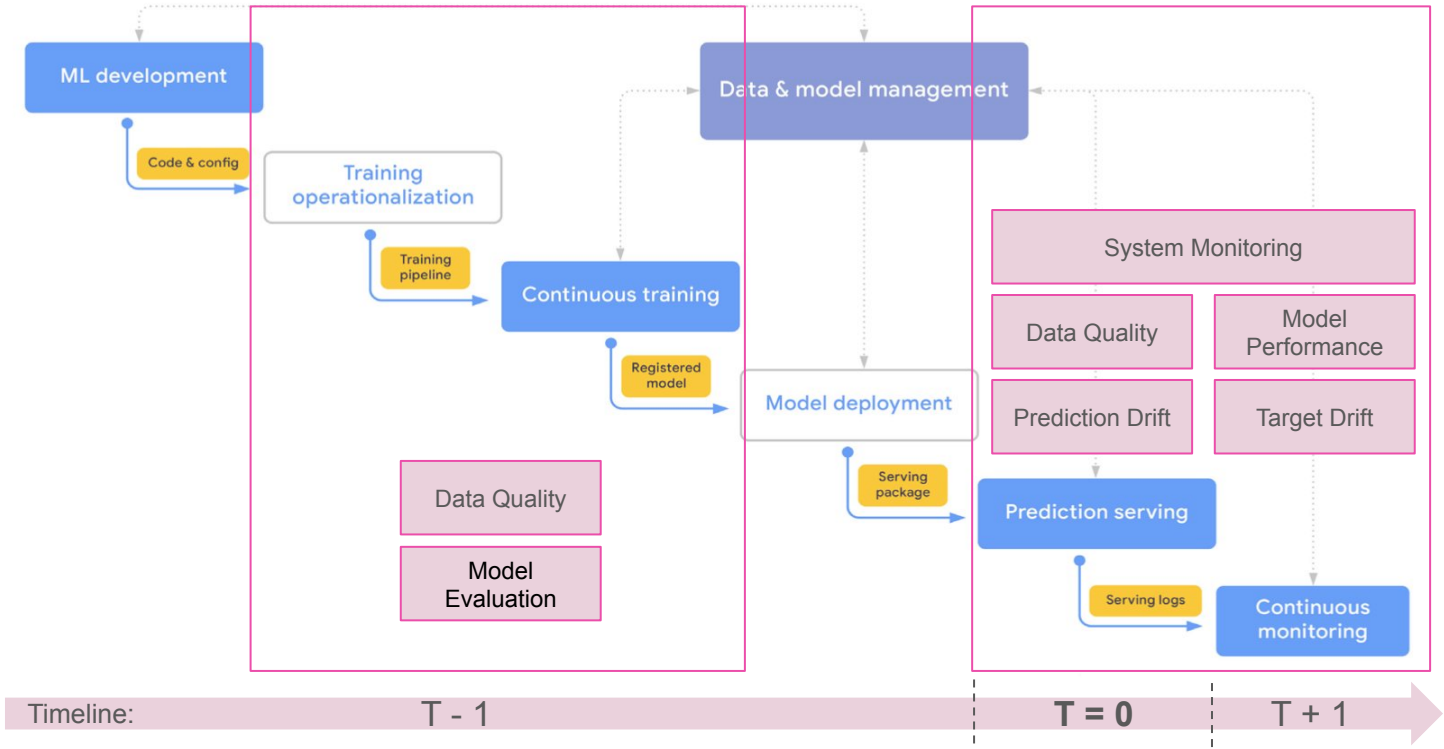
Key points:

- Monitoring: Performance metrics, alerts.
- Testing Strategy: Unit, integration, system tests.
- Quality Assurance: Regular testing/monitoring.



Example

Data monfrom a prototype to monitoring



Source: https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf

ML System Design - Production: EasyRide Taxi

Data Engineering

Data Ingestion: From Postgres every 30 minutes.

ETL Processes: Data cleaning, feature engineering.

Data Versioning: Ensuring reproducibility.

Methodology

Problem Type: Regression.

Approach: Supervised learning.

Metric: MAPE.

Baseline: Current predictions.

Validation

Validation Methodology: Shadow deployment, A/B testing.

Pilot Scope: 1 week

Success Criteria: Lower MAPE, higher booking rates.

Model Development

Automation: Training, deployment automation.

Versioning: Model Registry with DVC.

CI/CD: Continuous integration and deployment pipeline.

Solution

Target: trip duration

Format: Duration in minutes

Components: Taxi App, Data Ingestion, ML Solution, Backend

App/UI/UX

UI: ETA, trip duration and cost/earnings.

UX: Requesting a ride, viewing the predicted cost, booking the ride.

Deployment & Serving

Inference Type: Real-time.

CI/CD: Automated deployment pipeline in GitHub CI

Serving Infrastructure: FastAPI, Docker, Fly.io, PosgreSQL

Deployment Methodology: Shadow deployment, canary releases.

Infrastructure / Resources

-
-

Performance / ML Metrics

Prediction accuracy (MAPE) <= 15%

Prediction latency < 500ms

Business Metrics: Booking rate, Revenue Increase

Timeline: Daily metric evaluation.

Testing & Monitoring

Monitoring: Performance metrics, alerts.

Testing Strategy: Unit, integration, system tests.

Quality Assurance: Regular testing and monitoring.

Scalability and Infrastructure

Production ML System Design

> Guide: 5.6 - Scalability and Infrastructure Planning

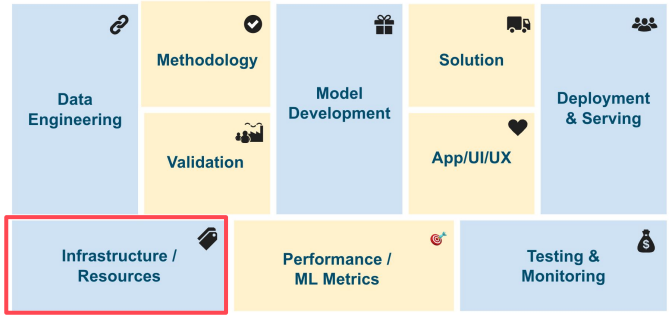
> Guide: 5.7 - Requirements & Constraints

5.6 – Scalability and Infrastructure

What infrastructure do we need?

Purpose

- To ensure the system can grow with the business and handle increased load.

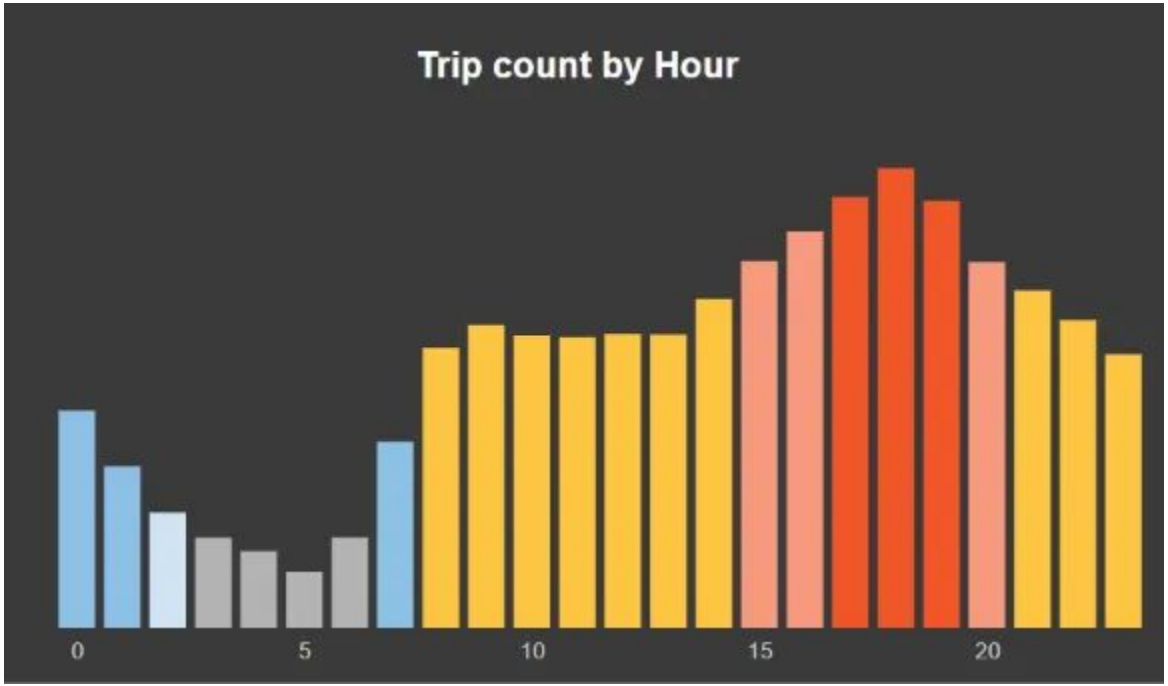


Guiding questions:

- What infrastructure do you need to train and serve the model (CPU, GPU…)?
- How will you host your system? On-premise, cloud, or hybrid?
- How will your system meet the performance (throughput and latency) requirements? Will it scale vertically or horizontally?
- How much will it cost to build and operate your system? Share estimated monthly costs (e.g., EC2 instances, Lambda, etc.)

Exercise: NY Taxi trip counts by hour

Calculate costs for a static and flexible serving infrastructure



Source: https://medium.com/@dustinhsu0605/nyc-taxi-trips-challenge-646a89a05bfd

Exercise: Scalability and Infrastructure

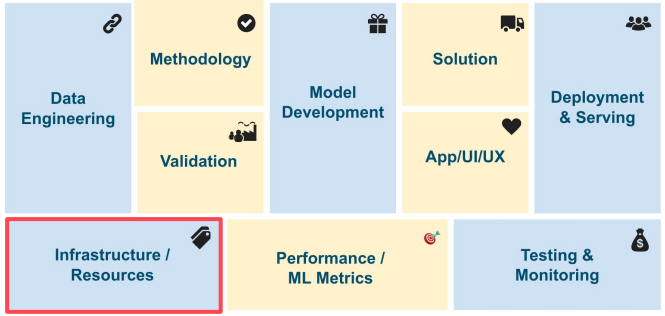
How do we ingest and store data?

Group task:

- Complete the Production ML System Design section:
  - 5.6 - Scalability and Infrastructure
- 5 min

Key points:

- Infrastructure requirements (CPU, GPU…)
- Scalability requirements and approach
- Infrastructure costs



Practice

5.6 – Scalability and Infrastructure

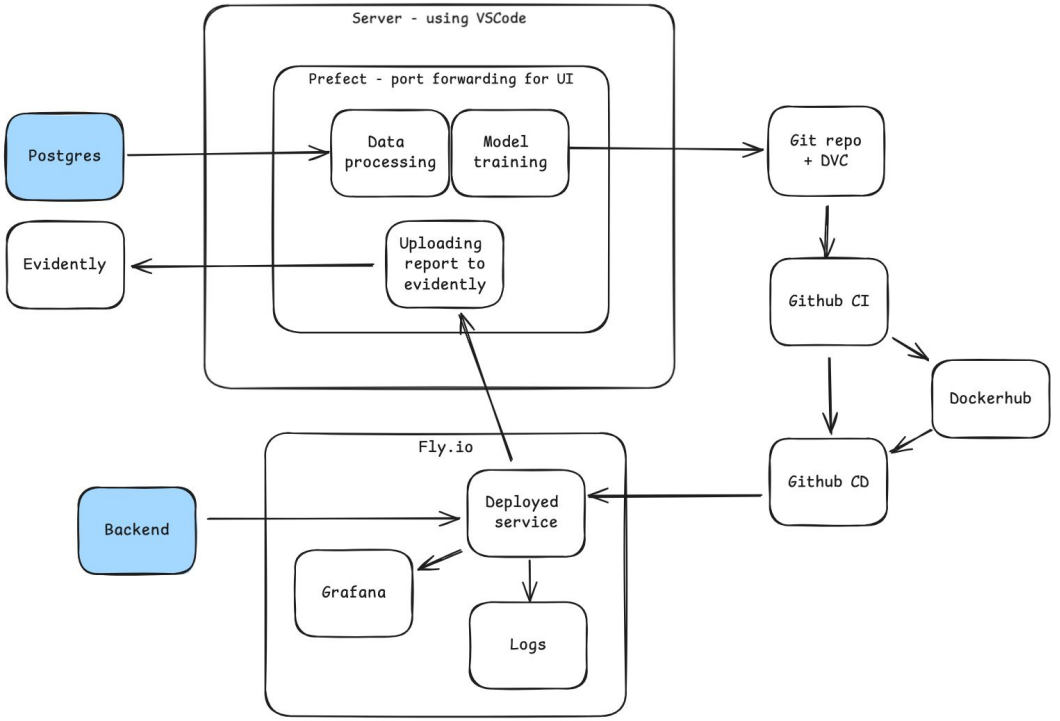
How do we ingest and store data?

Overview:

- The system is designed for scalability, with plans for future infrastructure growth and performance optimization strategies.

Key points:

- Cloud: Fly.io
- Resources:
  - ETL & Train: 16GB RAM, 4vcpu
  - Inference: 1GB RAM, 1vcpu
- Scalability: Not required at this step



Example

ML System Design - Production: EasyRide Taxi

Data Engineering

Data Ingestion: From Postgres every 30 minutes.

ETL Processes: Data cleaning, feature engineering.

Data Versioning: Ensuring reproducibility.

Methodology

Problem Type: Regression.

Approach: Supervised learning.

Metric: MAPE.

Baseline: Current predictions.

Validation

Validation Methodology: Shadow deployment, A/B testing.

Pilot Scope: 1 week

Success Criteria: Lower MAPE, higher booking rates.

Model Development

Automation: Training, deployment automation.

Versioning: Model Registry with DVC.

CI/CD: Continuous integration and deployment pipeline.

Solution

Target: trip duration

Format: Duration in minutes

Components: Taxi App, Data Ingestion, ML Solution, Backend

App/UI/UX

UI: ETA, trip duration and cost/earnings.

UX: Requesting a ride, viewing the predicted cost, booking the ride.

Deployment & Serving

Inference Type: Real-time.

CI/CD: Automated deployment pipeline in GitHub CI

Serving Infrastructure: FastAPI, Docker, Fly.io, PosgreSQL

Deployment Methodology: Shadow deployment, canary releases.

Infrastructure / Resources

Cloud: Fly.io

- ETL & Train: 16GB RAM, 4vcpu
- Inference: 1GB RAM, 1vcpu

Scalability: Not required at this step

Performance / ML Metrics

Prediction accuracy (MAPE) <= 15%

Prediction latency < 500ms

Business Metrics: Booking rate, Revenue Increase

Timeline: Daily metric evaluation.

Testing & Monitoring

Monitoring: Performance metrics, alerts.

Testing Strategy: Unit, integration, system tests.

Quality Assurance: Regular testing and monitoring.

Assignment

Start with Production ML System Design!

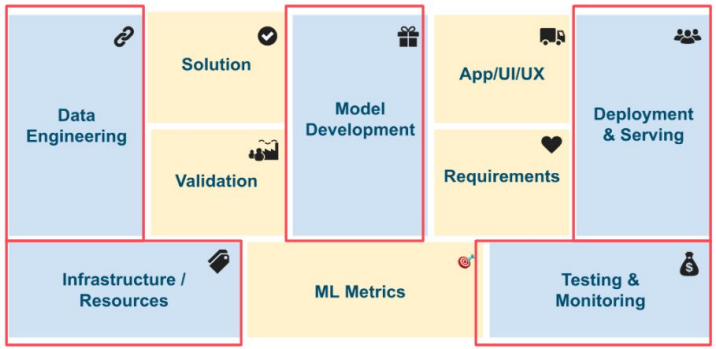
Prerequisites

- Complete ML Product Design: Business Understanding
- Complete ML Product Design: Solution
- Complete ML Product Design: Methodology

Practice: Production ML System

Automate and scale!

- Brainstorm a Production ML System design for for your project
- Follow the guide to describe each section in **Production ML System**
- Summarise **Production ML System** on the canvas
- Update “ML Solution” blocks if needed
- Update the “Cost Structure” if needed



Materials & Links

- Course Materials: Google Drive
- Practice - EasyRide Taxi - PUBLIC
- Guide - ML System Design - Canvas