edquest

Day 9 Agenda

Monitoring, Logging & Security

- Recap
- Quiz for the previous Day
- Monitoring
- Logging
- Security
- Quiz

Post Deployment Lifecycle of ML Model

A model's journey doesn't end at deployment—it starts there.

Why Post-Deployment Monitoring Matters

- ML models can **decay in performance** after going live
- Real-world data ≠ training data → leads to **data drift**
- Business impact: wrong predictions = financial loss, trust erosion, risk

ML Lifecycle

Train → Evaluate → Deploy → **Monitor → Secure → Iterate**

Fact: 85% of ML models never reach production stability due to lack of monitoring and feedback loops. — (VentureBeat, 2022)

Challenges After Deployment

- Model performance degrades (silent failures)
- Lack of visibility: no logs, no alerts
- Adversarial attacks or data poisoning
- Changing business requirements → no tracking

edquest

The Three Pillars of Post-Deployment MLOps

Pillar	What It Does	Tools/Practices
Monitoring	Track model behavior, drift, accuracy	Prometheus, Grafana, Evidently
Logging	Record inputs, outputs, errors	MLflow, ELK Stack, Sentry
Security	Prevent misuse, protect data & model	AuthN/AuthZ, model hardening

edquest

Typical Deployment Flow

- Data ingestion
- Model prediction service (API)
- Logging/Monitoring layer
- Alerting/notifications
- Governance & audit layer (security)

edquest

Model Monitoring

?

edquest

Model Monitoring

- Continuous tracking of model predictions and behavior in production
- Detects issues like performance degradation, drift, and outliers
- Enables early alerts and feedback loops

Why Monitoring is Critical ?

- Model predicts with 95% accuracy in training → drops to 65% in prod
- Business rules change → model is outdated
- Inference latency spikes due to infrastructure issues

Fact: You can’t improve what you don’t measure.

What to Monitor in ML Systems

Category	Metric Example
Performance	Accuracy, Precision, Recall, F1
Data Drift	Change in input feature distribution
Concept Drift	Change in relationship X → Y
Latency	Avg prediction time, spikes
Volume	Number of predictions/hour
Outliers	Anomalous input patterns

edquest

Data Drift vs Concept Drift

Data drift: Input feature distribution changes

Concept drift: Target variable behavior changes

Examples:

- Data Drift: “Age” input starts including 10-year-olds (unexpected)
- Concept Drift: Customer behavior shifts due to pandemic

Model Monitoring Summary

- Monitoring is essential for **reliable AI systems**
- Focus on **data + concept drift, latency**, and **performance**
- Use tools like **Evidently, Prometheus, Grafana**
- Set actionable **thresholds and alerts**

Why Logging matters in ML System ?

Why Logging matters in ML System ?

- Logging helps with **debugging, root cause analysis**, and **auditing**
- Logs tell the story:
  - what input came in,
  - what prediction was made
  - what went wrong
- Enables reproducibility and compliance

What should We Log ?

Log Type	Examples
Input data	Features, timestamps, request ID
Output	Prediction, confidence score
Metadata	Model version, latency, environment
Errors	Exceptions, missing fields, timeouts

edquest

Logging Best Practices

- Use **structured formats** (JSON preferred)
- Tag logs with model version + request ID
- Separate logs by severity (INFO, WARN, ERROR)
- Avoid logging **PII** or **sensitive data**
- Log **enough context** to debug

Logging in FastAPI Model Server

```python
import logging
logger = logging.getLogger("ml_logger")

@app.post("/predict")
def predict(data: InputData):
    logger.info({
        "model_version": "v1.2",
        "input": data.dict(),
        "prediction": result,
        "latency_ms": latency
    })
```

Tip: Use middleware to log all incoming requests automatically.

MLflow Logging Example

```python
import mlflow

with mlflow.start_run():
    mlflow.log_params({"learning_rate": 0.01, "max_depth": 4})
    mlflow.log_metrics({"accuracy": 0.89, "f1": 0.87})
    mlflow.log_artifact("confusion_matrix.png")
```

Use Case: Track multiple model runs, parameters, and performance side-by-side.

Logging Summary

- Logging = visibility into your ML system
- Structured logs enable faster debugging and better monitoring
- Use MLflow for experiment tracking; ELK for operational logs
- Never log PII or secrets
- Integrate alerts on errors and unusual patterns

Why
Model Security ?

Model Security

- Deployed models are **attack surfaces**
- Sensitive data can be **leaked, reversed**, or **manipulated**
- ML-specific attacks are **not well-covered** by traditional IT security

💡 "You can't secure what you don't understand — and ML behaves differently than traditional software."

API Protection and Rate Limiting

- Authenticate API calls (API keys, OAuth)
- Use **rate limiting** to avoid scraping or brute force
- Log IP addresses and block malicious users

```
from fastapi_limiter import FastAPILimiter
```

edquest

Model Versioning and Signing

- Sign model artifacts (e.g., using SHA256 or GPG)
- Store model hashes in source control or registry
- Prevent serving tampered or outdated models

Tools: MLflow Model Registry or SageMaker Model Registry

Secure Storage and Deployment

- Store models in private object storage (e.g., AWS S3 with IAM)
- Encrypt model files at rest
- Deploy in isolated environments (e.g., Docker containers, VPCs)

Access Control for ML Systems

- Who can retrain the model?
- Who can deploy a new version?
- Who can call the model in production?

Model Security Summary

- ML introduces new security risks → address them early
- Protect models, APIs, and data
- Use **authentication, rate limiting, encryption**, and **monitoring**
- ML security is a team responsibility: Dev + Data + Security

💡 “Secure models = trustworthy models”