SQUIDSPARK

Evolving DevOps into MLOps

Managing Code, Data, And Models

White Paper

A comparative perspective

TABLE OF CONTENT

INTRODUCTION

DevOps unifies software development and IT operations to deliver applications faster and more reliably. It emphasizes automation, CI/CD, infrastructure as code, and collaboration between developers and operations.

MLOps builds on this foundation but adds complexity. Instead of managing only code, it must also manage data pipelines, training workflows, model artifacts, and monitoring for drift and fairness. Where DevOps ensures code reliability, MLOps ensures the full ML lifecycle remains healthy and repeatable. This paper compares DevOps and MLOps across key areas to highlight similarities, differences, and the unique demands of machine learning.

GOAL & SCOPE

While **DevOps** focuses on accelerating software delivery through CI/CD and reliable operations, **MLOps** expands the scope to include the entire machine learning lifecycle, from data ingestion to retraining, ensuring both code and models remain effective over time.

DevOps

Speeds up software delivery, bridging development and operations through CI/CD, automation, and continuous monitoring.

MLOps

Expands this to cover the full ML lifecycle—data ingestion, preprocessing, model training, validation, deployment, monitoring, and retraining.

Similarities

Both emphasize automation, scalability, and cross-team collaboration.

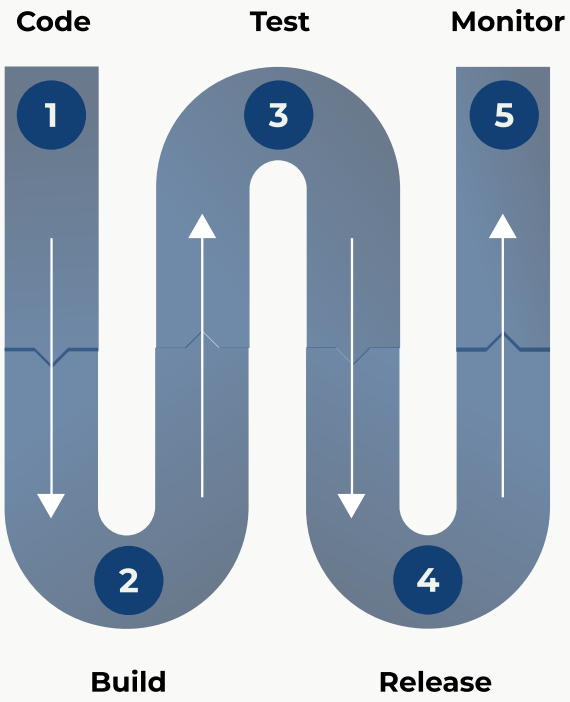
Differences

DevOps manages deterministic code; MLOps manages code, data, and models that evolve with changing inputs.

LIFECYCLE COMPARISON

DevOps

The **DevOps** lifecycle is a continuous, cyclical process designed to deliver software rapidly and reliably. It moves through stages of coding, building, testing, releasing, and monitoring, with automation and CI/CD pipelines ensuring predictable delivery and quick feedback loops that drive continuous improvement.



MLOps

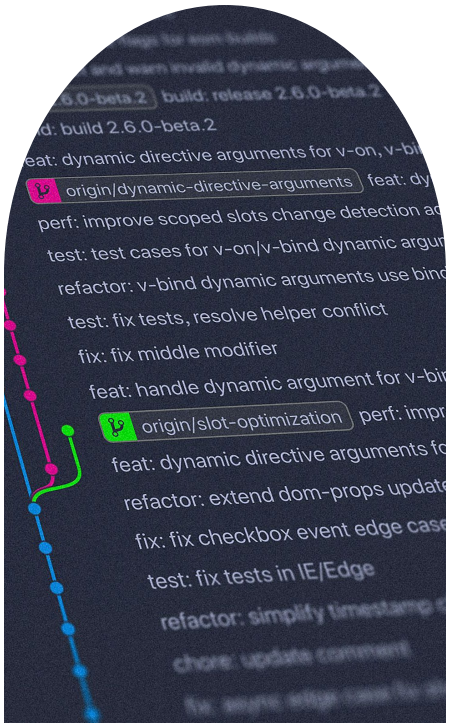
The **MLOps** lifecycle builds on these principles but extends them to handle data and models. It begins with data collection and feature engineering, followed by model training, validation, deployment, and monitoring. Unlike **DevOps**, it incorporates a retraining loop, since model performance degrades over time due to data and concept drift, requiring ongoing adaptation to stay accurate and relevant.

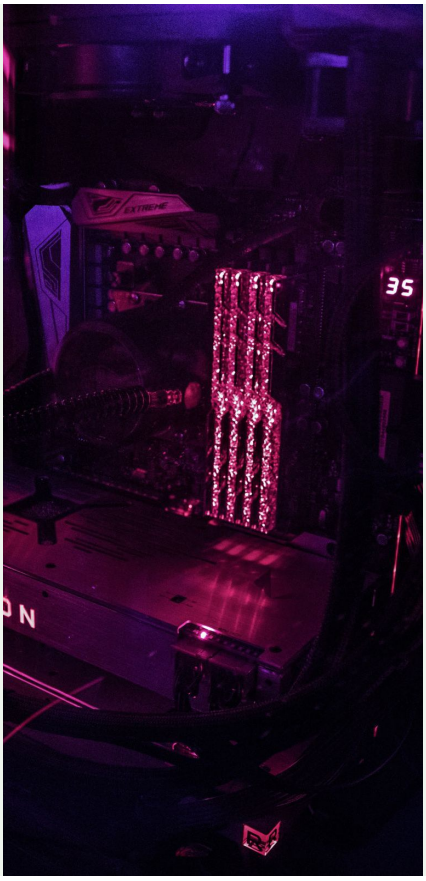
Version Control

In **DevOps**, Git or similar systems track source code, enabling branching, merging, and rollback. Code versioning ensures reproducibility and fuels CI/CD pipelines.

MLOps requires additional versioning:

- Datasets (raw snapshots, processed outputs, schemas)
- Features (curated model inputs that evolve over time)
- Models (trained artifacts like .pkl or .onnx)
- Experiment configurations (hyperparameters, seeds, environments)





Infrastructure

DevOps uses IaC (Terraform, Ansible) and container orchestration (Kubernetes) to automate provisioning, scaling, and environment consistency. **MLOps** infrastructure adds:

- Specialized hardware (GPUs/TPUs for training).
- Distributed training clusters for large-scale datasets.
- Feature stores to keep features consistent across training and inference.
- Model registries to track model versions, approvals, and metadata.
- Data pipelines (Spark, Databricks, Azure Data Factory) for large-scale preparation.

Continuous Integration

DevOps CI validates that new code compiles, passes tests, and integrates smoothly. A typical pipeline builds the app, runs unit and integration tests, and enforces coding standards.

MLOps CI must go further, validating:

- Data – schema consistency, missing values, anomalies.
- Features – correct computation from raw inputs.
- Model training – successful convergence and accuracy thresholds.
- Reproducibility – consistent outputs across runs with fixed seeds.

Continuous Delivery

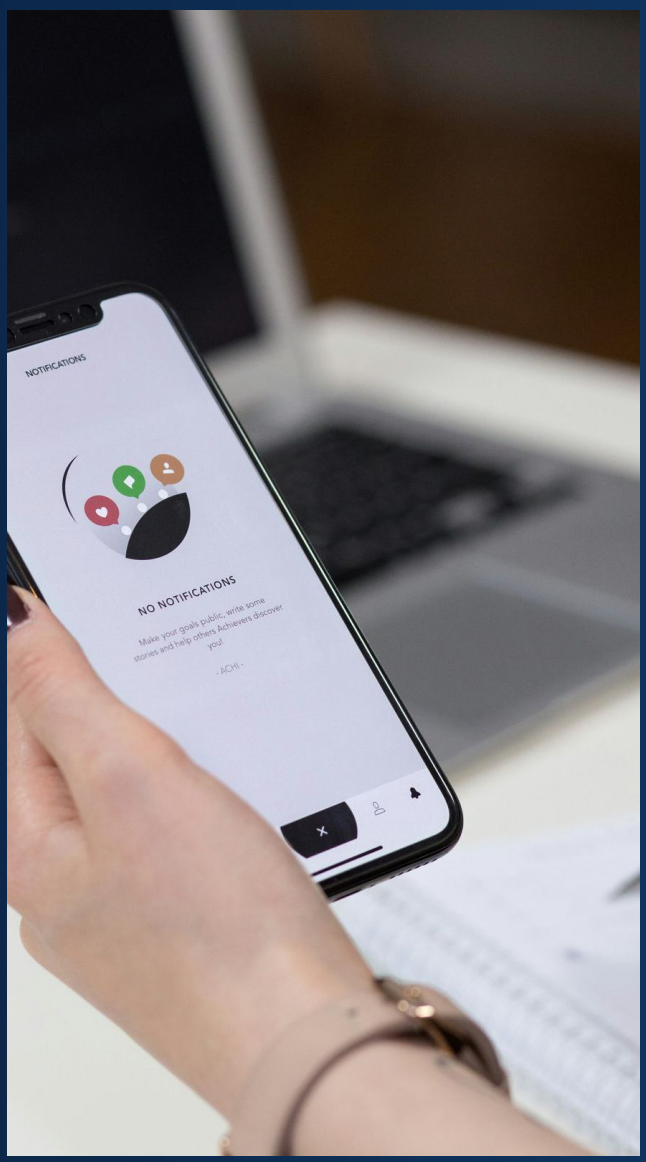
DevOps CD deploys applications automatically via containers, Kubernetes, and blue/green rollouts. MLOps CD deploys trained models plus serving infrastructure.

Pipelines include:

- Packaging the model (e.g., ONNX in Docker).
- Deploying to a serving platform (Seldon, SageMaker, TF Serving).
- Post-deployment validation or shadow deployment.
- A/B testing and, in sensitive domains, human approval.

For example, a retail platform's microservice can be rolled out via DevOps CD. But a new recommendation model must be tested cautiously, since even higher training accuracy might fail with new customer behavior or market shifts. This is a type of drift. **Drift** refers to the gradual change in data patterns or relationships (data drift or concept drift) that causes a deployed model's predictions to become less accurate over time.

TESTING



DevOps testing: unit, integration, performance, and security tests. Behavior is deterministic—same input yields same output.

MLOps testing:

- Data validation – schema and distribution checks.
- Model validation – metrics (accuracy, recall, F1) plus fairness and bias testing.
- Pipeline reproducibility – ensuring retraining with same conditions produces consistent results.

Example: A banking app validates API responses under load (**DevOps**). A credit risk model also tests for > 90% recall, reproducibility across runs, and fairness across groups (**MLOps**).

Monitoring

DevOps monitoring involves uptime, latency, error rates, and resource usage. Ensures services remain available and performant.

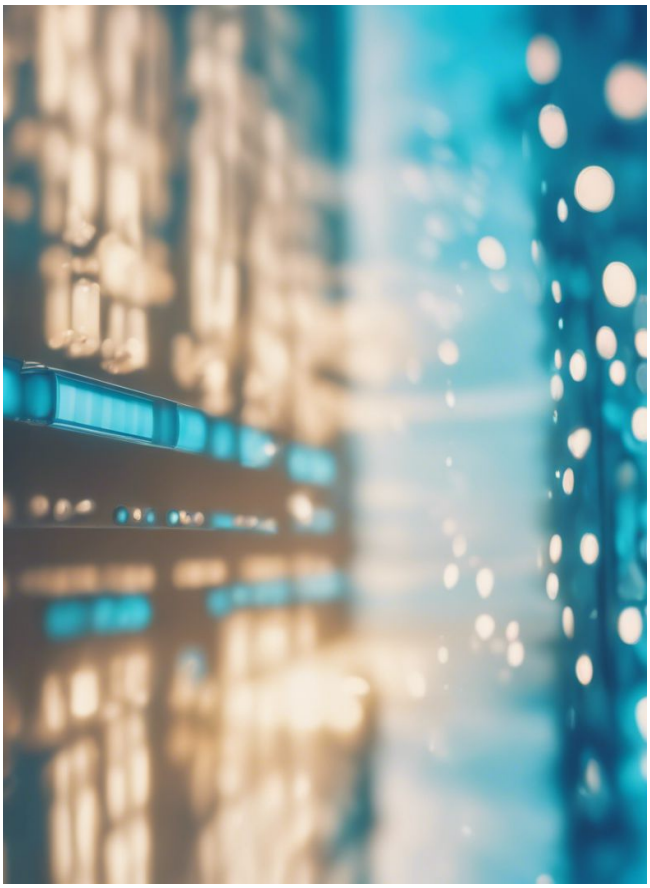
MLOps monitoring: adds model-specific metrics:

- Accuracy, recall, precision, or KPIs in production.
- Data drift (input distributions changing).
- Concept drift (relationships between inputs and labels shifting).
- Fairness, bias, and explainability checks.
- Retraining triggers when thresholds are breached.

For example, an e-commerce site tracks checkout uptime (DevOps). Its recommendation model also monitors seasonal drift, fairness across sellers, and triggers retraining when performance drops (MLOps).

CONCLUSION

DevOps revolutionized software delivery with automation, CI/CD, and collaboration. MLOps extends these ideas to ML systems, introducing practices for data, models, and retraining. The overlap provides efficiency and reliability, but MLOps adds the ability to handle drift, fairness, and probabilistic outcomes. Together, these practices ensure that both applications and ML-powered systems remain accurate, reliable, and scalable in production.



We Turn Complexity into Clarity

White Paper