

**Michael Tyiska**

Posted on Dec 8, 2024 • Edited on Dec 12, 2024



2



1

Setting Up Machine Learning Pipelines with GitOps Principles

#mlops #gitops #kubernetes #devops

In the ever-evolving world of DevOps and machine learning, building scalable and automated workflows has become a cornerstone for success. This guide walks you through setting up an end-to-end MLOps pipeline using GitOps principles, incorporating tools like Argo Workflows, Argo Events, MinIO, FastAPI, MLflow, Kubernetes, and Evidently AI. By following this setup, you'll have a robust system to detect data drift, retrain models, and deploy them seamlessly.

Architecture Overview

This architecture is designed to handle the entire lifecycle of a machine learning model, ensuring it stays accurate and reliable over time.

Pre-Deployment

We start with **Data Preparation**, where raw datasets are cleaned and split into training and testing sets. Next, in **Model Development**, we train and evaluate multiple models using MLflow to identify the best-performing one. Finally, the best model, along with the processed datasets and reference files, is stored in **MinIO** for version control and easy retrieval.

CI/CD Integration

The pipeline incorporates continuous integration and deployment principles to ensure that changes to the system, whether in the codebase or data, are quickly and safely integrated. This includes linting, testing, and automated deployment steps.

Deployment

The trained model is deployed via a lightweight **FastAPI** application, which serves predictions and continuously monitors incoming data for drift. This ensures that the model remains accessible and up-to-date.

Post-Deployment

Using **Evidently AI**, the system monitors the production data for drift. If significant drift is detected, a **Kubernetes CronJob** triggers the retraining process. The retraining uses the updated data, and the new model is saved back to MinIO. The **FastAPI** app then dynamically reloads the updated model, enabling seamless updates without manual intervention.

Feedback Loop

This workflow creates a fully automated feedback loop: it detects drift, retrains models, and redeploys them, ensuring that the system remains reliable and accurate over time.

Here's a high-level diagram of the architecture:



Getting Started

Environment Setup

This section focuses on preparing your Python environment for the MLOps pipeline. By using tools like conda and pip, we ensure that all dependencies for retraining models, deploying applications, and development are properly installed. Setting up a clean environment is critical to avoid conflicts between packages and to maintain reproducibility across systems. The MLflow server, a key component, is used to log and track your machine learning experiments in a centralized way.

Start by setting up your Python environment and installing dependencies:

```
conda create --name mlops_env python=3.11 -y
conda activate mlops_env
pip install -r requirements/requirements.retrain.txt --force-reinstall
pip install -r requirements/requirements.app.txt
pip install -r requirements/requirements.dev.txt
```

Launch the MLflow server to track experiments:

```
mlflow server --host 127.0.0.1 --port 8080
```

Makefile Commands

Makefiles streamline repetitive tasks by defining commands that can be run consistently with a single line. This section highlights common operations like linting, testing, data preparation, and model training. Each Makefile command corresponds to a specific part of the workflow, making it easy to execute complex tasks without manually typing long commands.

Here are the key commands:

1. Linting and Testing

```
make lint
make test
```

1. Data Preparation

```
make run-data-preparation
```

1. Model Training

```
make run-model-training
```

1. Model Evaluation

```
make run-evaluate
```

1. Model Retraining

```
make run-retrain
```

1. Run FastAPI Locally

```
make run-fastapi
```

Access the API documentation:

<http://127.0.0.1:8000/docs>

Setting Up Kubernetes

Kubernetes is a container orchestration platform that allows you to deploy, manage, and scale services. This section guides you through creating namespaces, which provide logical isolation for different services like MLflow, MinIO, and FastAPI. Using namespaces ensures that each service runs independently, simplifying resource management and troubleshooting.

Namespace Creation

Create separate namespaces for each service:

```
kubectl create namespace mlserver  
kubectl create namespace minio  
kubectl create namespace mlflow  
kubectl create namespace fastapi  
kubectl create namespace argo  
kubectl create namespace argo-events
```

MinIO Setup

MinIO is an object storage system compatible with AWS S3. It is used in this setup to store large datasets, models, and other artifacts. Deploying MinIO involves setting up services and ingress configurations to enable access. Exposing MinIO services allows interaction with the storage directly, making it a critical part of the data pipeline.

1. Deploy MinIO for object storage:

```
kubectl apply -f minio_depl.yml  
kubectl apply -f minio-ingress.yaml
```

1. Expose MinIO services:

```
kubectl port-forward svc/minio-service 9000:9000 -n minio
```

Deploy MLServer

MLServer serves machine learning models and provides an interface for inference requests. This section outlines how to build a Docker image for MLServer, push it to a

container registry, and deploy it on Kubernetes. By port-forwarding the service, you can access MLServer locally, making it easier to test and debug deployments.

Build and push the MLServer image:

```
docker build --platform linux/amd64 -t measureapp/mlserver:0.0.2 .  
docker push measureapp/mlserver:0.0.2
```

Apply the deployment:

```
kubectl apply -f mlserver.yaml  
kubectl get pods -n mlserver
```

Access the service:

```
kubectl port-forward svc/mlserver-service 5000:5000 -n mlserver
```

Steps to Set Up MLServer

This section expands on deploying MLServer by detailing additional steps like creating a Docker registry secret for pulling container images securely. It also covers verifying the deployed MLServer instance and accessing the MLflow UI for monitoring workflows within the pod. This ensures the MLServer is correctly configured and running.

1. Build and Push MLServer Docker Image:

```
docker build --platform linux/amd64 -t measureapp/mlserver:0.0.2 .  
docker push measureapp/mlserver:0.0.2
```

1. Create Docker Registry Secret:

```
kubectl create secret -n mlserver docker-registry docker-registry-creds \  
  --docker-server=https://index.docker.io/v1/ \  
  --docker-username=measureapp --docker-password=sensitive_password
```

1. Deploy MLServer:

```
kubectl apply -f mlserver.yaml
```

1. Verify MLServer Pods:

```
kubectl get pods -n mlserver
```

1. Access MLServer via Port Forwarding:

- Forward the service port:

```
kubectl port-forward svc/mlserver-service 5000:5000 -n mlserver
```

- Open the MLflow UI in your browser:

```
http://localhost:5000
```

1. Alternative Port Forwarding:

```
kubectl port-forward svc/mlserver-service 8080:8080 -n mlserver
```

1. Verify MLflow Process Inside MLServer Pod:

- Execute into the pod:

```
kubectl exec -it mlserver-<pod-name> -n mlserver -- bash
```

- Check running processes:

```
ps aux | grep mlflow  
ps aux | grep mlserver
```

Steps to Set Up Model Job

The model job refers to the training and evaluation pipeline for machine learning models. This section discusses building a containerized job, securely managing container images via Kubernetes secrets, and deploying the job on the cluster. It ensures that model training runs reliably and integrates well with other components.

1. Create Docker Registry Secret for MLflow Namespace:

```
kubectl create secret -n mlflow docker-registry docker-registry-creds \  
  --docker-server=https://index.docker.io/v1/ \  
  --docker-username=measureapp --docker-password=passwordbuild
```

1. Build and Push Model Job Docker Image:

```
docker build --platform linux/amd64 -t measureapp/ml_job:0.0.26 -f docker/model/D
docker push measureapp/ml_job:0.0.26
```



1. Deploy Model Job:

```
kubectl apply -f model-train-job.yaml
```

1. Verify Model Job Pods:

```
kubectl -n mlflow get pods
```

Steps to Set Up FastAPI

FastAPI is used to serve the API endpoints for interacting with the machine learning pipeline. This section explains setting up Docker registry credentials, building and deploying FastAPI as a container, and accessing the service locally. FastAPI provides a user-friendly interface for interacting with models and simulating various scenarios.

1. Create Docker Registry Secret for FastAPI Namespace:

```
kubectl create secret -n fastapi docker-registry docker-registry-creds \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username=measureapp --docker-password=sensitive_password
```

1. Build and Push FastAPI Docker Image:

```
docker build --platform linux/amd64 -t measureapp/demo_ai_api:0.0.40 -f docker/fa
docker push measureapp/demo_ai_api:0.0.40
```



1. Deploy FastAPI:

```
kubectl apply -f fastapi-depl.yaml
```

1. Verify FastAPI Pods:

```
kubectl -n fastapi get pods
```

1. Access FastAPI Service via Port Forwarding:


```
kubectl -n fastapi port-forward svc/fastapi-service 8000:80
```

Open in browser:

```
http://localhost:8000
```

Steps to Set Up Argo Workflows and Events

Argo Workflows orchestrate complex workflows in Kubernetes. This section describes setting up Argo for managing pipeline automation and integrating Argo Events for triggering workflows based on specific events like data drift. The integration ensures your pipeline is dynamic and responsive to changes.

Install Argo Workflows

1. Set Argo Workflows Version:

```
ARGO_WORKFLOWS_VERSION="v3.6.0"
```

1. Install Argo Workflows:

```
kubectl apply -n argo -f "https://github.com/argoproj/argo-workflows/releases/download/v3.6.0/argo-workflows.yaml"
```



1. Verify Installation:

```
kubectl get all -n argo
```

1. Wait for Pods to Be Ready.

2. Access Argo Server:

- Port forward the Argo server:

```
kubectl -n argo port-forward service/argo-server 2746:2746
```

- Open the UI in your browser:

```
https://localhost:2746
```

Create Secrets

1. GitHub Credentials for Argo:

```
kubectl create secret generic git-credentials \
  --from-literal=username=mtyiska \
  --from-literal=token=sensitive_token -n argo
```

1. GitHub Token for Argo Events:

```
kubectl create secret generic github-token-secret \
  --from-literal=token=sensitive_token \
  --namespace=argo-events
```

1. Docker Registry Credentials for Argo:

```
kubectl create secret -n argo docker-registry docker-registry-creds \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username=measureapp --docker-password=sensitive_password
```

1. GitHub Credentials for Argo Events:

```
kubectl create secret generic git-credentials \
  --from-literal=username=mtyiska \
  --from-literal=token=sensitive_token -n argo-events
```

1. Docker Registry Credentials for Argo Events:

```
kubectl create secret -n argo-events docker-registry docker-registry-creds \
  --docker-server=https://index.docker.io/v1/ \
  --docker-username=measureapp --docker-password=sensitive_password
```

Install Argo Events

Drift detection is essential for monitoring model performance over time. This section explains how to set up workflows and events to detect and handle drift automatically. By simulating drift through FastAPI or triggering events manually, you can validate the robustness of the drift handling pipeline.

1. Install Argo Events:

```
kubectl apply -f https://raw.githubusercontent.com/argoproj/argo-events/stable/ma
```

1. Install Validating Webhook:

```
kubectl apply -f https://raw.githubusercontent.com/argoproj/argo-events/stable/ma
```

1. Deploy Native EventBus:

```
kubectl apply -n argo-events -f https://raw.githubusercontent.com/argoproj/argo-e
```

1. Create RBAC Policies:

- For sensors:

```
kubectl apply -n argo-events -f https://raw.githubusercontent.com/argoproj/argo-e
```

- For workflows:

```
kubectl apply -n argo-events -f https://raw.githubusercontent.com/argoproj/argo-e
```

1. Verify Installation:

```
kubectl get all -n argo-events
```

1. Wait for Pods to Be Ready.

2. Verify Default Service Account:

```
kubectl -n argo-events describe serviceaccount default
```

1. Apply Patch:

- Navigate to the `serviceaccount` directory:

```
cd serviceaccount
```

- Apply the patch:

```
kubectl apply -f .
```

1. Verify Service Account:

```
kubectl -n argo-events describe serviceaccount default
```

Drift Workflow and Events

This section focuses on creating and deploying a drift detection job. Drift detection jobs analyze incoming data for changes that might impact model performance, ensuring that models remain accurate and reliable. It highlights the importance of monitoring changes in data distribution.

1. Navigate to Drift Workflow Directory:

```
cd argo
```

1. Apply Drift Workflow and Events:

```
kubectl apply -f .
```

Build and Deploy Drift Detection Job

1. Build Drift Detection Docker Image:

```
docker build --platform linux/amd64 -t measureapp/drift_detection:0.0.4 -f docker
```



1. Push Drift Detection Docker Image:

```
docker push measureapp/drift_detection:0.0.4
```

1. Deploy Drift Detection Job:

```
kubectl apply -f drift-job.yaml
```

1. Verify Pods:

```
kubectl -n mlflow get pods
```

Access Argo Server

Accessing the Argo server enables you to view and manage workflow executions. This section explains how to connect to the Argo UI, where you can visualize workflows, monitor their progress, and troubleshoot issues. The UI provides insights into the automation and health of your pipeline.

1. Port Forward Argo Server:

```
kubectl -n argo port-forward service/argo-server 2746:2746
```

1. Access UI:

```
https://localhost:2746
```

Manually Submit the Workflow

Manual workflow submission helps test the pipeline and ensure all configurations are correct. This section explains how to trigger workflows directly from the Argo CLI, which is helpful during development and debugging. Monitoring execution logs provides visibility into each step, ensuring workflows run as expected.

1. Submit Workflow Using Argo CLI:

```
argo submit --from workflowtemplate/drift-retrain-template \
  -p data-dir="s3://minio/data/app/data/processed" \
  -p model-dir="s3://minio/models/best_model" \
  -p reference-data-path="s3://minio/data/app/data/processed/X_train.csv" \
  -p current-data-path="s3://minio/data/app/data/processed/X_test.csv" \
  -n argo-events
```

1. Monitor Workflow Execution:

```
kubectl logs -n argo-events -f $(kubectl get pods -n mlflow --selector=workflows.
```



Test and Simulate Drift

Testing drift scenarios is crucial for validating your pipeline's ability to handle changes in data. This section explains how to simulate drift using FastAPI endpoints and

monitor logs to verify that drift events are detected and processed correctly. This step ensures your pipeline is prepared for real-world data challenges.

1. Trigger Drift Event:

```
kubectl exec -it fastapi-deployment-555f9fd4cb-sbv9z -n fastapi -- curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"drift_score": 0.5}' \
  http://drift-detection-eventsourcesvc.argo-events.svc.cluster.local:12000/dr
```

1. Check Event Logs:

```
kubectl get pods -n argo-events
kubectl -n argo-events logs drift-detection-eventsourcesvc-lgtsv-6f949fdcf7-znvh2
kubectl -n argo-events logs drift-detection-sensor-sensor-kwdvx-bbbc5db57-qd659
```

1. Simulate Drift via FastAPI:

```
curl -X POST "http://localhost:8000/simulate-drift" \
  -H "Content-Type: application/json" \
  -d '{"drift_type": "numerical_shift"}'
```

Final Thoughts

This guide demonstrates how to build an automated pipeline for model drift detection and retraining using GitOps principles. By combining Kubernetes-native tools like Argo Workflows, Events, and MinIO, you can ensure your machine learning workflows are scalable, reliable, and efficient.

Check out the complete repository here:

[GitHub Repository Link](#)

If you enjoyed this article or have questions, feel free to connect with me on [LinkedIn](#) or drop a comment below. Let's build smarter, together! 🚀



MongoDB

PROMOTED



A dark-themed advertisement for MongoDB. On the left, the MongoDB logo is at the top, followed by the text "Join the AI Revolution" in a large, white, sans-serif font. Below this is a bright green rectangular button with the text "Start Building" in white. On the right, there is a stylized representation of a code editor window with a light blue header bar and three green window control dots. Inside the editor, a MongoDB query is written in a light green monospace font. The query filters for documents where 'fullyManaged' is true, 'security' is 'built-in', and 'cloud' is in an array containing 'AWS', 'Azure', and 'GC'. It also sorts by '_id' in descending order and uses a cursor to print the JSON of each document. A comment at the bottom of the code says 'try: MongoDB Atlas Today'.

[Gen AI apps are built with MongoDB Atlas](#)

MongoDB Atlas is the developer-friendly database for building, scaling, and running gen AI & LLM apps—no separate vector DB needed. Enjoy native vector search, 115+ regions, and flexible document modeling. Build AI faster, all in one place.

[Start Free](#)

Top comments (0)

[Code of Conduct](#) • [Report abuse](#)



Heroku PROMOTED





[Tired of jumping between terminals, dashboards, and code?](#)

Check out this demo showcasing how tools like Cursor can connect to Heroku through the MCP, letting you trigger actions like deployments, scaling, or provisioning—all without leaving your editor.

[Learn More](#)



Michael Tyiska

Building next-gen software solutions with a focus on DevOps, Kubernetes, and cutting-edge AI/ML tools. 🚀 Passionate about automation and scalable architectures.

LOCATION

Seattle Washington

EDUCATION

University of Houston

WORK

Freelancer specializing in DevOps, full-stack development, and AI/ML solutions.

JOINED

Mar 5, 2022

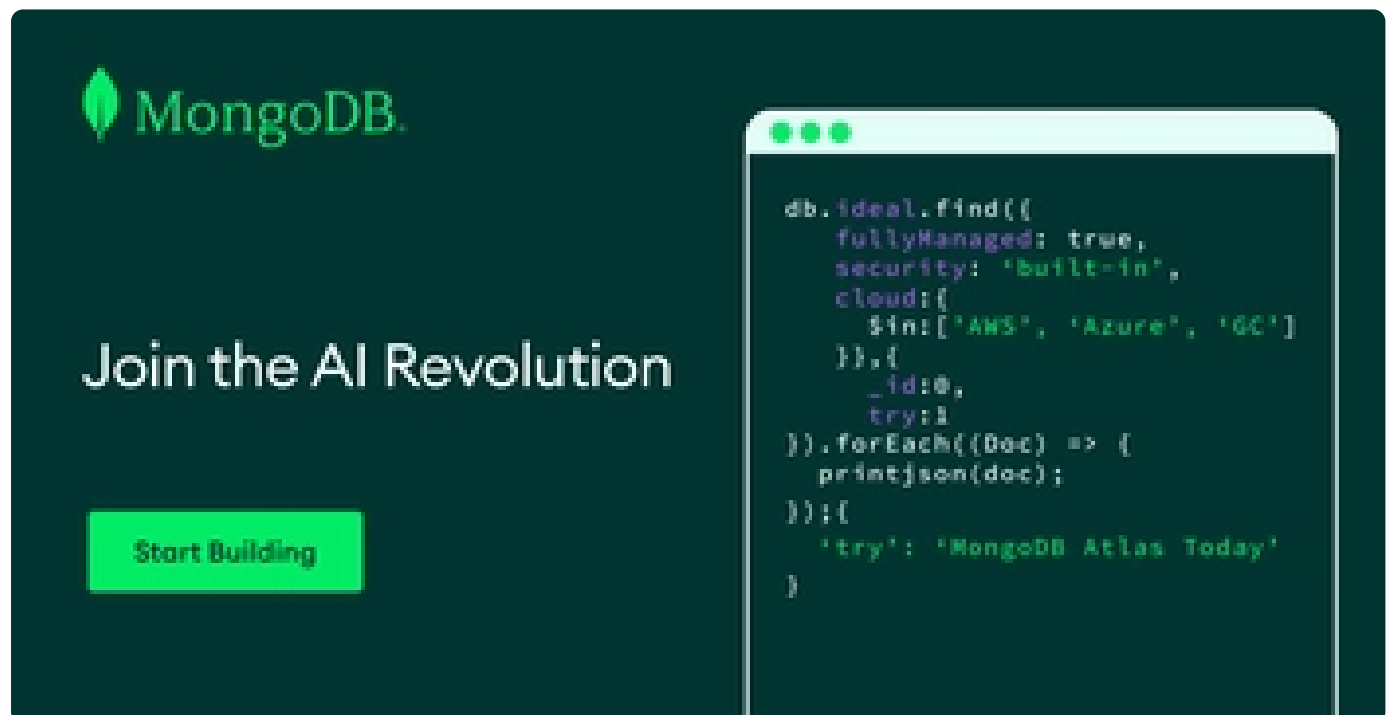
More from [Michael Tyiska](#)

Automating CI/CD Pipelines for Kubernetes with Argo Rollouts, Argo CD, Argo Workflow & Events

#devops #kubernetes #argoworkflows #gitops

 MongoDB PROMOTED

...



The advertisement features the MongoDB logo in green on a dark teal background. Below the logo, the text "Join the AI Revolution" is displayed in white. A red button with the text "Start Building" is positioned below the text. To the right, a code editor window shows a MongoDB query in JavaScript:

```
db.ideal.find({
  fullyManaged: true,
  security: 'built-in',
  cloud: {
    $in: ['AWS', 'Azure', 'GCP']
  }, {
    _id: 0,
    try: 1
  }).forEach((doc) => {
    printjson(doc);
  }); {
    'try': 'MongoDB Atlas Today'
  }
```

[Gen AI apps are built with MongoDB Atlas](#)

MongoDB Atlas is the developer-friendly database for building, scaling, and running gen AI & LLM apps—no separate vector DB needed. Enjoy native vector search, 115+ regions, and flexible document modeling. Build AI faster, all in one place.

[Start Free](#)
