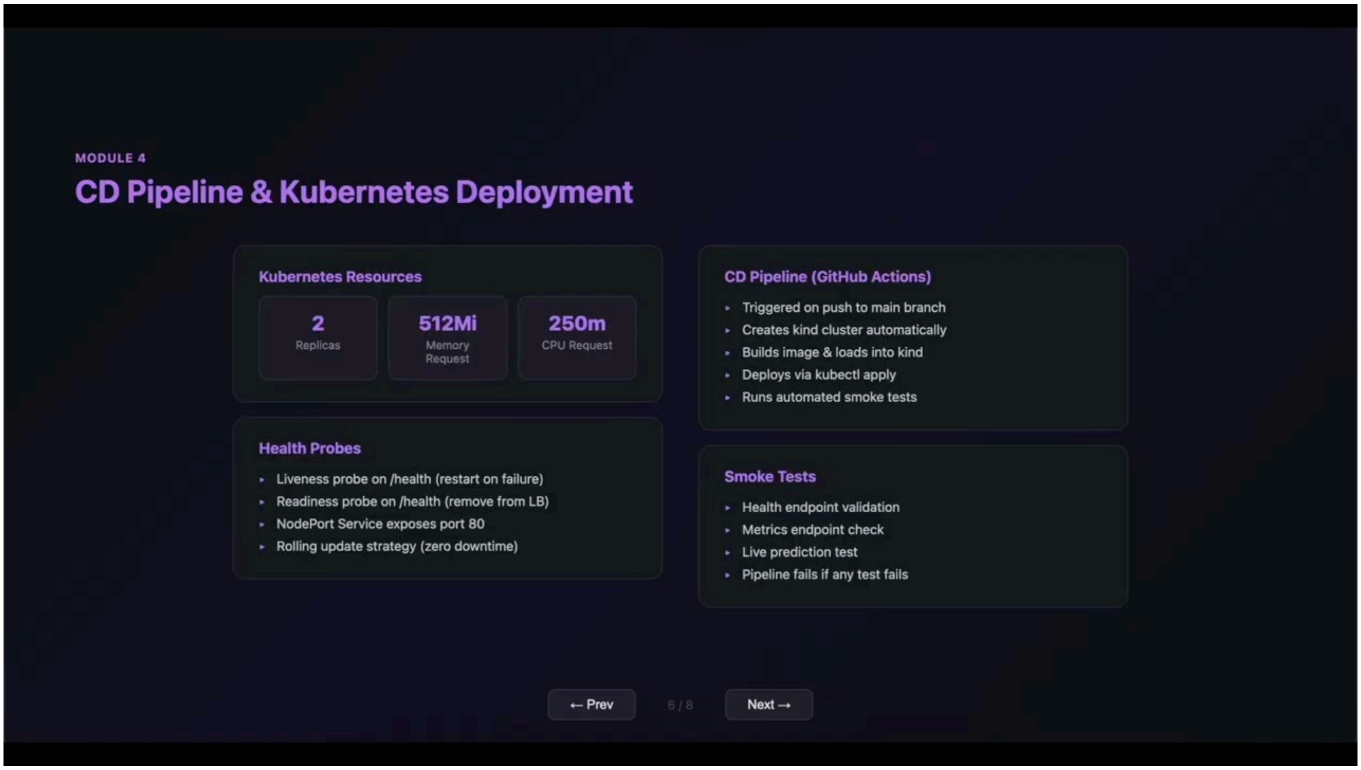
# MLOps Assignment 2: Cats vs Dogs Classification Pipeline

End-to-end MLOps pipeline for binary image classification (Cats vs Dogs) for a pet adoption platform.

# MLOPS Assignment 2

🔗 Repository: https://github.com/ps2program/MLOPS_ASSIGNMENT_2.git

# 🎬 Project Demo



MODULE 4

CD Pipeline & Kubernetes Deployment

**Kubernetes Resources**

| 2 | 512Mi | 250m |
|---|---|---|
| Replicas | Memory Request | CPU Request |

**CD Pipeline (GitHub Actions)**

- Triggered on push to main branch
- Creates kind cluster automatically
- Builds image & loads into kind
- Deploys via kubectl apply
- Runs automated smoke tests

**Health Probes**

- Liveness probe on /health (restart on failure)
- Readiness probe on /health (remove from LB)
- NodePort Service exposes port 80
- Rolling update strategy (zero downtime)

**Smoke Tests**

- Health endpoint validation
- Metrics endpoint check
- Live prediction test
- Pipeline fails if any test fails

▶️ Watch the Full End-to-End MLOps Demo

# Project Structure

```
.
├── data/                  # Data directory (tracked by DVC)
│   ├── raw/              # Raw dataset
│   ├── processed/        # Preprocessed data
│   └── .gitignore
├── src/                   # Source code
│   ├── data/             # Data processing scripts
│   ├── models/           # Model definitions
│   ├── training/         # Training scripts
│   └── inference/       # Inference service
├── tests/                # Unit tests
├── notebooks/            # Jupyter notebooks for exploration
├── mlruns/               # MLflow experiment tracking
├── deployment/           # Deployment manifests
│   ├── kubernetes/      # K8s manifests
│   └── docker-compose/  # Docker Compose config
├── .github/
│   └── workflows/       # GitHub Actions CI/CD
├── Dockerfile           # Container image definition
├── requirements.txt     # Python dependencies
├── .dvcignore          # DVC ignore patterns
├── .gitignore          # Git ignore patterns
└── README.md           # This file
```

# Setup Instructions

1. **Clone and initialize:**

```
git clone https://github.com/ps2program/MLOPS_ASSIGNMENT_2.git
cd MLOPS_ASSIGNMENT_2
dvc init
```

2. **Install dependencies:**

```
pip install -r requirements.txt
```

3. **Download dataset:**
   - Dataset: bhavikjikadara/dog-and-cat-classification-dataset
   - Total: 25,038 images (12,519 cats, 12,519 dogs)
   - Use Kaggle CLI:

```
kaggle datasets download -d bhavikjikadara/dog-and-cat-classification-dataset -p data/raw --unzip
```

   - Or use the script: `./scripts/download_dataset.sh` (after setting up Kaggle credentials)
   - Run `dvc add data/raw/` to track with DVC

4. **Train model:**

```
python src/training/train.py
```

### Or Download Pre-trained Model Artifacts

To download the trained model files from Google Drive:

```
chmod +x download_drive_folder.sh
./download_drive_folder.sh
```

5. **Run inference service:**

```
docker build -t cats-dogs-classifier .
docker run -p 8000:8000 cats-dogs-classifier
```

## Dataset

- **Source:** bhavikjikadara/dog-and-cat-classification-dataset
- **Size:** 25,038 images (12,519 cats, 12,519 dogs)
- **Tracked with:** DVC (Data Version Control)
- See `DATASET_INFO.md` for detailed information

## Modules

- **M1**: Model Development & Experiment Tracking (✅ Complete)
- **M2**: Model Packaging & Containerization (✅ Complete)
- **M3**: CI Pipeline for Build, Test & Image Creation (✅ Complete)
- **M4**: CD Pipeline & Deployment (✅ Complete)
- **M5**: Monitoring, Logs & Final Submission (✅ Complete)

## Repository

- **GitHub:** https://github.com/ps2program/MLOPS_ASSIGNMENT_2
- **Status:** All modules implemented and tested

# Setup Guide

This guide will help you set up and run the complete MLOps pipeline for Cats vs Dogs classification.

## Prerequisites

- Python 3.10+ (tested with 3.10 and 3.13)

- Docker Desktop (for containerization)
- Git (for version control)
- kind (for local Kubernetes deployment)
- Kaggle account (for dataset download)

# Step 1: Clone and Initialize Repository

```
# Clone the repository {#clone-the-repository }
git clone https://github.com/ps2program/MLOPS_ASSIGNMENT_2.git
cd MLOPS_ASSIGNMENT_2

# DVC is already initialized in this repo {#dvc-is-already-initialized-in-this-repo }
```

# Step 2: Install Dependencies

```
# Create virtual environment (recommended) {#create-virtual-environment-recommended }
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install Python packages {#install-python-packages }
pip install -r requirements.txt
```

# Step 3: Download Dataset

## Option 1: Using Kaggle CLI (Recommended)

**Dataset:** bhavikjikadara/dog-and-cat-classification-dataset

- **Total Images:** 25,038 (12,519 cats, 12,519 dogs)
- **Size:** ~775MB

1. Download your Kaggle API credentials from https://www.kaggle.com/settings
2. Place `kaggle.json` in `~/.kaggle/`
3. Run the download script:

```
# Update the script to use the correct dataset, or run directly: {#update-the-script-to-use-the-correct-
kaggle datasets download -d bhavikjikadara/dog-and-cat-classification-dataset -p data/raw --unzip

# Organize the dataset (if needed) {#organize-the-dataset-if-needed }
# The script will handle organization automatically {#the-script-will-handle-organization-automatically
```

## Option 2: Manual Download

1. Download from: https://www.kaggle.com/datasets/bhavikjikadara/dog-and-cat-classification-dataset

2. Extract and organize into:

    ```
    data/raw/
      cats/
        *.jpg  (12,519 images)
      dogs/
        *.jpg  (12,519 images)
    ```

## Track Data with DVC

```
# Add data to DVC {#add-data-to-dvc }
dvc add data/raw

# Commit DVC files {#commit-dvc-files }
git add data/raw.dvc .gitignore
git commit -m "Add dataset with DVC"
```

# Step 4: Train the Model

> **Important:** You must set `PYTHONPATH` to the project root so that `src` is importable.

```
# Train with default parameters {#train-with-default-parameters }
PYTHONPATH=. python src/training/train.py

# Or with custom parameters {#or-with-custom-parameters }
PYTHONPATH=. python src/training/train.py \
    --raw_data_dir data/raw \
    --processed_data_dir data/processed \
    --model_save_dir models \
    --num_epochs 10 \
    --batch_size 32 \
    --learning_rate 0.001
```

The training script will:

- Preprocess images (resize to 224x224, split 80/10/10 train/val/test)
- Apply data augmentation (random horizontal flip, rotation, color jitter)
- Train a CNN model with batch normalization and dropout
- Track experiments with MLflow (parameters, metrics, confusion matrix)
- Save the best model to `models/best_model.pt`

## View Training Results in MLflow

```
mlflow ui --port 5000
# Open http://localhost:5000 {#open-httplocalhost5000 }
```

Click the **cats_dogs_classification** experiment, then click the run to see:

- **Parameters:** epochs, batch_size, learning_rate
- **Metrics:** train/val accuracy, loss, precision, recall, F1
- **Artifacts:** confusion_matrix.png, saved model

## Step 5: Run Tests

```
# Run unit tests (14 tests) {#run-unit-tests-14-tests }
PYTHONPATH=. pytest tests/ -v

# Run with coverage report {#run-with-coverage-report }
PYTHONPATH=. pytest tests/ -v --cov=src --cov-report=html
```

## Step 6: Build Docker Image

The model ( `models/best_model.pt` ) is baked into the image during build.

```
# Build the image (use --load if Docker sign-in enforcement is enabled) {#build-the-image-use--load-if-c
docker build --load -t cats-dogs-classifier:latest .

# Verify the image {#verify-the-image }
docker images | grep cats-dogs-classifier
```

## Step 7: Run Inference Service Locally

```
# Run with Docker (model is already inside the image) {#run-with-docker-model-is-already-inside-the-imag
docker run -d --name cats-dogs-api -p 8000:8000 cats-dogs-classifier:latest

# Or with Docker Compose (includes Prometheus) {#or-with-docker-compose-includes-prometheus }
cd deployment/docker-compose
docker compose up -d
cd ../..
```

# Step 8: Test the API

```
# Health check {#health-check }
curl http://localhost:8000/health
# Expected: {#expected  status="true" :"healthy"="true" model_loaded="true" :true="true"}

# Prediction with a cat image {#prediction-with-a-cat-image }
curl -X POST -F "file=@data/raw/cats/cat_000.jpg" http://localhost:8000/predict

# Prediction with a dog image {#prediction-with-a-dog-image }
curl -X POST -F "file=@data/raw/dogs/dog_000.jpg" http://localhost:8000/predict

# Prometheus metrics {#prometheus-metrics }
curl http://localhost:8000/metrics | grep inference

# Run smoke tests {#run-smoke-tests }
bash scripts/smoke_tests.sh http://localhost:8000

# Clean up {#clean-up }
docker stop cats-dogs-api && docker rm cats-dogs-api
```

# Step 9: CI/CD Setup

## GitHub Actions

The CI/CD pipelines are already configured in `.github/workflows/` :

- **CI Pipeline** ( `.github/workflows/ci.yml` ):
  - Triggers on push/PR to main/develop
  - Runs unit tests with pytest
  - Builds Docker image
  - Pushes to GitHub Container Registry
- **CD Pipeline** ( `.github/workflows/cd.yml` ):
  - Triggers on push to main
  - Provisions a kind cluster
  - Builds and loads the Docker image into kind
  - Deploys to Kubernetes via `kubectl apply`
  - Runs smoke tests
  - Cleans up the cluster

No additional secrets are required -- the CD pipeline uses kind for local-style deployment within the GitHub Actions runner.

# Step 10: Deploy to Local Kubernetes (kind)

## Option 1: Docker Compose (Local)

```
cd deployment/docker-compose
docker compose up -d

# Check status {#check-status }
docker compose ps

# View logs {#view-logs }
docker compose logs -f classifier-api

# Tear down {#tear-down }
docker compose down
cd ../..
```

## Option 2: Kubernetes with kind (Recommended)

```
# Install kind (if not already installed) {#install-kind-if-not-already-installed }
# macOS: {#macos }
brew install kind
# Linux: {#linux }
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
chmod +x ./kind && sudo mv ./kind /usr/local/bin/kind

# Create a cluster {#create-a-cluster }
kind create cluster --name mlops-cluster --wait 60s

# Build and load the Docker image into kind {#build-and-load-the-docker-image-into-kind }
docker build --load -t cats-dogs-classifier:latest .
kind load docker-image cats-dogs-classifier:latest --name mlops-cluster

# Deploy {#deploy }
kubectl apply -f deployment/kubernetes/deployment.yaml

# Wait for pods to be ready {#wait-for-pods-to-be-ready }
kubectl rollout status deployment/cats-dogs-classifier --timeout=120s

# Check status (expect 2/2 pods Running) {#check-status-expect-22-pods-running }
kubectl get pods,svc,deployment

# Port-forward to access the service {#port-forward-to-access-the-service }
kubectl port-forward svc/cats-dogs-classifier-service 8080:80 &

# Test health {#test-health }
curl http://localhost:8080/health

# Test prediction {#test-prediction }
curl -X POST -F "file=@data/raw/cats/cat_000.jpg" http://localhost:8080/predict

# Run smoke tests {#run-smoke-tests-1 }
bash scripts/smoke_tests.sh http://localhost:8080
```

## Tear Down Kubernetes

```
# Delete the cluster when done {#delete-the-cluster-when-done }
kind delete cluster --name mlops-cluster
```

# Step 11: Monitoring & Evaluation

## Prometheus Metrics

The API exposes Prometheus-compatible metrics at `/metrics` :

```
# View key metrics {#view-key-metrics }
curl http://localhost:8080/metrics | grep -E "(inference_requests_total |inference_request_duration|prec
```

Key metrics:

| Metric | Description |
|---|---|
| `inference_requests_total` | Total number of inference requests |
| `inference_request_duration_seconds` | Latency histogram (p50, p90, p99) |
| `predictions_total{class="cat/dog"}` | Prediction count by class |

## Prometheus Setup

**Docker Compose:**
Prometheus UI is automatically available at `http://localhost:9090` when using Docker Compose.

**Kubernetes:**
To deploy Prometheus with Kubernetes:

```
# Deploy Prometheus configuration and service {#deploy-prometheus-configuration-and-service }
kubectl apply -f deployment/kubernetes/prometheus-config.yaml
kubectl apply -f deployment/kubernetes/prometheus-deployment.yaml

# Port-forward to access Prometheus UI {#port-forward-to-access-prometheus-ui }
kubectl port-forward svc/prometheus-service 9090:9090

# Access Prometheus UI at http://localhost:9090 {#access-prometheus-ui-at-httplocalhost9090 }
```

Prometheus will automatically scrape metrics from your classifier service pods.

## Grafana Setup

**Kubernetes:**
To deploy Grafana for visualization:

```
# Deploy Grafana configuration and service {#deploy-grafana-configuration-and-service }
kubectl apply -f deployment/kubernetes/grafana-config.yaml
kubectl apply -f deployment/kubernetes/grafana-datasource.yaml
kubectl apply -f deployment/kubernetes/grafana-deployment.yaml

# Port-forward to access Grafana UI {#port-forward-to-access-grafana-ui }
kubectl port-forward svc/grafana-service 3000:3000

# Access Grafana UI at http://localhost:3000 {#access-grafana-ui-at-httplocalhost3000 }
# Login: admin / admin {#login-admin--admin }
```

Grafana is pre-configured to connect to Prometheus. You can create dashboards to visualize:

- Total inference requests
- Request rate over time
- Latency percentiles (p50, p90, p99)
- Prediction distribution by class

See `deployment/kubernetes/GRAFANA_SETUP.md` for detailed instructions.

## Application Logs

```
# Docker {#docker }
docker logs cats-dogs-api

# Docker Compose {#docker-compose }
docker compose logs -f classifier-api

# Kubernetes {#kubernetes }
kubectl logs deployment/cats-dogs-classifier --tail=20
```

Logs include structured prediction entries with confidence and latency:

```
2026-02-11 08:25:20 - src.inference.app - INFO - Prediction: dog, Confidence: 0.9895, Latency: 0.7850s
```

## Post-Deployment Model Evaluation

Run the evaluation script to assess model performance on real images:

```
python scripts/evaluate_deployed_model.py --url http://localhost:8080 --num-samples 20
```

This sends images with known labels (from `data/raw/cats/` and `data/raw/dogs/` ) to the deployed API and reports accuracy, precision, recall, F1 score, and a confusion matrix. If real images are not available, it falls back to simulated test images.

## MLflow UI

View experiment tracking:

```
mlflow ui --port 5000
# Open http://localhost:5000 {#open-httplocalhost5000-1 }
```

# Troubleshooting

### `ModuleNotFoundError: No module named 'src'`

You need `PYTHONPATH` set to the project root:

```
PYTHONPATH=. python src/training/train.py
PYTHONPATH=. pytest tests/ -v
```

## Model not found error

Ensure the model file exists:

```
ls -la models/best_model.pt
```

If missing, train the model first (Step 4).

## Docker sign-in enforcement error

If `docker build` fails with a sign-in error, use the `--load` flag:

```
docker build --load -t cats-dogs-classifier:latest .
```

## Port already in use

Change the host port:

- Docker: `-p 8001:8000`
- Docker Compose: Update `ports` in `docker-compose.yml`
- Kubernetes port-forward: `kubectl port-forward svc/cats-dogs-classifier-service 8081:80`

## DVC issues

If DVC cache is corrupted:

```
dvc cache dir
# Clear and re-add data {#clear-and-re-add-data }
dvc remove data/raw.dvc
dvc add data/raw
```

## kind cluster issues

```
# Check if cluster exists {#check-if-cluster-exists }
kind get clusters

# Delete and recreate if broken {#delete-and-recreate-if-broken }
kind delete cluster --name mlops-cluster
kind create cluster --name mlops-cluster --wait 60s
```

# Support

For issues or questions, please open an issue in the repository.