Email/SMS Spam Classifier - MLOps Implementation Report

Course: CSE 816 - Software Production Engineering

Pranav Kulkarni - IMT2022053

Abhik Kumar - IMT2022117

GitHub Repository: https://github.com/Abhik-04/SPE_Final_Project

Dockerhub Links: https://hub.docker.com/repository/docker/abhikkumar04/spam-classifier/general

Dockerhub Links: https://hub.docker.com/repository/docker/abhikkumar04/mlflow-server/general

---

1. Project Overview

1.1 Project Summary

This project implements a **complete MLOps pipeline** for an Email/SMS spam classification system. The implementation demonstrates:

- **Automated CI/CD**: From code commit to production deployment
- **Container Orchestration**: Kubernetes-based scaling and management
- **ML Lifecycle Management**: Experiment tracking and model versioning with MLflow
- **Production Monitoring**: Centralized logging with ELK Stack
- **Security**: Secrets management with HashiCorp Vault
- **Zero-Downtime Deployments**: Rolling updates with health checks

1.2 Application Details

What the Application Does:
- Classifies text messages as **spam** or **not spam** using machine learning
- Provides a user-friendly **Streamlit web interface** for predictions

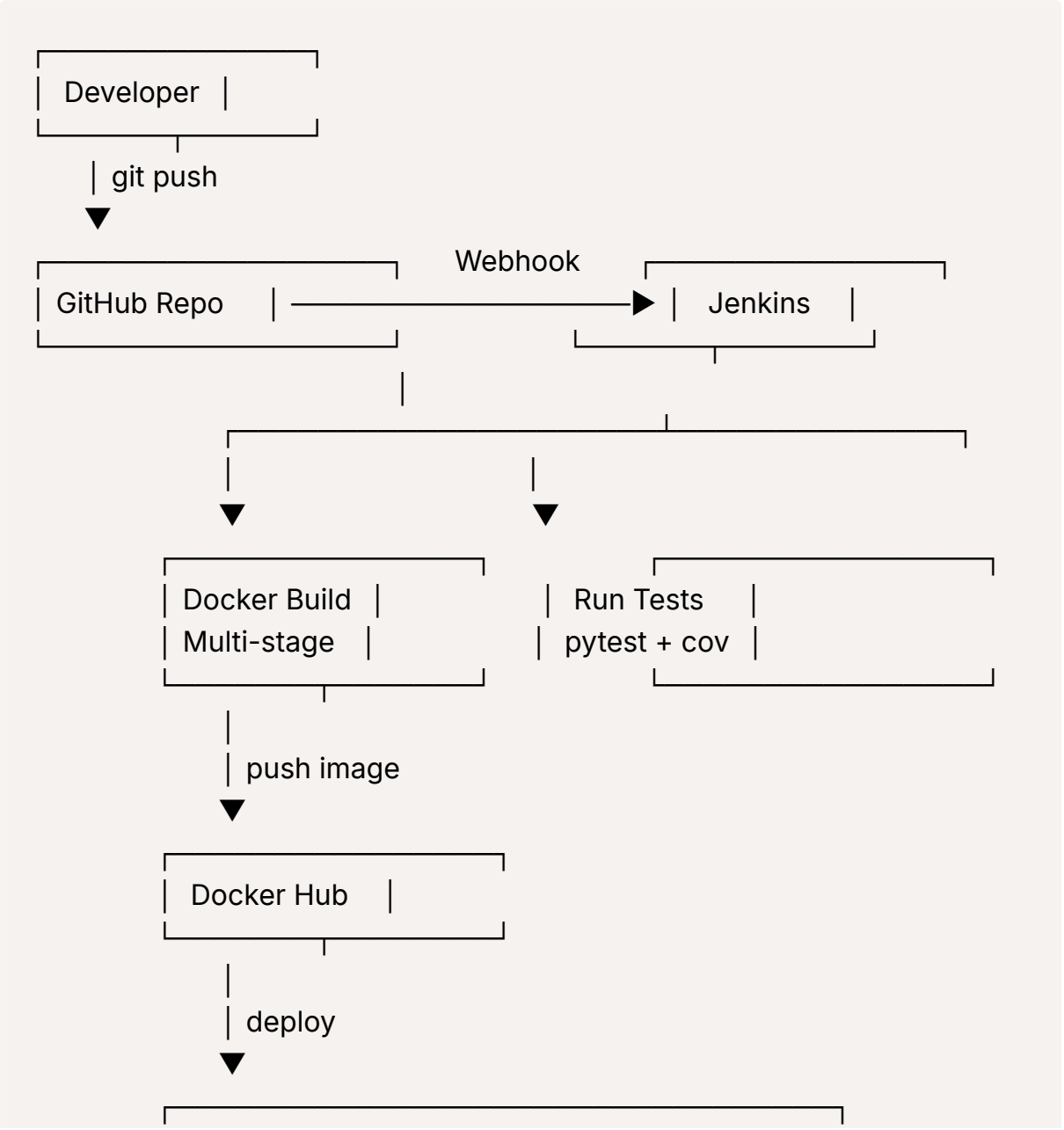
- Offers a **Flask REST API** for programmatic access
- Logs all predictions with confidence scores for analysis

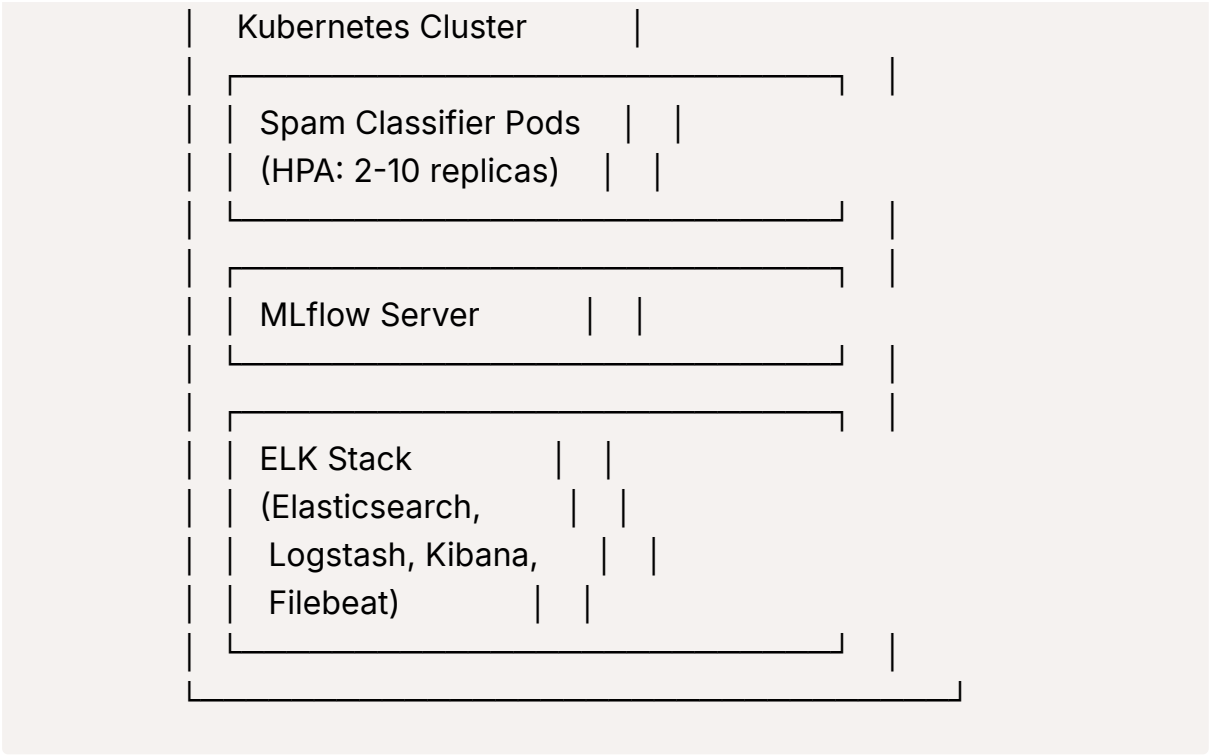
ML Model:
- **Algorithm**: Naive Bayes (Multinomial)
- **Feature Extraction**: TF-IDF Vectorization (3000 features)
- **Accuracy**: 97.58%
- **Dataset**: SMS Spam Collection (5,574 messages)

---

2. Architecture & Technology Stack

2.1 High-Level Architecture





2.2 Technology Stack

Component	Technology	Purpose
ML Framework	scikit-learn, NLTK	Model training and text processing
Application	Streamlit, Flask	User interface and REST API
Version Control	Git, GitHub	Source code management
CI/CD	Jenkins	Automated pipeline (12 stages)
Containerization	Docker	Application packaging
Orchestration	Kubernetes (Minikube)	Container management, scaling
Config Management	Ansible	Infrastructure automation (4 roles)
Monitoring	ELK Stack	Centralized logging and visualization
MLOps	MLflow	Experiment tracking, model registry
Secrets Management	HashiCorp Vault	Secure credential storage
Testing	pytest, pytest-cov	Unit and integration tests (82% coverage)

3. Implementation Flow

This section walks through the complete implementation process, step by step, as it was actually built.

Phase 1: Docker - Containerization

Objective: Package the application into Docker containers for consistent deployment across environments.

Step 1.1: Create Multi-Stage Dockerfile

The project uses a **multi-stage Dockerfile** to optimize image size and security:

Features:
- Non-root user execution (UID 1000)
- Two-stage build (reduces image size by ~60%)
- Health checks configured
- Minimal base image

Step 1.2: Build and Push Docker Images

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % docker push ${DOCKERHUB_USERNAME}/spam-classifier:latest
The push refers to repository [docker.io/abhikkumar04/spam-classifier]
5fc74dffc503: Pushed
676f4959b3e4: Pushed
21706f6a4c91: Pushed
faa8861819f0: Pushed
913c05e4a1c0: Pushed
7cfbba4dec61: Pushed
3325bfc50aa4: Pushed
52a7c81316e6: Pushed
7bf6e8047453: Pushed
f626fba1463b: Pushed
b4684b769b98: Pushed
016c5c475990: Pushed
latest: digest: sha256:5ec30257d97f20248f99fc70b60434fab2890a3da484a650af4b2156fbcb5d97 size: 856
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % docker push ${DOCKERHUB_USERNAME}/mlflow-server:latest
The push refers to repository [docker.io/abhikkumar04/mlflow-server]
5f019232328b: Pushed
7bf6e8047453: Mounted from abhikkumar04/spam-classifier
5695c90786c6: Pushed
f626fba1463b: Mounted from abhikkumar04/spam-classifier
c5e8bbf9c60a: Pushed
faa8861819f0: Mounted from abhikkumar04/spam-classifier
913c05e4a1c0: Mounted from abhikkumar04/spam-classifier
284c3775b37f: Pushed
latest: digest: sha256:e462f1fbf09c4d9a54d078e0c013f261bd4484946e8f33ed88b09ca997fdc14c size: 856
```

Figure 1.1: Built Docker images successfully pushed to Docker Hub registry

Images Created:
1. `spam-classifier:latest` - Main application image
2. `mlflow-server:latest` - MLflow tracking server image

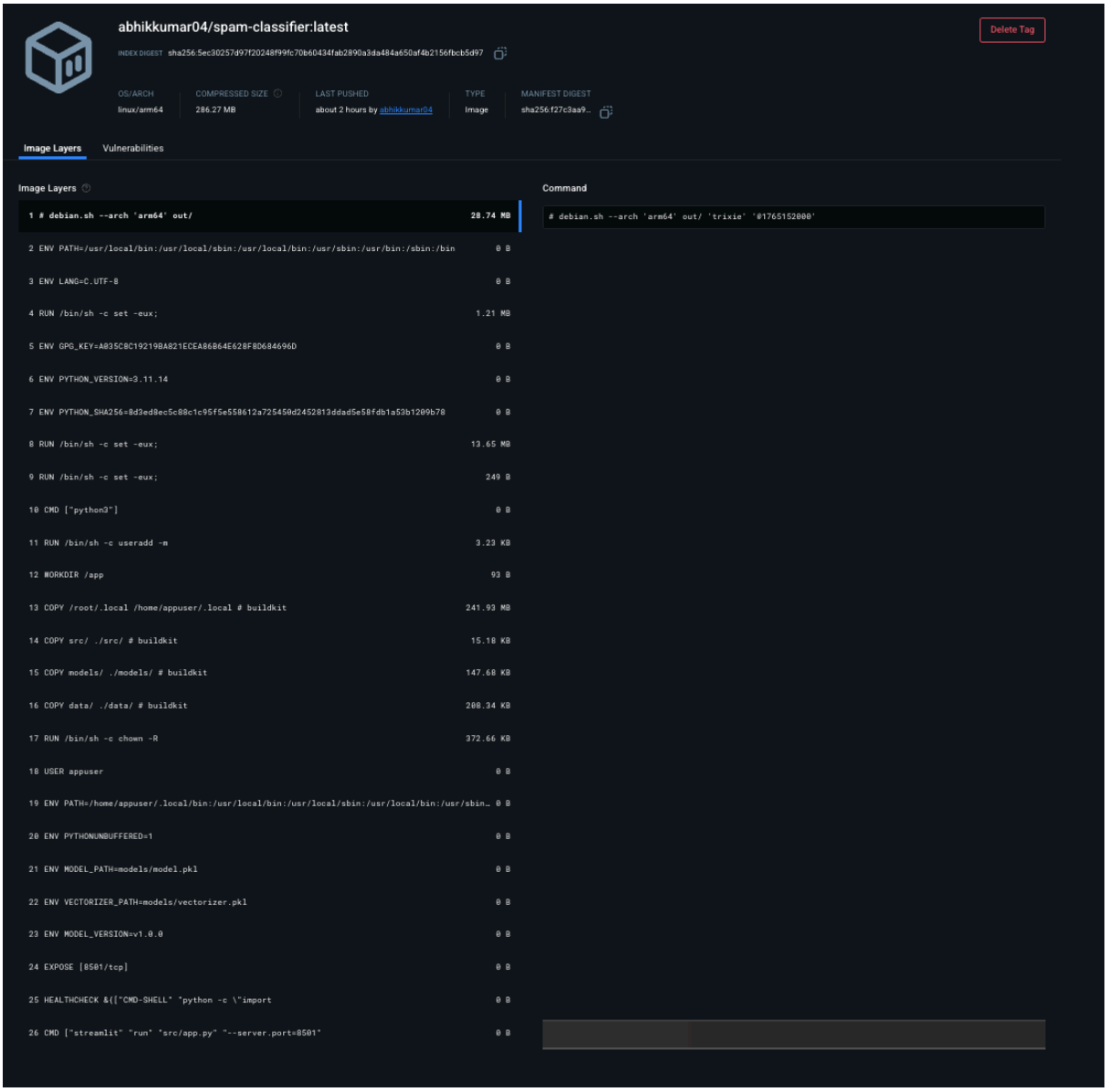


Figure 1.2: Spam classifier Docker image details

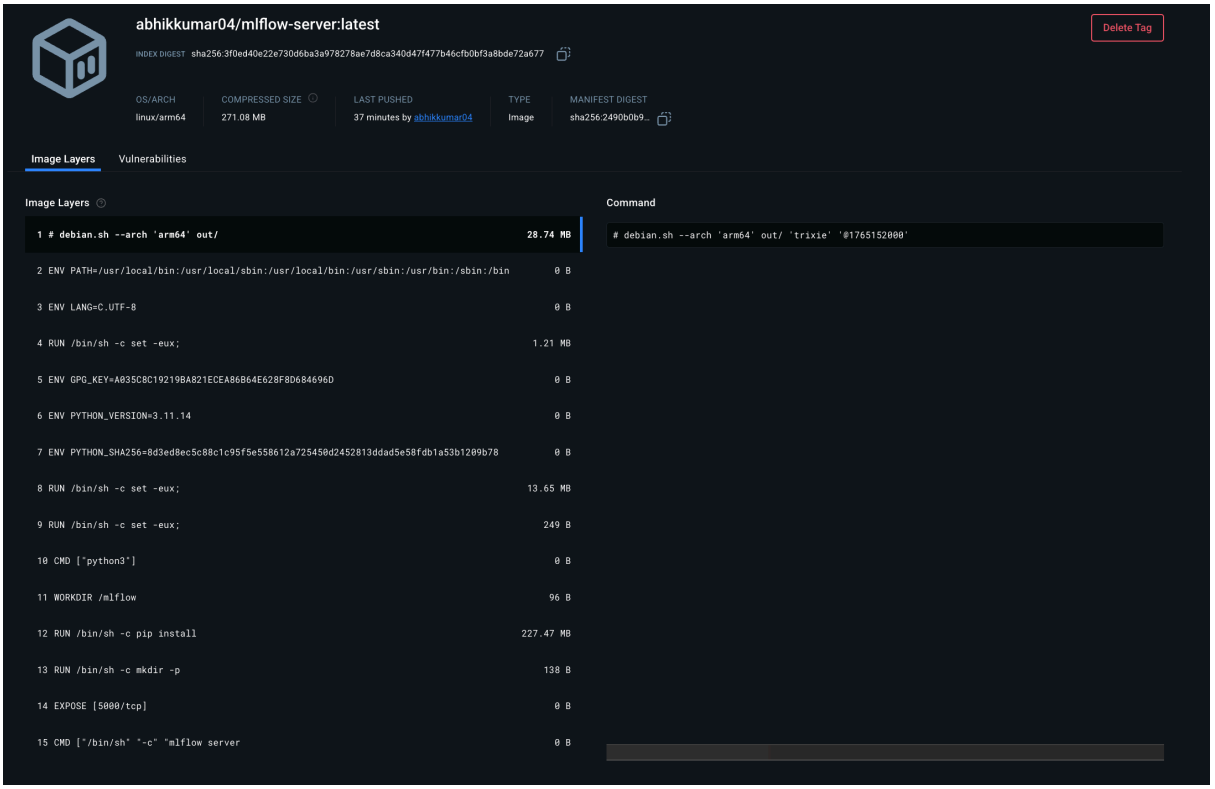


Figure 1.3: MLflow server Docker image

Step 1.3: Test Locally with Docker Compose

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % docker run -d -p 8888:8888 --name spam-test ${DOCKERHUB_USERNAME}/spam-cl
assifier:latest
5bc0f812a77cb25051028458b79131e68ef29dcf28cb9e6af073126f30c11eb9
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % docker logs spam-test

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.


  You can now view your Streamlit app in your browser.

  URL: http://0.0.0.0:8501

(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % curl http://localhost:8888/_stcore/health
Handling connection for 8888
ok%
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % docker stop spam-test
docker rm spam-test
spam-test
spam-test
```

Figure 1.4: Application running locally via Docker Compose for validation

---

Phase 2: Kubernetes - Infrastructure Setup

Objective: Deploy the application to Kubernetes with auto-scaling, health checks, and proper resource management.

Step 2.1: Create Namespace and ConfigMaps

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/namespace.yaml
namespace/spam-classifier created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/configmap.yaml
configmap/spam-classifier-config created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/secrets.yaml
secret/spam-classifier-secrets created
```

Figure 2.1: Creating Kubernetes namespace and configuration resources

What was created:
- **Namespace**: `spam-classifier` (isolated environment)
- **ConfigMaps**: Application configuration (model paths, environment variables)
- **Secrets**: Docker Hub credentials, API keys (later managed by Vault)

Step 2.2: Deploy MLflow Tracking Server

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/mlflow-deployment.yaml
deployment.apps/mlflow created
service/mlflow-service created
persistentvolumeclaim/mlflow-pvc created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier
NAME                     READY   STATUS            RESTARTS   AGE
mlflow-cb7787447-klrz7   0/1     PodInitializing   0          6s
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier
NAME                     READY   STATUS            RESTARTS   AGE
mlflow-cb7787447-klrz7   0/1     PodInitializing   0          8s
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier
NAME                     READY   STATUS    RESTARTS   AGE
mlflow-cb7787447-klrz7   1/1     Running   0          12s
```

Figure 2.2: MLflow tracking server deployed to Kubernetes

MLflow Setup:
- Persistent volume for experiment data
- Service exposure on port 5000
- Backend store for metrics and parameters
- Artifact storage for models

Step 2.3: Verify Deployment

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier
kubectl get svc -n spam-classifier
NAME                                READY   STATUS    RESTARTS   AGE
elasticsearch-0                     1/1     Running   0          5h2m
filebeat-w2dfr                      1/1     Running   0          4h57m
kibana-549f8b8b45-t7hj7             1/1     Running   0          4h59m
logstash-c796ddb6c-dxswf            1/1     Running   0          5h
mlflow-86c5759c59-2rnl7             1/1     Running   0          3h
spam-classifier-6464df77c7-wq8vs    1/1     Running   0          19m
spam-classifier-6464df77c7-znqr4    1/1     Running   0          20m
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
elasticsearch           ClusterIP      10.97.133.227    <none>        9200/TCP,9300/TCP   5h2m
kibana                  LoadBalancer   10.102.255.172   <pending>     5601:32050/TCP      4h59m
logstash                ClusterIP      10.99.20.13      <none>        5044/TCP            5h
mlflow-service          ClusterIP      10.109.216.136   <none>        5000/TCP            5h22m
spam-classifier-service LoadBalancer   10.102.213.6     <pending>     80:30197/TCP        3h
```

Figure 2.3: All deployments, services, and HPA verified as running

Verification Steps:

Step 2.4: Access MLflow UI

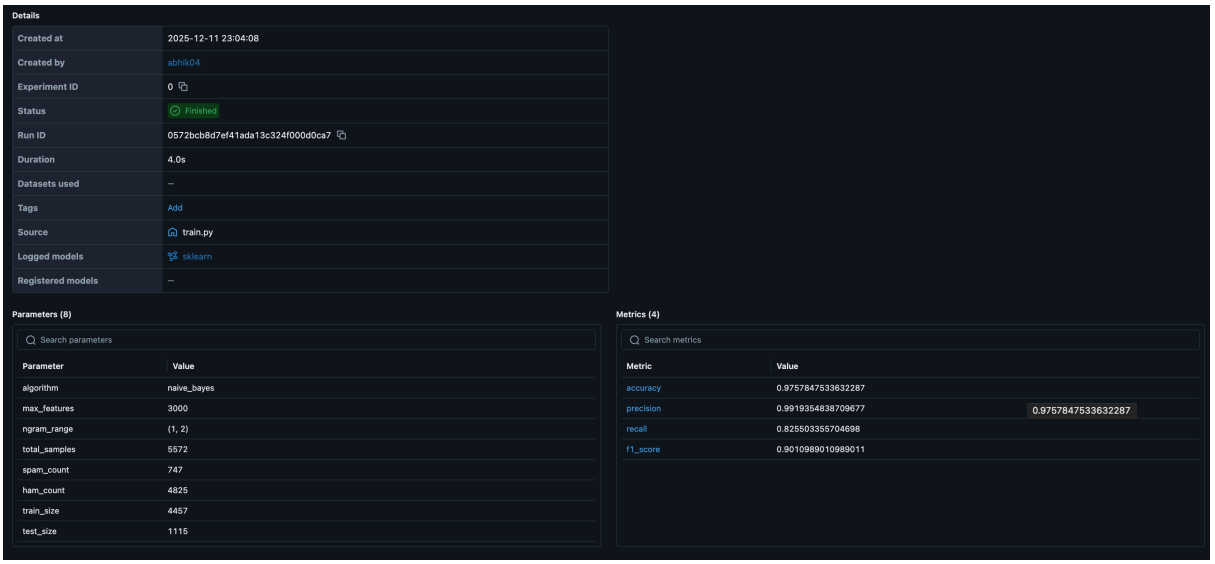


Figure 2.4: MLflow web interface accessible and tracking experiments

---

Phase 3: ELK Stack - Monitoring & Logging

Objective: Set up centralized logging infrastructure to collect, process, and visualize all application logs.

Step 3.1: Deploy Elasticsearch

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/elk/elasticsearch.yaml
statefulset.apps/elasticsearch created
service/elasticsearch created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl wait --for=condition=ready pod -l app=elasticsearch -n spam-class
ifier --timeout=600s
pod/elasticsearch-0 condition met
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier | grep elasticsearch
elasticsearch-0          1/1     Running   0          35s
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier | grep elasticsearch
elasticsearch-0          1/1     Running   0          45s
```

Figure 3.1: Elasticsearch pods running for log storage and indexing

Step 3.2: Deploy Logstash

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/elk/logstash.yaml
configmap/logstash-config created
deployment.apps/logstash created
service/logstash created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl wait --for=condition=ready pod -l app=logstash -n spam-classifier
 --timeout=300s
pod/logstash-c796ddb6c-dxswf condition met
```

Figure 3.2: Logstash pipeline processing and transforming logs

Logstash Configuration:
- **Input**: Receives logs from Filebeat (port 5044)
- **Filter**: Parses JSON logs, extracts fields
- **Output**: Sends to Elasticsearch with index pattern `spam-classifier-logs-*`

Step 3.3: Deploy Kibana

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/elk/logstash.yaml
configmap/logstash-config created
deployment.apps/logstash created
service/logstash created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl wait --for=condition=ready pod -l app=logstash -n spam-classifier
 --timeout=300s
pod/logstash-c796ddb6c-dxswf condition met
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/elk/kibana.yaml
deployment.apps/kibana created
service/kibana created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl wait --for=condition=ready pod -l app=kibana -n spam-classifier -
-timeout=300s
pod/kibana-549f8b8b45-t7hj7 condition met
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get svc kibana -n spam-classifier
NAME     TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kibana   LoadBalancer   10.102.255.172   <pending>     5601:32050/TCP   59s
```

Figure 3.3: Kibana service ready for log visualization

Kibana Features:
- Web UI on port 5601
- Data visualization dashboards
- Log search and filtering
- Real-time log streaming

Step 3.4: Deploy Filebeat DaemonSet

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/elk/filebeat-daemonset.yaml
configmap/filebeat-config created
daemonset.apps/filebeat created
serviceaccount/filebeat created
clusterrole.rbac.authorization.k8s.io/filebeat created
clusterrolebinding.rbac.authorization.k8s.io/filebeat created
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get daemonset -n spam-classifier
NAME       DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
filebeat   1         1         0       1            0           <none>          6s
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get daemonset -n spam-classifier
NAME       DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
filebeat   1         1         1       1            1           <none>          14s
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get daemonset -n spam-classifier
NAME       DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
filebeat   1         1         1       1            1           <none>          19s
```

Figure 3.4: Filebeat DaemonSet collecting logs from all Kubernetes nodes

Filebeat Configuration:
- **Deployment**: DaemonSet (runs on every node)
- **Collection**: Scrapes container logs from `/var/log/containers/*.log`
- **Forwarding**: Sends to Logstash for processing

Phase 4: MLflow - ML Lifecycle Management

Objective: Track machine learning experiments, manage model versions, and maintain reproducibility.

Step 4.1: MLflow Server Setup

A MLflow server was setup with given configuration for tracking etc purposes.

MLflow Configuration:
- **Tracking URI**: `http://localhost:5000`
- **Backend Store**: SQLite database for metadata
- **Artifact Location**: Local filesystem (in production: S3/GCS)
- **Experiments**: `spam-classifier-training`

Step 4.2: Model Training with MLflow

The training script (`src/train.py`) integrates with MLflow to track:

Parameters Logged:
- Algorithm type (naive_bayes, random_forest, svm)
- Max TF-IDF features (3000)
- N-gram range (1, 2)
- Dataset size and distribution

Metrics Logged:
- Accuracy: 97.58%
- Precision: 99.19%
- Recall: 82.55%
- F1-Score: 90.11%

Artifacts Saved:
- Trained model (`model.pkl`)
- TF-IDF vectorizer (`vectorizer.pkl`)
- Confusion matrix visualization
- Training statistics JSON

Training Command:

```
export MLFLOW_TRACKING_URI=http://localhost:5000
python -m src.train --algorithm naive_bayes --max-features 3000
```

Phase 5: GitHub - Version Control & Webhooks

Objective: Set up source code repository and configure webhooks to trigger automated deployments.

Step 5.1: Create GitHub Repository

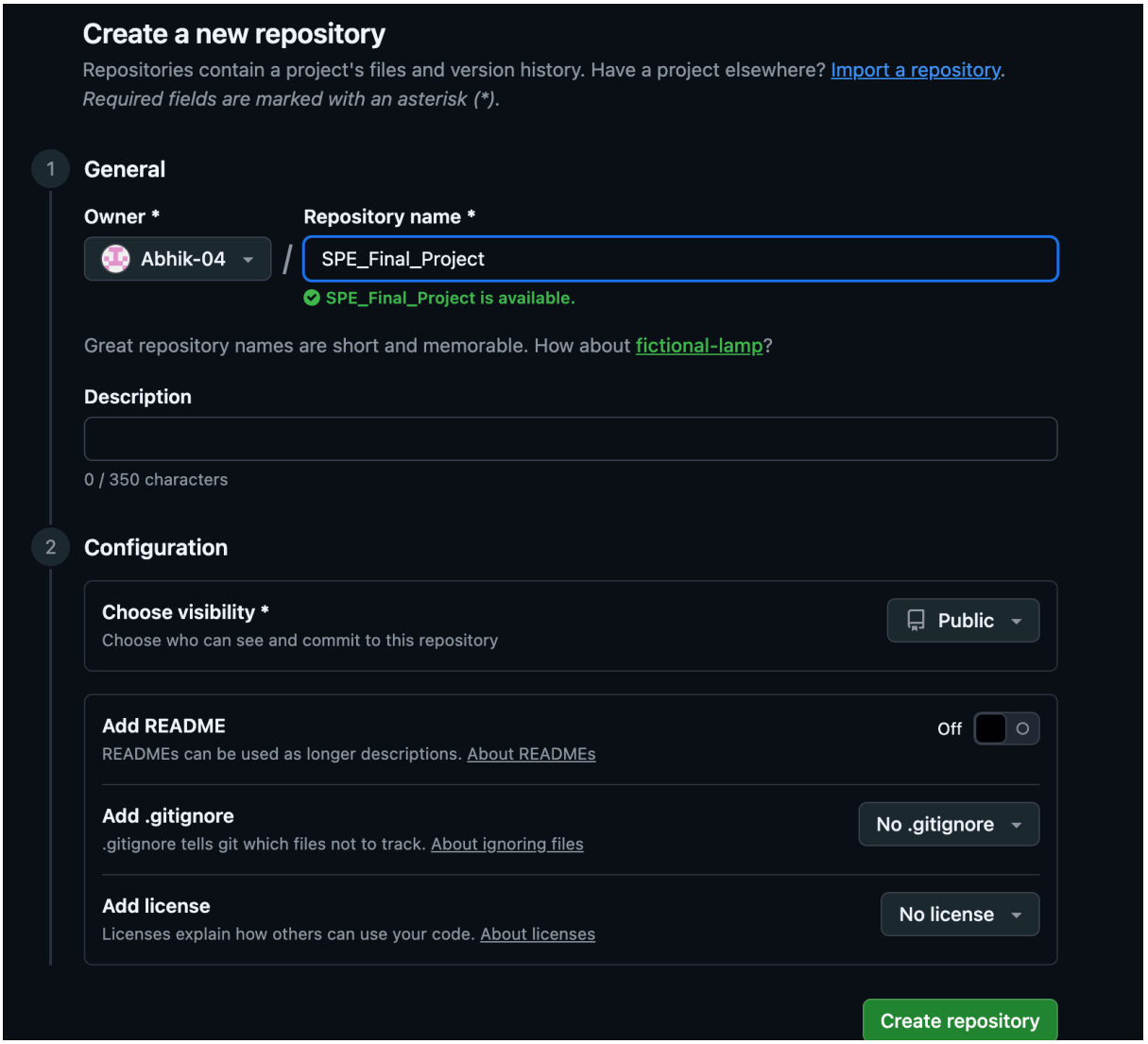


Figure 5.1: GitHub repository initialized for the project

Step 5.2: Push Project Files

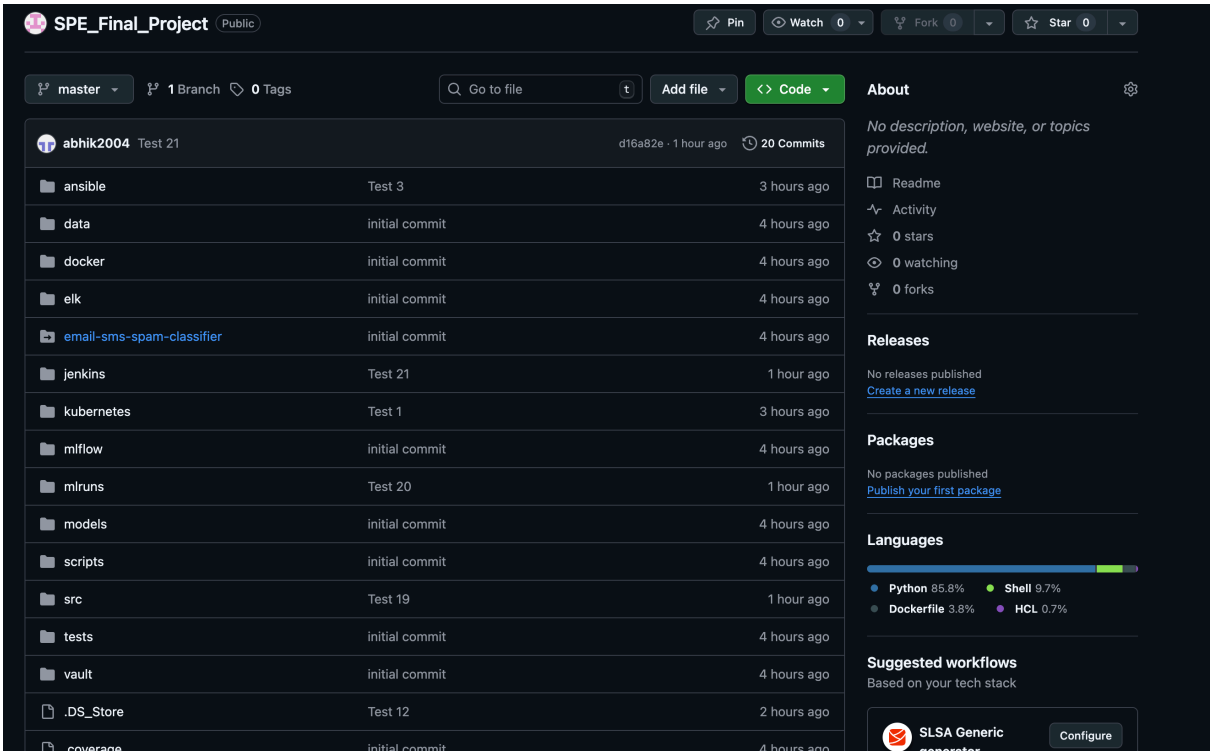


Figure 5.2: All project files committed and pushed to GitHub repository

Repository Structure:

```
Final_Project/
├── src/              # Application source code
├── tests/            # Test suite (pytest)
├── docker/           # Dockerfile configurations
├── kubernetes/       # K8s manifests
├── ansible/          # Ansible playbooks and roles
├── jenkins/          # Jenkinsfiles (CI/CD pipelines)
├── models/           # Trained ML models
├── data/             # Training dataset
└── vault/            # Vault policies
```

Step 5.3: Setup ngrok for Webhook Delivery

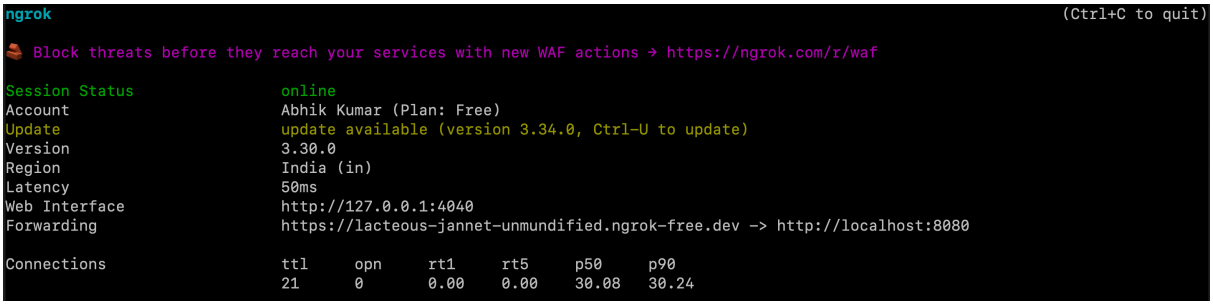


Figure 5.3: ngrok tunnel created to expose local Jenkins to GitHub webhooks

Why ngrok?

- Jenkins runs locally (http://localhost:8080)
- GitHub webhooks need a public URL
- ngrok creates a secure tunnel: `https://xxxx.ngrok.io → localhost:8080`

Step 5.4: Configure GitHub Webhook

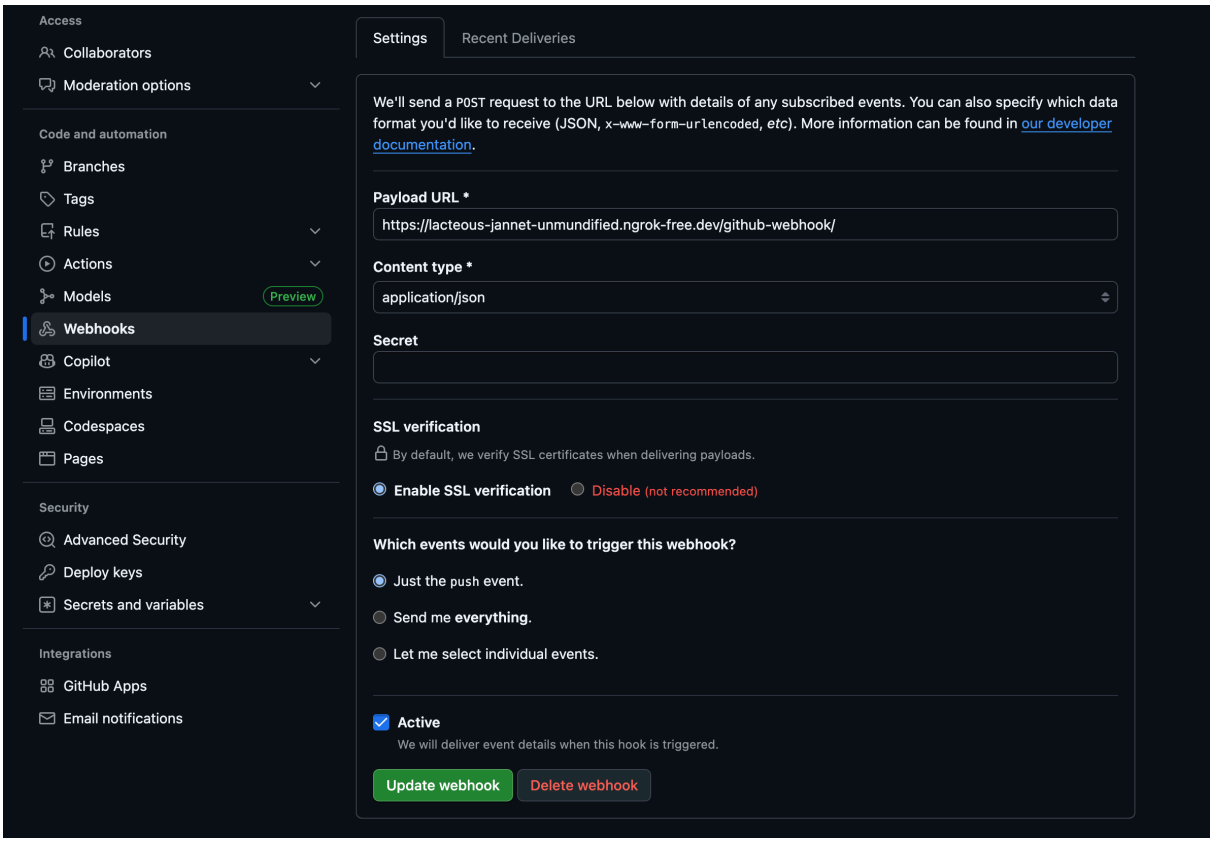


Figure 5.4: Webhook configured to trigger Jenkins on every git push

Phase 6: Jenkins - CI/CD Automation

Objective: Automate the complete software delivery pipeline from code commit to production deployment.

Step 6.1: Create Jenkins Pipeline

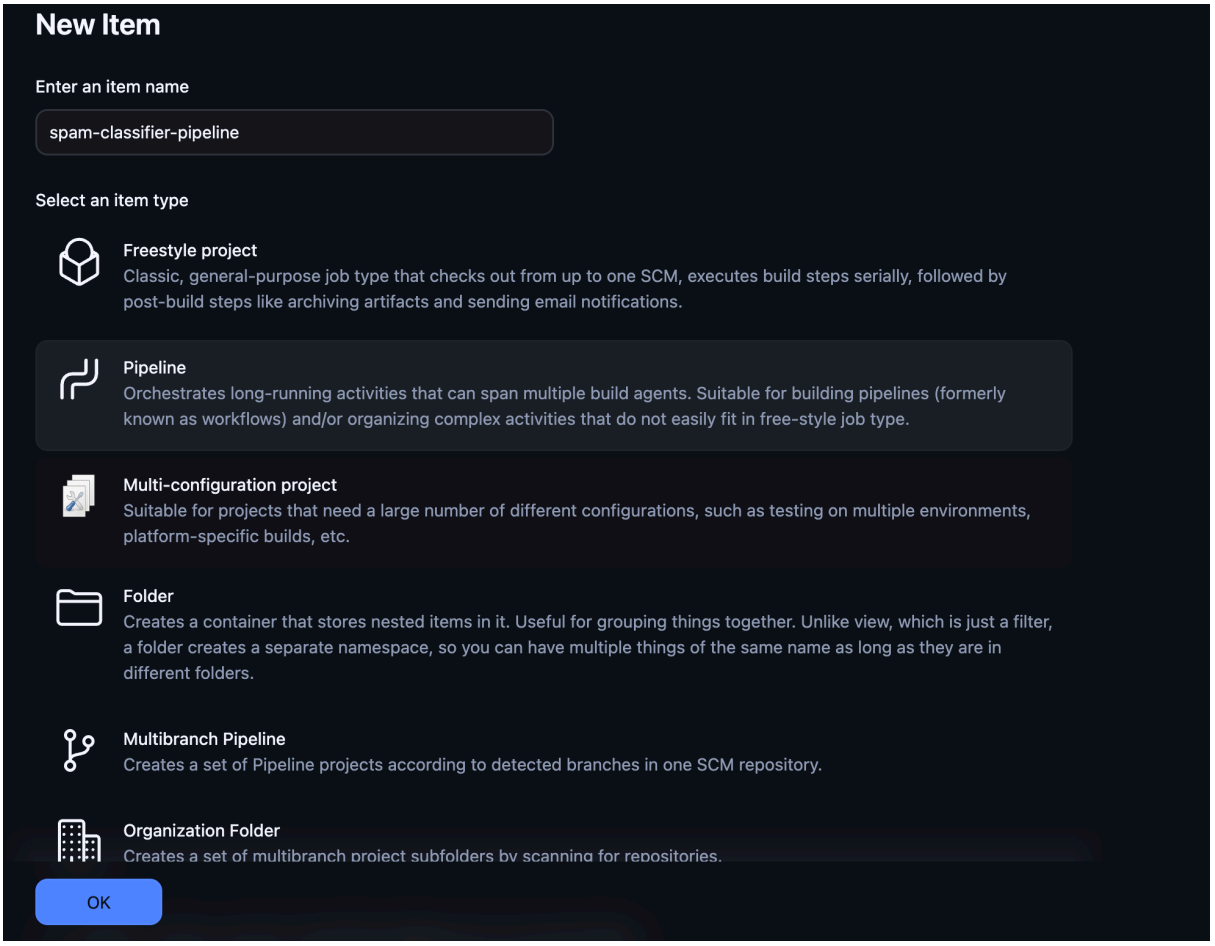


Figure 6.1: Creating the main CI/CD pipeline in Jenkins

Pipeline Configuration:
- **Name**: `spam-classifier-pipeline`
- **Type**: Pipeline
- **Source**: Git SCM
- **Jenkinsfile**: `jenkins/Jenkinsfile`
- **Trigger**: GitHub hook (automatic on push)

Step 6.2: Configure Pipeline Settings

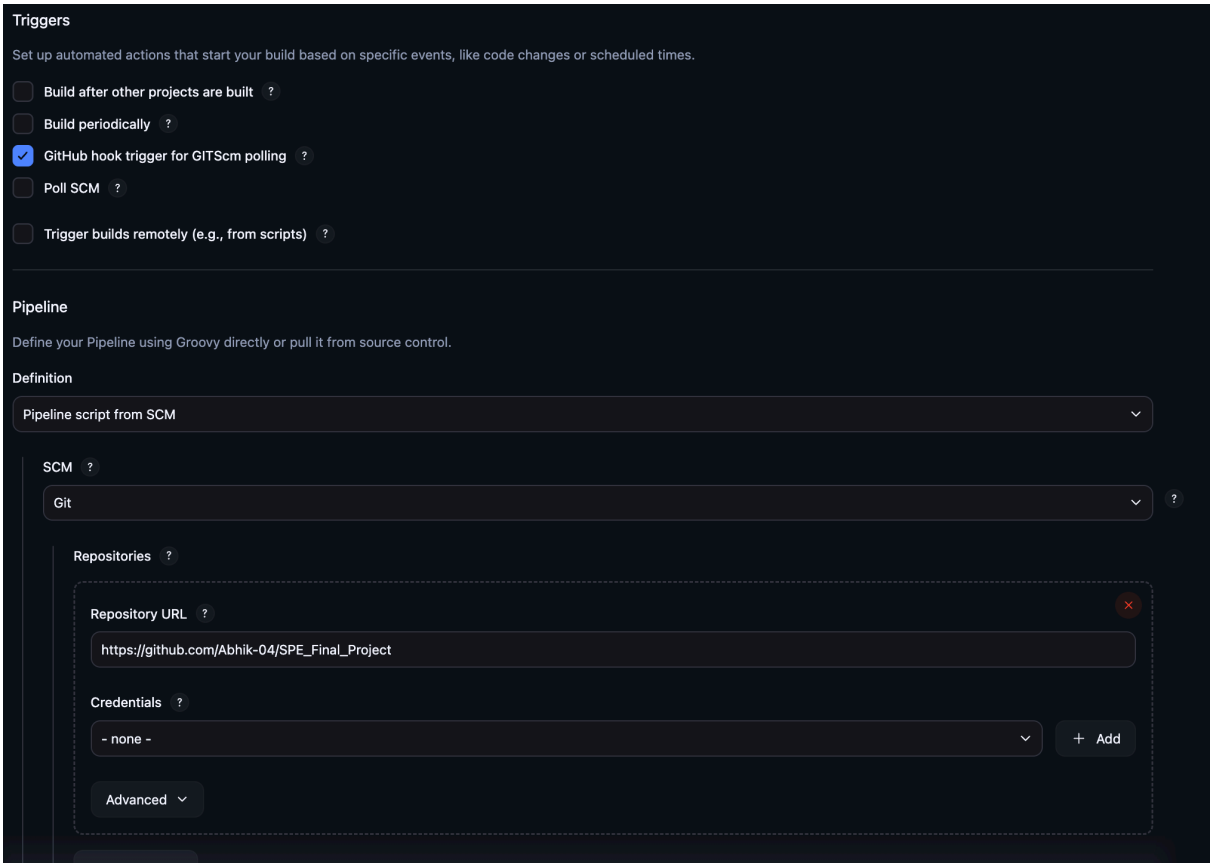


Figure 6.2: Pipeline configured with GitHub webhook integration and credentials

Key Configurations:
- GitHub hook trigger enabled
- Docker Hub credentials stored
- Kubernetes CLI configured
- Ansible playbooks integrated

Step 6.3: Create Model Training Pipeline

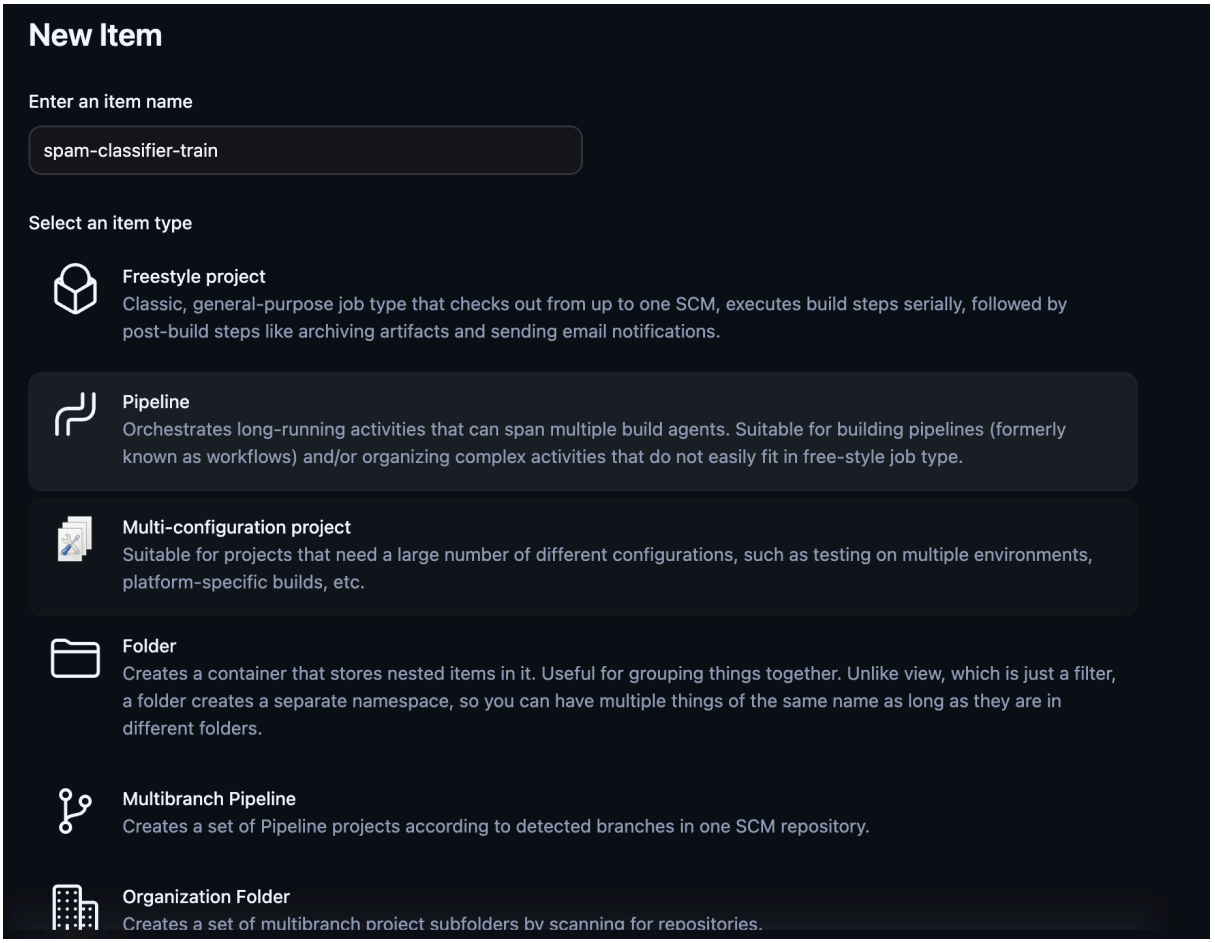


Figure 6.3: Separate pipeline created for automated model training

Training Pipeline Features:

- Triggered by main pipeline or manually
- Runs training script with MLflow tracking
- Evaluates model performance
- Promotes successful models

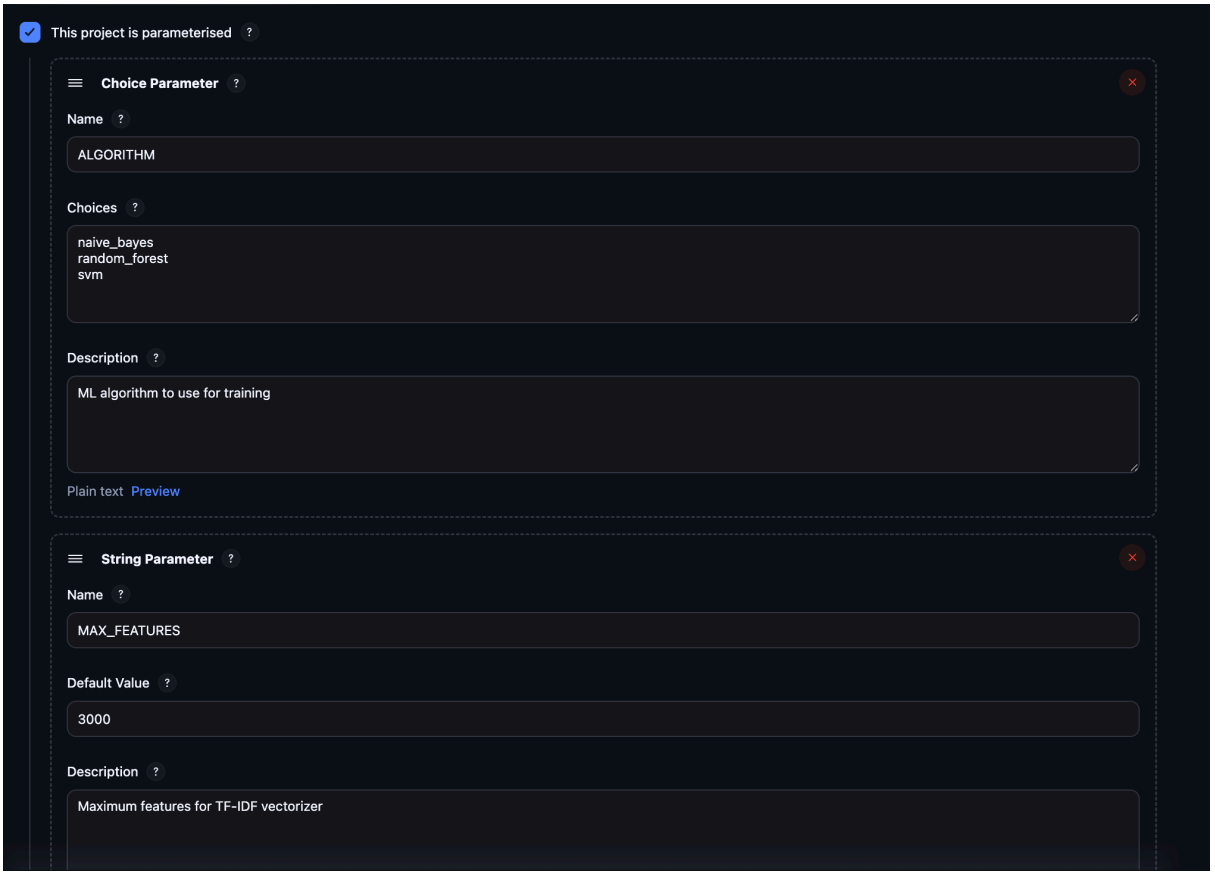


Figure 6.4: Training pipeline configuration and parameters

Step 6.4: Jenkins Pipeline Stages (12 Total)

The main CI/CD pipeline includes these stages:

1. **Checkout** - Clone repository from GitHub
2. **Setup Environment** - Create Python venv, install dependencies
3. **Run Tests** - Execute pytest with coverage (03e80%)
4. **Build Docker Image** - Multi-stage build with version tags
5. **Push to Docker Hub** - Upload images to registry
6. **Update Kubernetes Deployment** - Inject Git commit, build metadata
7. **Deploy with Ansible** - Execute deployment playbooks
8. **Deploy with kubectl** - Fallback deployment method
9. **Verify Deployment** - Check rollout status and pod health
10. **Health Check** - Validate application endpoints
11. **Trigger Model Training** - Launch training pipeline

12. **Cleanup** - Remove temporary files and images

Phase 7: Application Deployment

Objective: Deploy the application to Kubernetes and verify it's running correctly with auto-scaling enabled.

Step 7.1: Deploy Application to Kubernetes

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/deployment.yaml
deployment.apps/spam-classifier unchanged
serviceaccount/spam-classifier-sa unchanged
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/service.yaml

service/spam-classifier-service unchanged
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl apply -f kubernetes/hpa.yaml

horizontalpodautoscaler.autoscaling/spam-classifier-hpa unchanged
```

Figure 7.1: Application deployed to Kubernetes cluster

Kubernetes Resources Deployed:
- **Deployment**: 2 initial replicas with rolling update strategy
- **Service**: LoadBalancer type exposing port 80
- **HPA**: Auto-scaling from 2 to 10 pods
- **ServiceAccount**: For Vault secret injection

Step 7.2: Verify Deployment Status

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get pods -n spam-classifier
NAME                              READY   STATUS    RESTARTS   AGE
elasticsearch-0                   1/1     Running   0          169m
filebeat-w2dfr                    1/1     Running   0          164m
kibana-549f8b8b45-t7hj7           1/1     Running   0          166m
logstash-c796ddb6c-dxswf          1/1     Running   0          167m
mlflow-86c5759c59-2rnl7           1/1     Running   0          47m
spam-classifier-5c4dfdd45c-znpr5  1/1     Running   0          5m28s
spam-classifier-5c4dfdd45c-znv4c  1/1     Running   0          5m28s
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get svc -n spam-classifier
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
elasticsearch           ClusterIP      10.97.133.227    <none>        9200/TCP,9300/TCP   170m
kibana                  LoadBalancer   10.102.255.172   <pending>     5601:32050/TCP      167m
logstash                ClusterIP      10.99.20.13      <none>        5044/TCP            168m
mlflow-service          ClusterIP      10.109.216.136   <none>        5000/TCP            3h10m
spam-classifier-service LoadBalancer   10.102.213.6     <pending>     80:30197/TCP        48m
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % kubectl get hpa -n spam-classifier
NAME                 REFERENCE                     TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
spam-classifier-hpa  Deployment/spam-classifier  cpu: 5%/70%, memory: 22%/80%   2         10        2          30m
```

Figure 7.2: All pods running, health checks passing, HPA configured

Phase 8: Complete End-to-End Demo

Objective: Demonstrate the entire CI/CD flow from code change to production deployment with zero downtime.

Step 8.1: Git Push Triggers Pipeline

```
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % git add .
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % git commit -m "Test 21"
[master d16a82e] Test 21
 1 file changed, 20 insertions(+), 6 deletions(-)
(venv) (pydev) abhik04@Abhiks-MacBook-Air Final_Project % git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 10 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 689 bytes | 689.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/Abhik-04/SPE_Final_Project
   5de31d7..d16a82e  master -> master
```

Figure 8.1: Code changes pushed to GitHub, triggering the automated pipeline

Step 8.2: Jenkins Pipeline Automatically Triggered

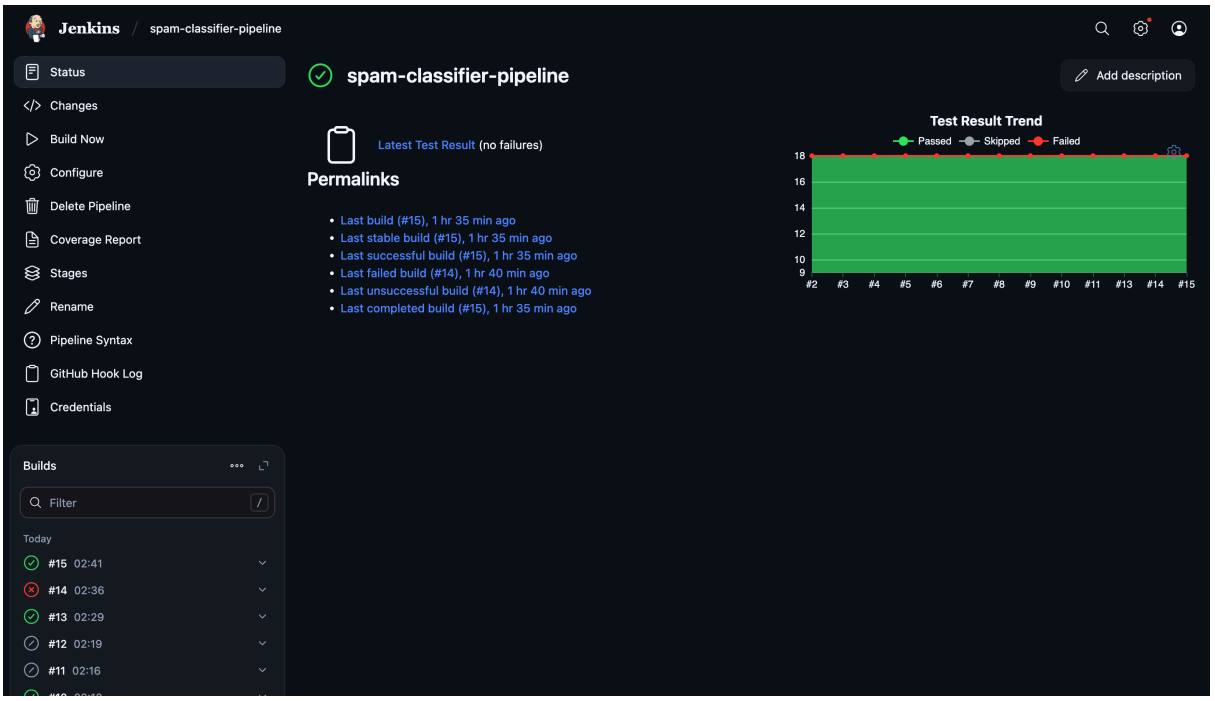


Figure 8.2: GitHub webhook triggers Jenkins pipeline execution

Step 8.3: Stage 1 - Checkout SCM

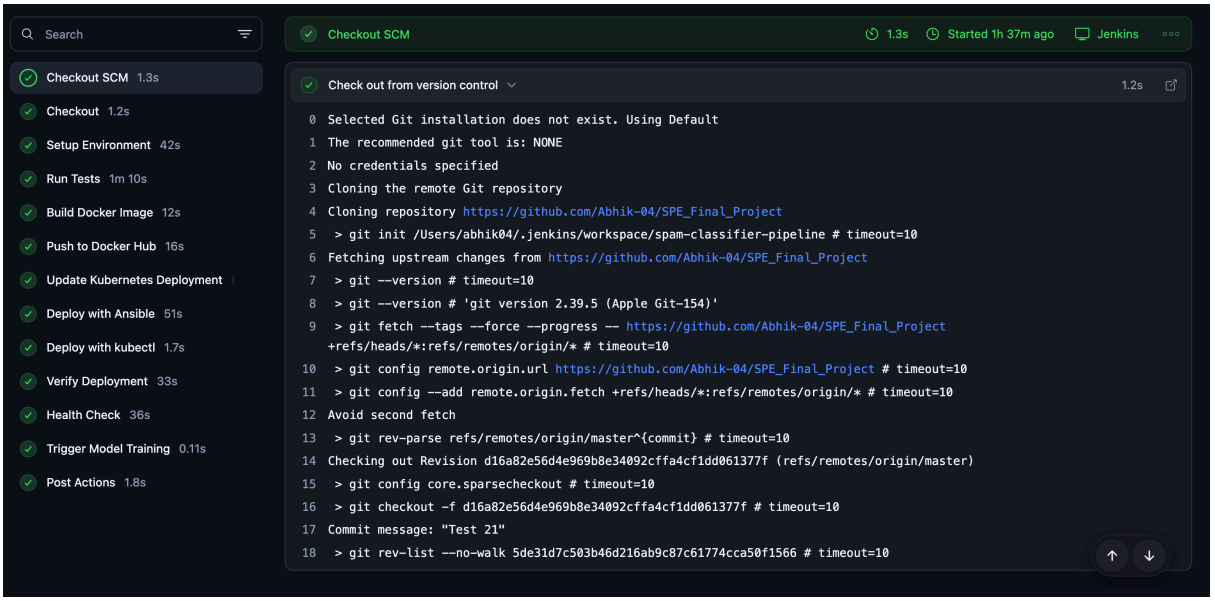


Figure 8.3: Jenkins clones the repository and displays commit details

Step 8.4: Stage 2 - Setup Environment

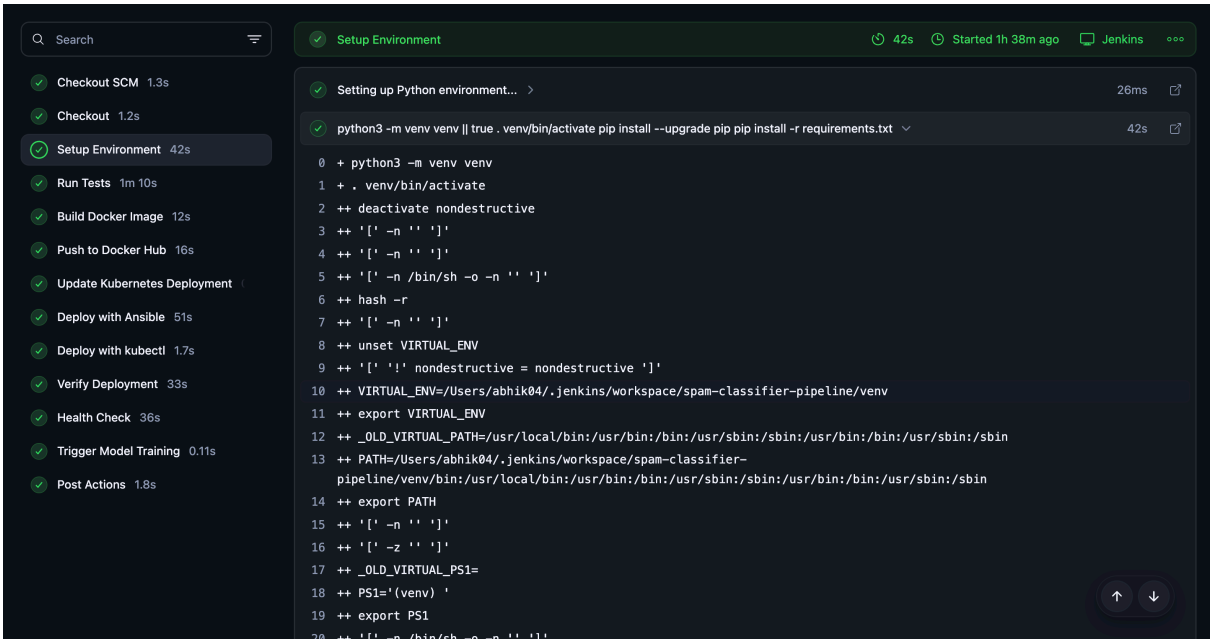


Figure 8.4: Python virtual environment created and dependencies installed

Step 8.5: Stage 3 - Run Tests

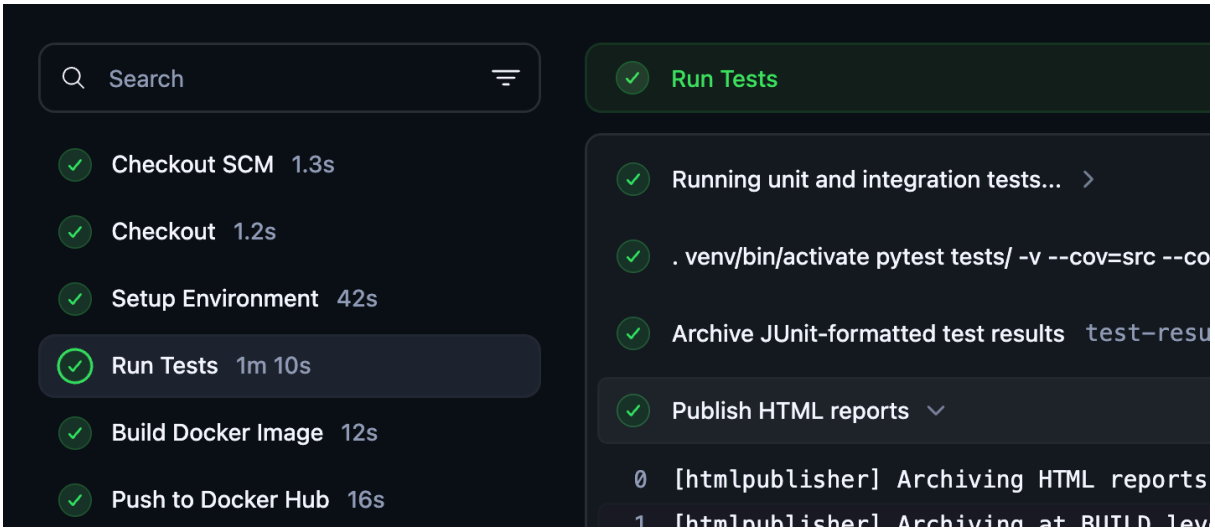


Figure 8.5: Automated test suite executed with coverage reporting

Step 8.6: Stage 4 - Build Docker Image

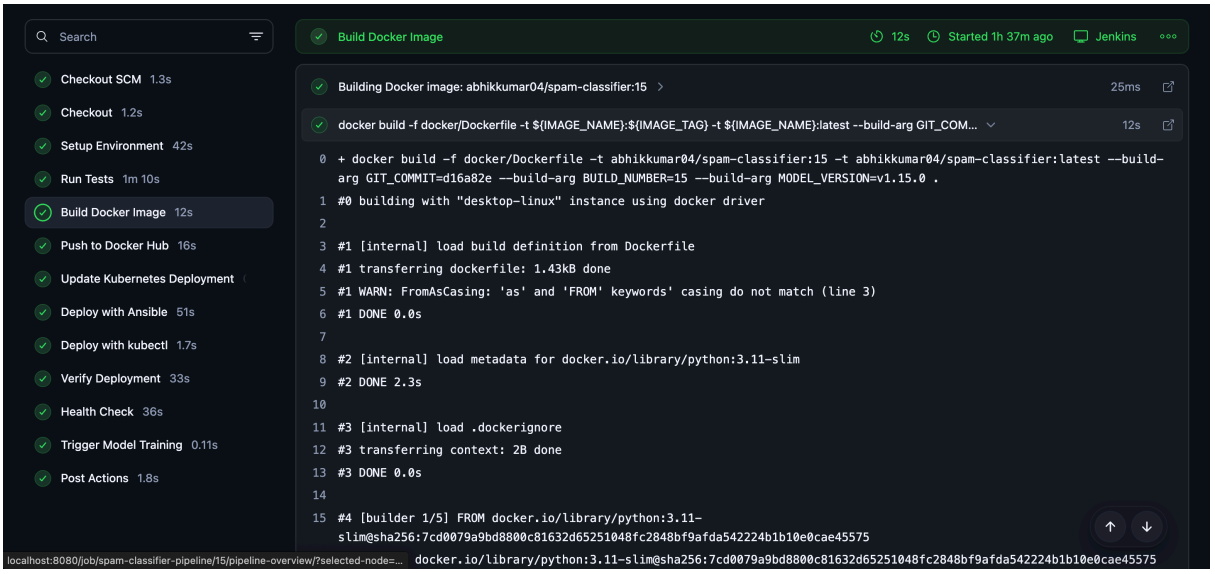


Figure 8.6: Multi-stage Docker image built with version tagging

Image Tags:

- `spam-classifier:latest`
- `spam-classifier:BUILD_NUMBER`
- Includes Git commit SHA in metadata

Step 8.7: Stage 5 - Push to Docker Hub

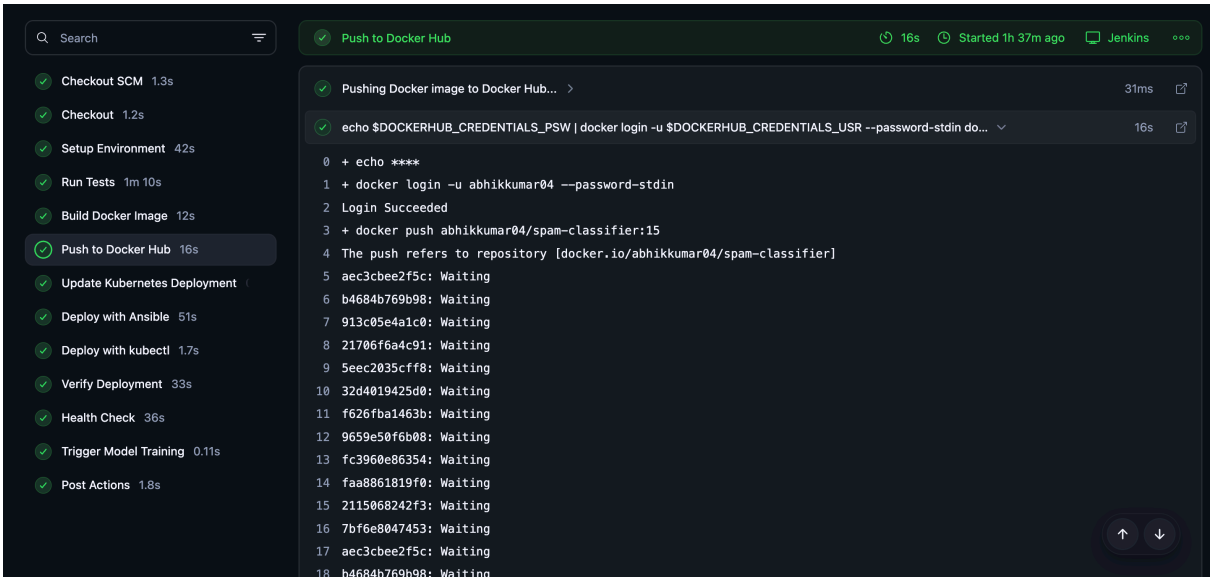


Figure 8.7: Built images pushed to Docker Hub registry

Step 8.8: Stage 6 - Update Kubernetes Deployment

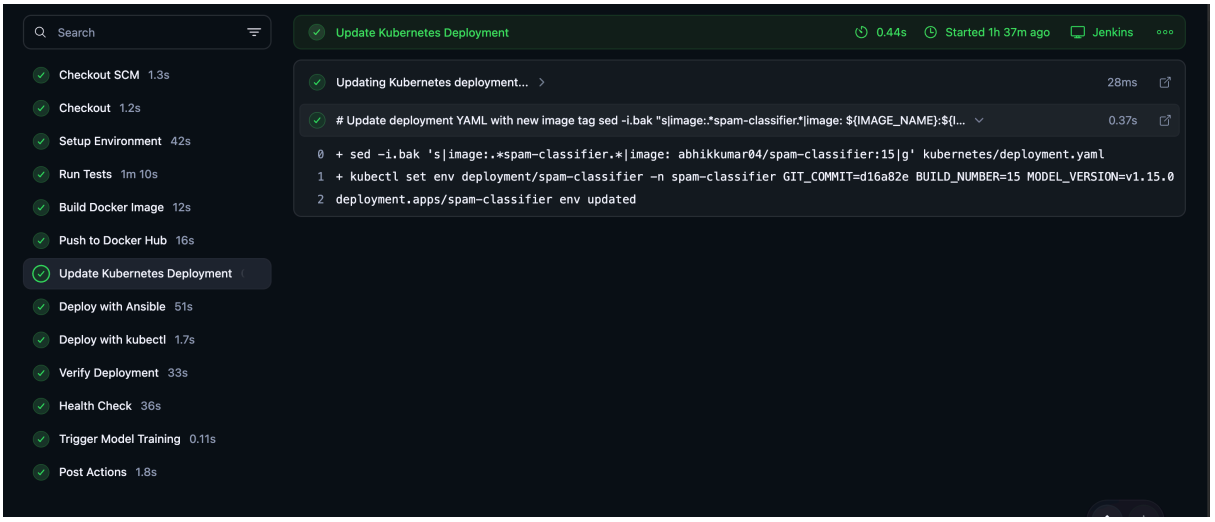


Figure 8.8: Kubernetes deployment updated with new image version

Updates Applied:
- New image tag set in deployment
- Environment variables updated (GIT_COMMIT, BUILD_NUMBER)
- Deployment YAML updated

Step 8.9: Stage 7 - Deploy with Ansible

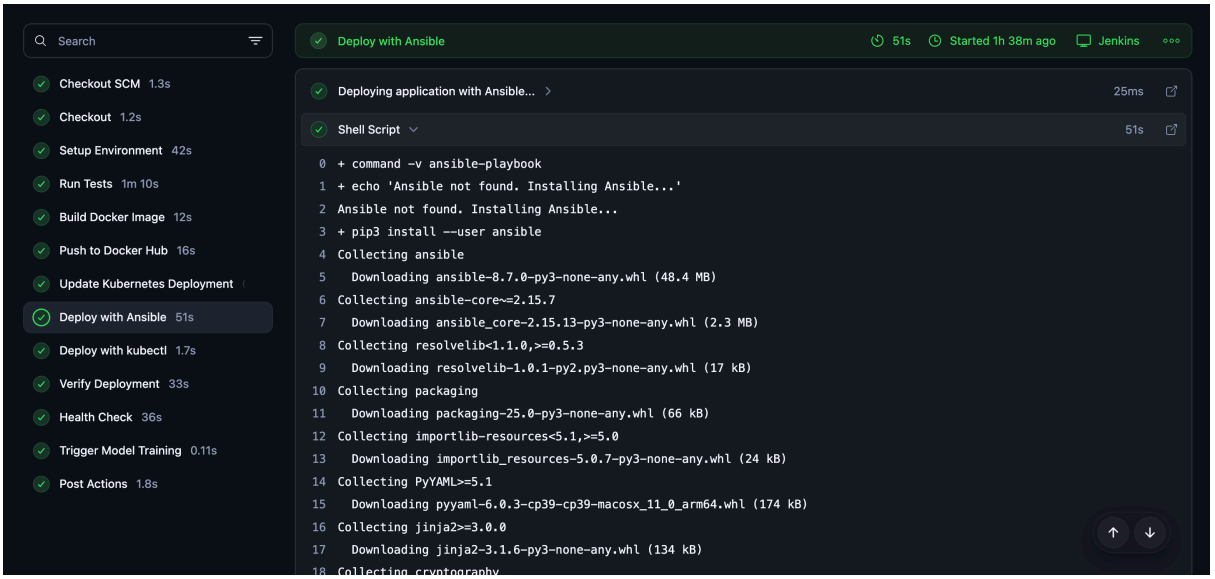


Figure 8.9: Ansible playbook executes deployment tasks

Step 8.10: Stage 8 - Deploy with kubectl (Fallback)

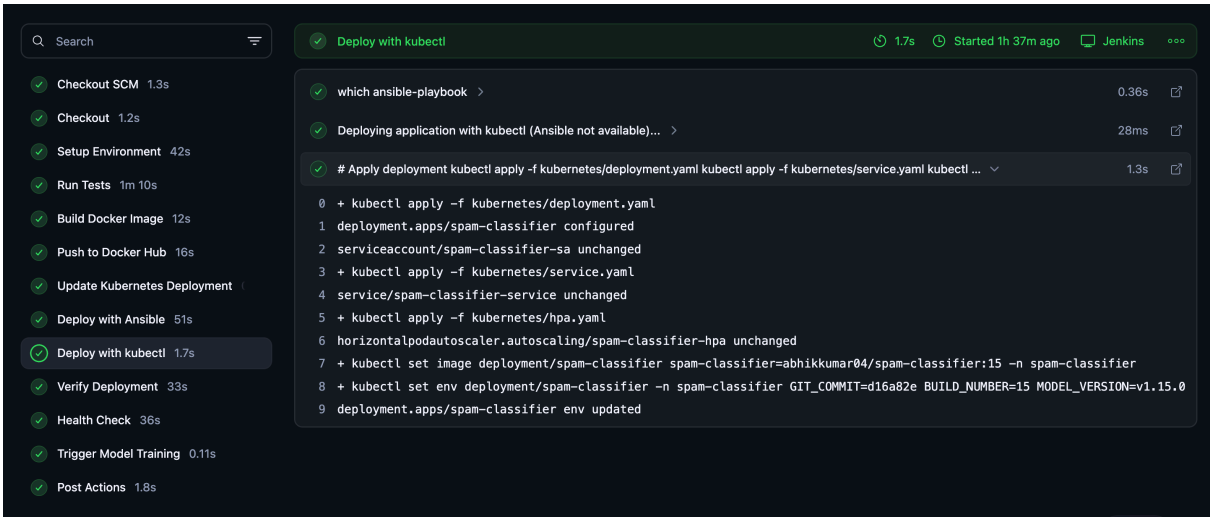


Figure 8.10: Kubernetes deployment via kubectl as fallback method

Step 8.11: Stage 9 - Verify Deployment

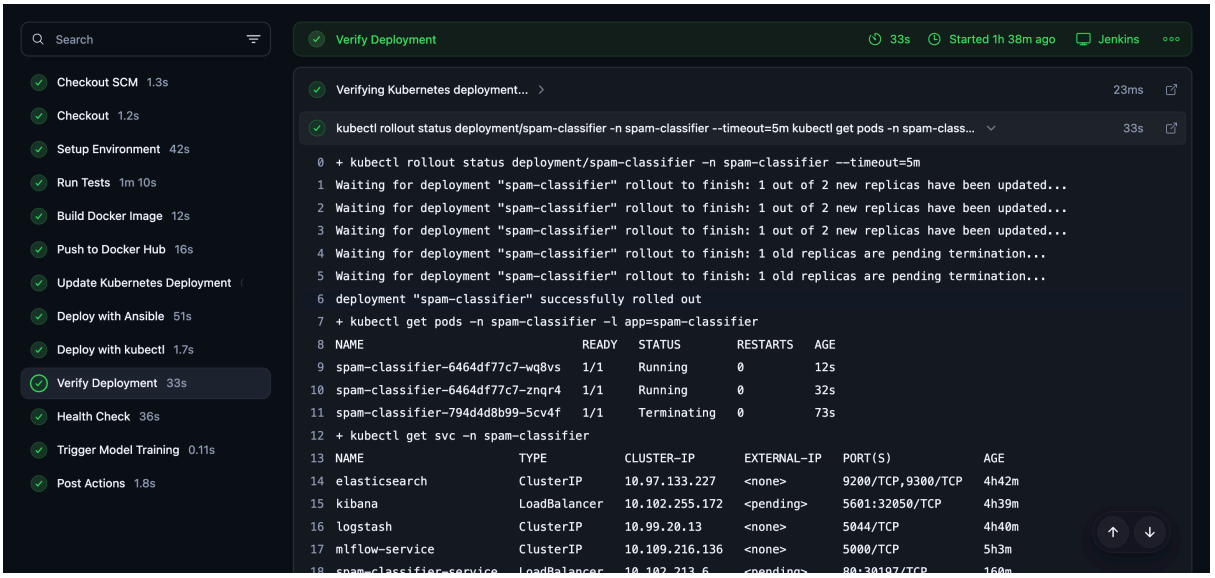


Figure 8.11: Rollout status checked, pods verified as healthy

Verification Shows:
- Rolling update completed successfully
- New pods running with updated version
- Old pods gracefully terminated
- Zero downtime maintained

Step 8.12: Stage 10 - Health Check

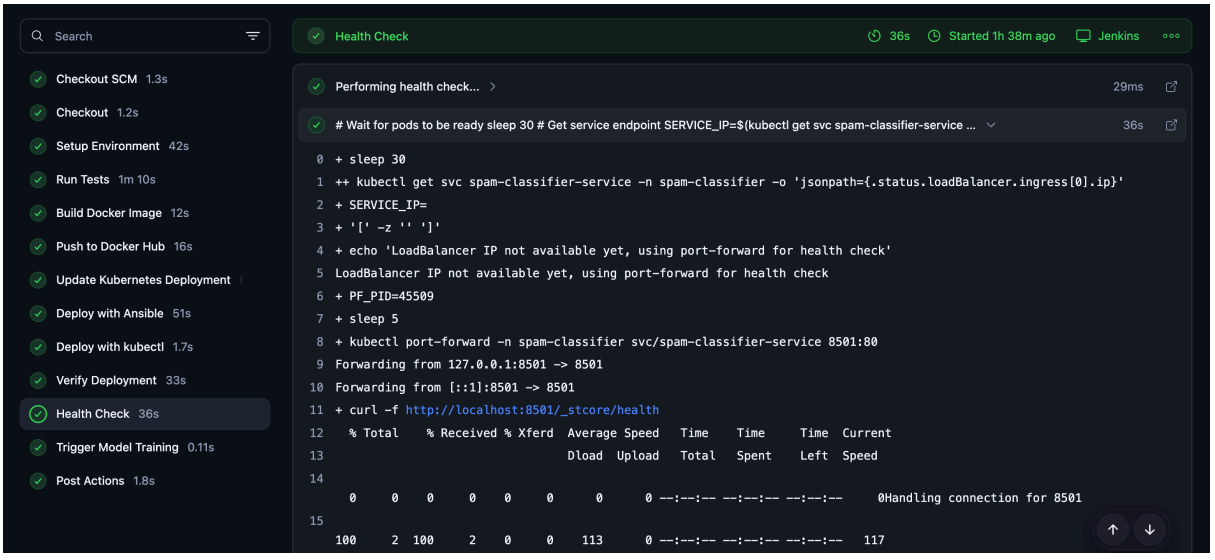


Figure 8.12: Application health endpoint validated post-deployment

Step 8.13: Trigger Model Training Pipeline

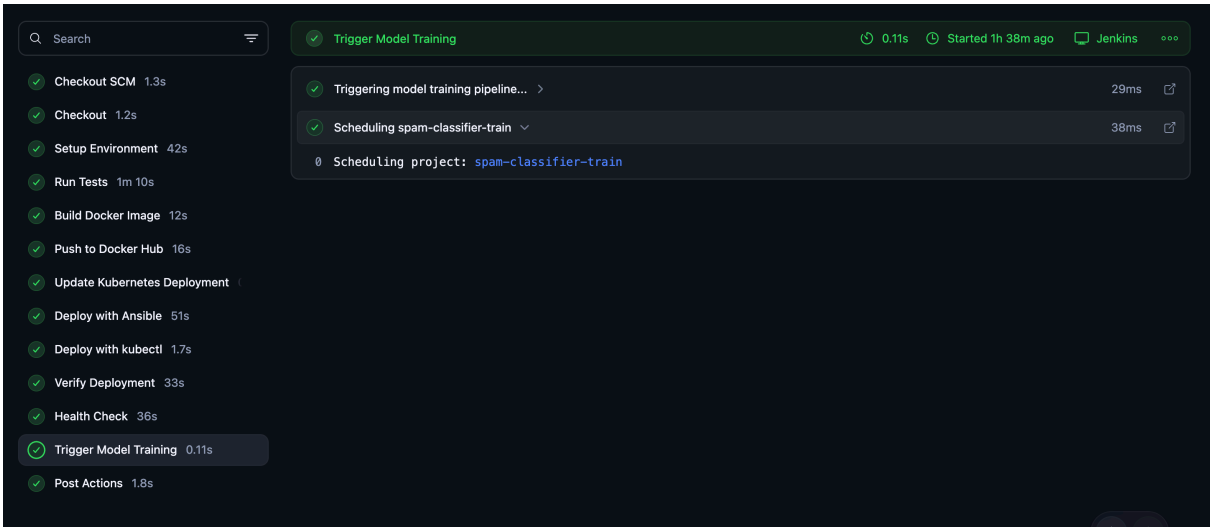


Figure 8.13: Separate model training pipeline triggered automatically

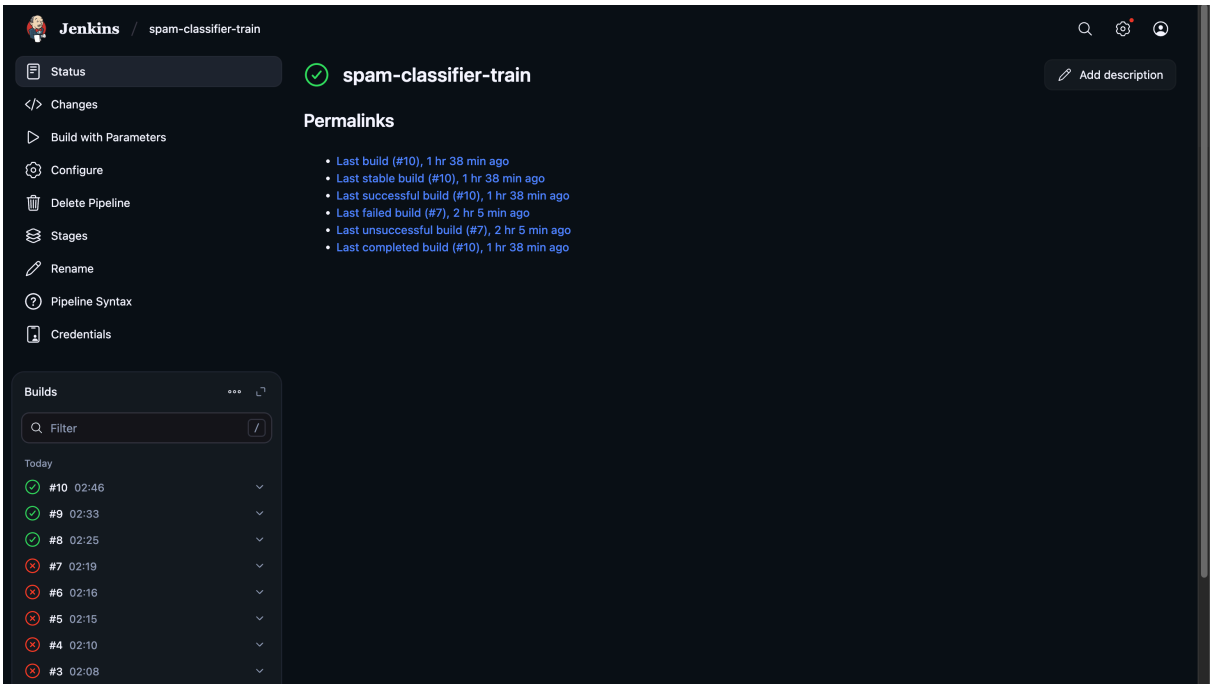


Figure 8.14: Training pipeline begins execution

Step 8.14: Training Pipeline - Checkout

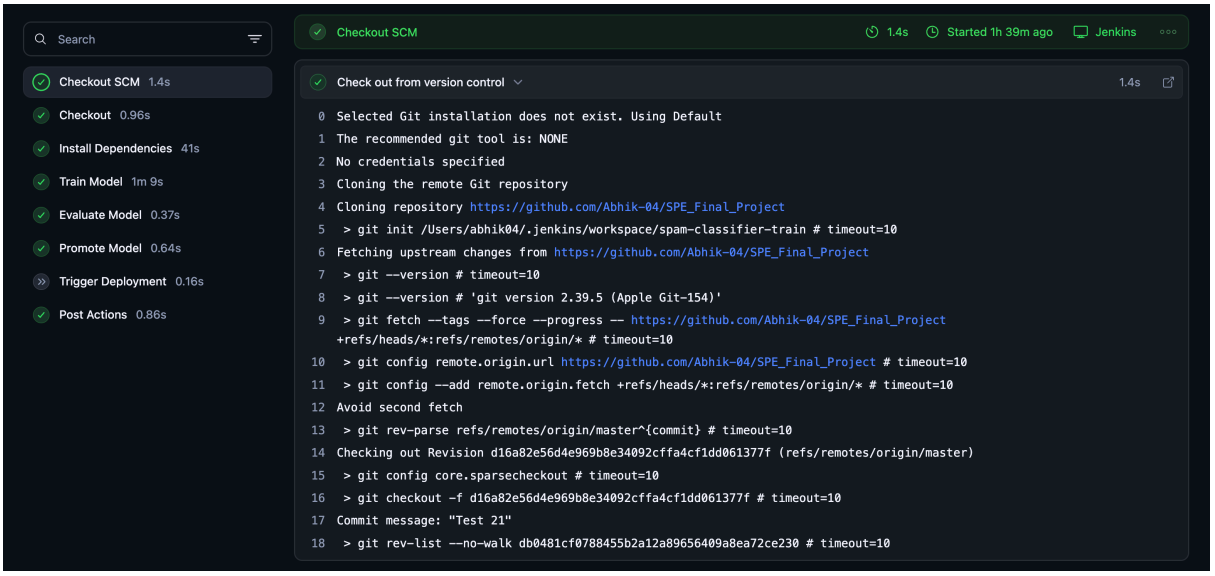


Figure 8.15: Training pipeline clones repository for model retraining

Step 8.15: Training Pipeline - Install Dependencies

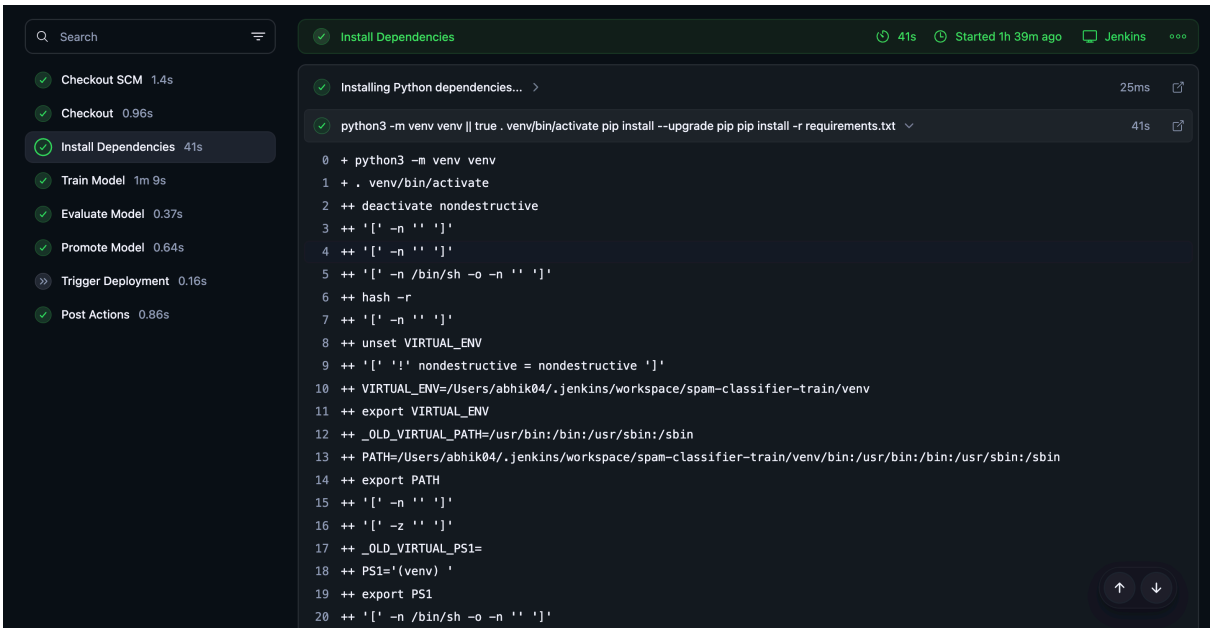


Figure 8.16: Training environment setup with ML libraries

Step 8.17: Training Pipeline - Train Model

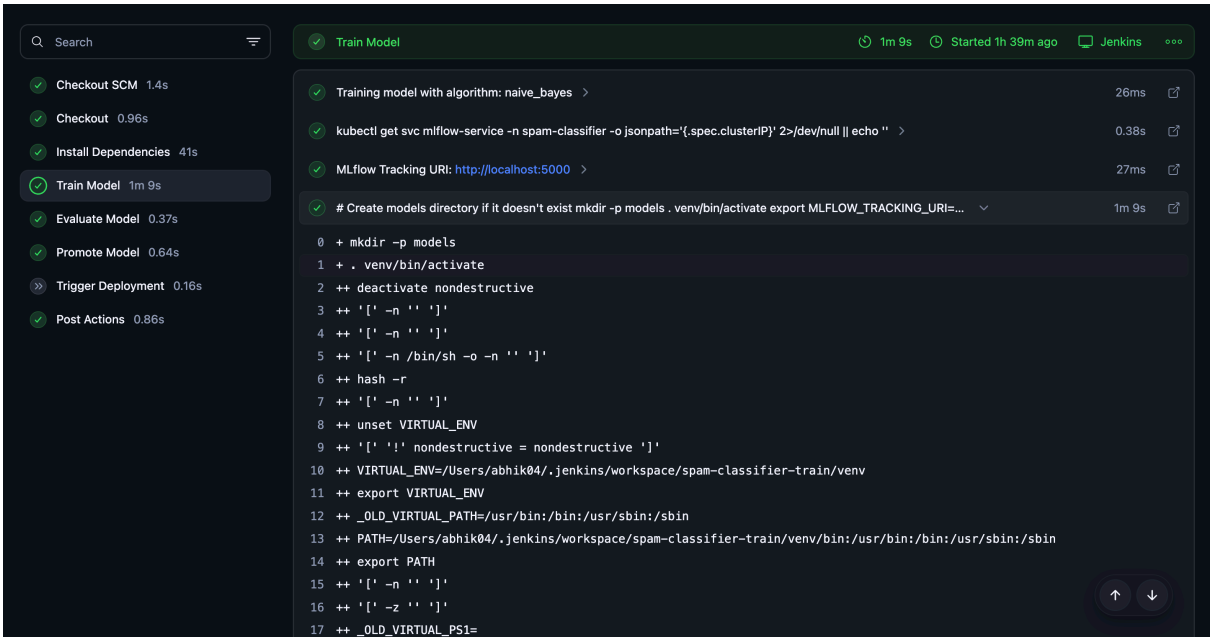


Figure 8.17: Model training executed with MLflow tracking

Step 8.18: Training Pipeline - Evaluate Model

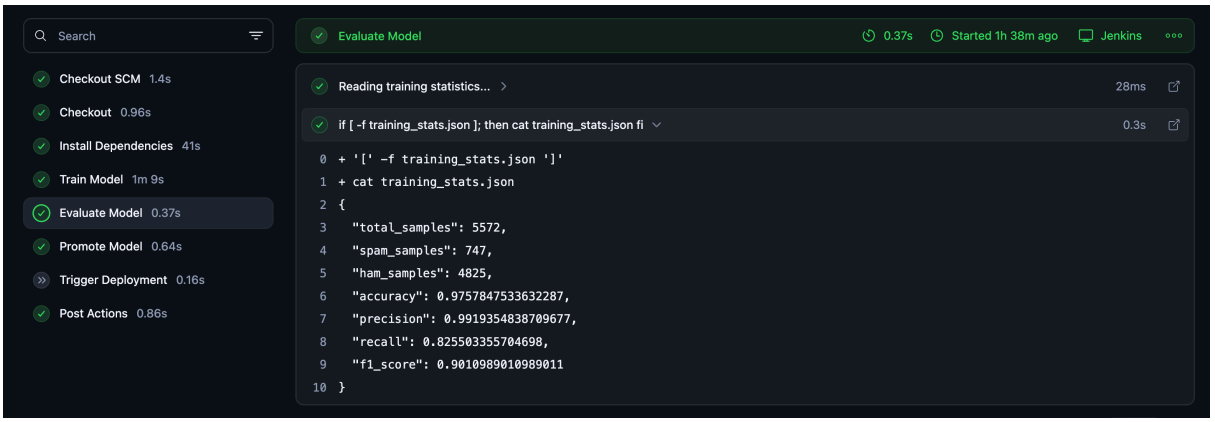


Figure 8.18: Model evaluation metrics calculated and logged

Metrics:
- Accuracy: 97.58%
- Precision: 99.19%
- Recall: 82.55%
- F1-Score: 90.11%

Step 8.19: Training Pipeline - Promote Model

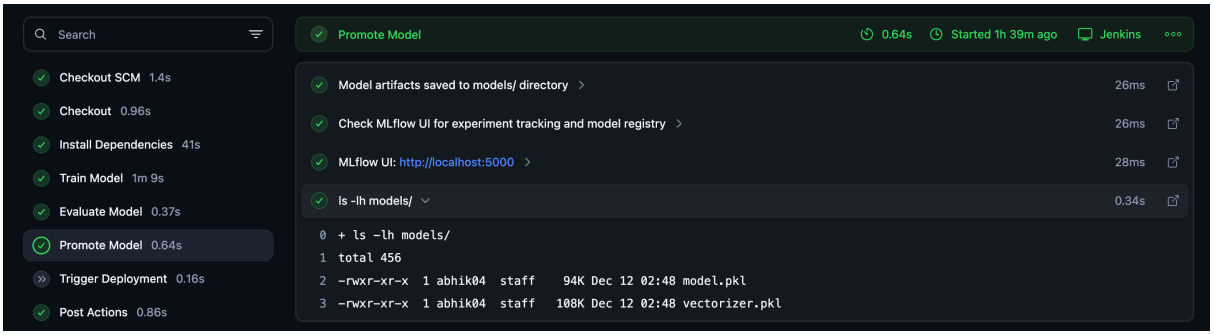


Figure 8.19: Successful model registered and promoted in MLflow

Step 8.20: Test Application - Spam Example

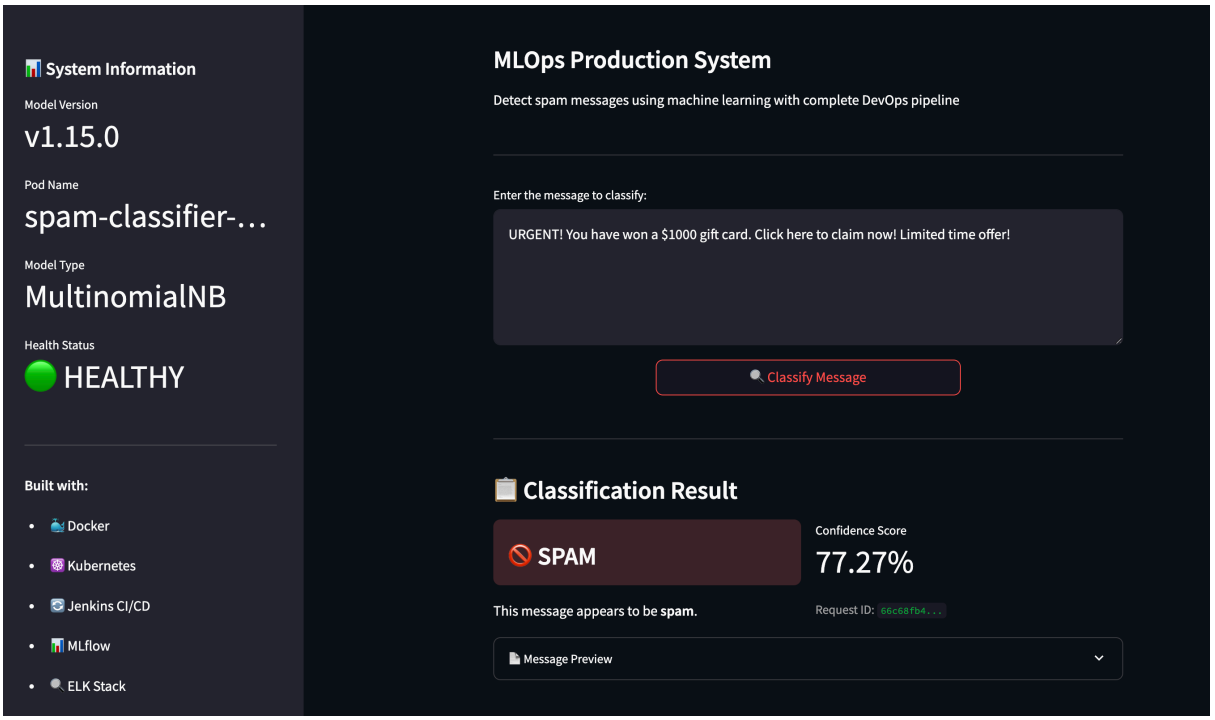


Figure 8.20: Application correctly identifies spam message with high confidence

Result:
- **Prediction**: SPAM
- **Confidence**: 77.27%
- **Request ID**: Logged for tracing

Step 8.21: Test Application - Not Spam Example

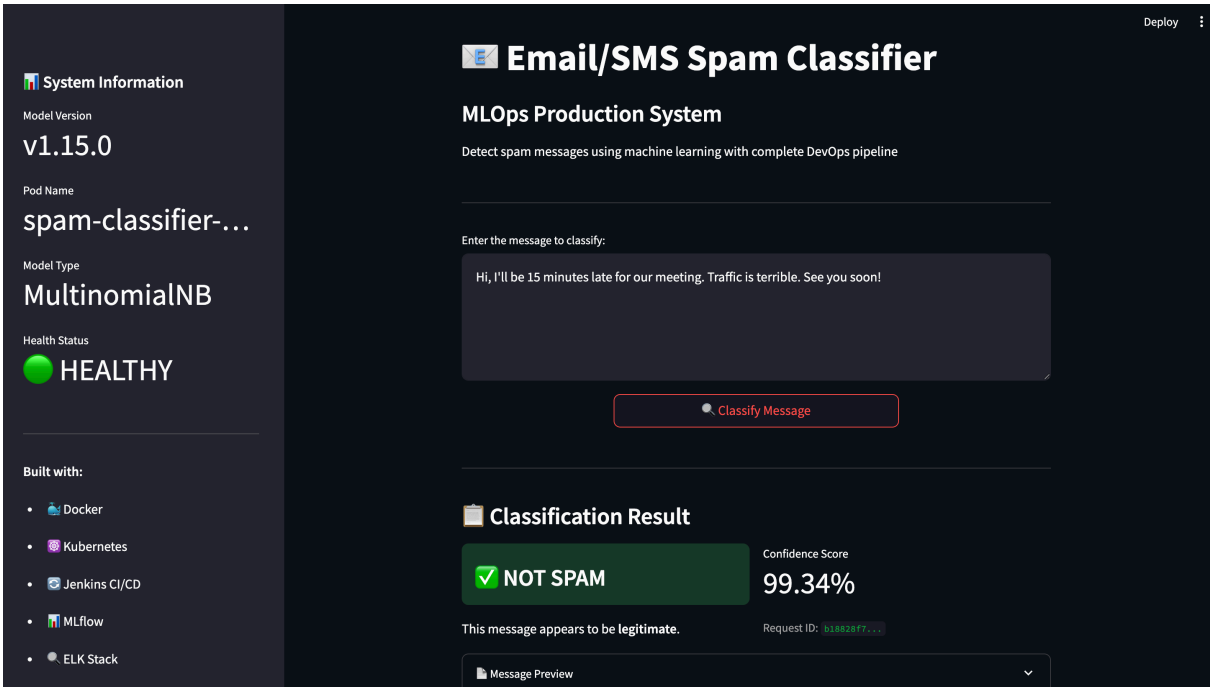


Figure 8.21: Application correctly identifies legitimate message

Result:
- **Prediction**: NOT SPAM
- **Confidence**: 99.34%

Step 8.22: View Logs in ELK Stack

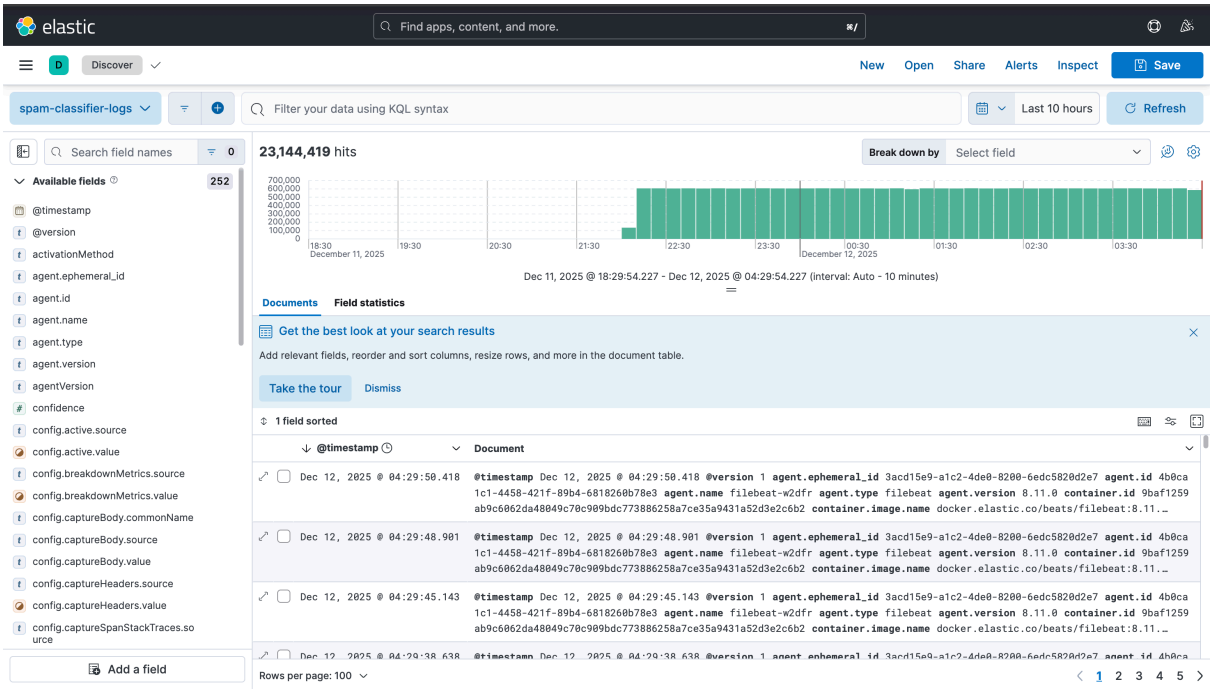


Figure 8.22: Real-time application logs showing prediction events

Logs Include:
- Prediction requests with text input
- Confidence scores
- Request IDs for tracing
- Pod names and timestamps
- Model version information

Step 8.23: Kibana Dashboard Visualizations

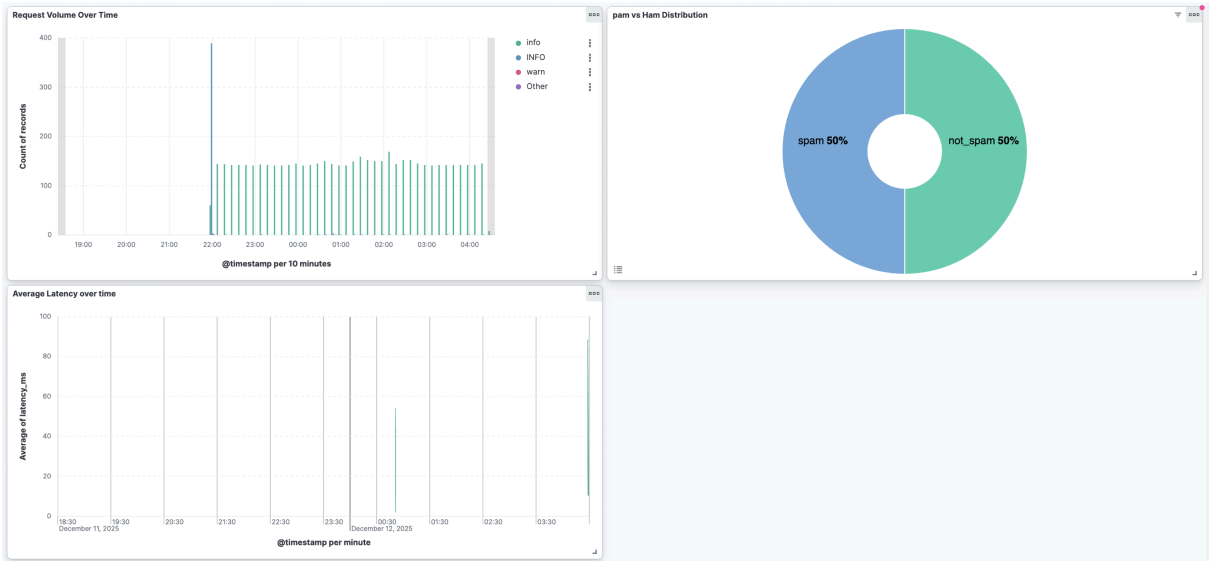


Figure 8.23: Custom Kibana dashboard showing prediction metrics and trends

Dashboard Widgets:
- Spam vs. Not-Spam distribution (pie chart)
- Prediction confidence histogram
- Request count over time (line graph)
- Pod-level log counts (bar chart)
- Error rate monitoring

Step 8.24: MLflow Experiment Tracking

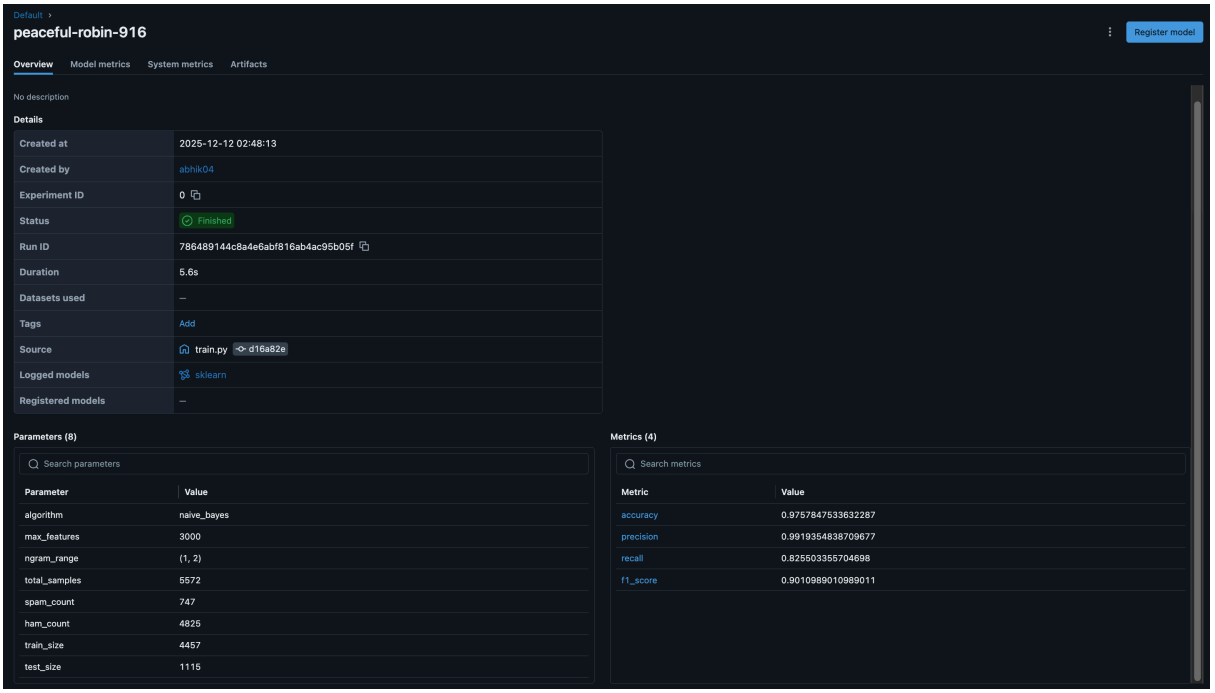


Figure 8.24: MLflow UI showing all experiment runs with performance metrics

MLflow Features Demonstrated:
- Multiple experiment runs compared
- Hyperparameter tracking (algorithm, max_features)
- Metric comparison (accuracy, precision, recall, F1)
- Artifact storage (models, confusion matrices)
- Run IDs linked to Git commits

---

4. Security - Vault Integration

4.1 HashiCorp Vault Overview

Purpose: Securely manage and inject secrets into the application without hardcoding credentials in code or configuration files.

Secrets Managed:
- Docker Hub credentials (username, password)
- Database connection strings
- API keys and tokens
- MLflow backend credentials

4.2 Vault Policy Configuration

The project includes a **Vault policy** (`vault/policies/app-policy.hcl`) that defines what secrets the application can access:

```
# Policy for spam-classifier application
path "secret/data/spam-classifier/*" {
  capabilities = ["read", "list"]
}

path "secret/data/dockerhub/*" {
  capabilities = ["read", "list"]
}

path "database/creds/mlflow" {
  capabilities = ["read"]
}
```

Explanation:
- `secret/data/spam-classifier/*`: Application-specific secrets (read-only access)
- `secret/data/dockerhub/*`: Docker Hub credentials for image pulls
- `database/creds/mlflow`: Dynamic database credentials for MLflow backend

4.3 Kubernetes Integration

Vault is integrated into Kubernetes using **Vault Agent Injector** annotations in the deployment manifest:

How It Works:
1. Vault Agent sidecar container is injected into the pod
2. Agent authenticates to Vault using Kubernetes ServiceAccount
3. Secrets are retrieved based on the `spam-classifier` role
4. Secrets are written to `/vault/secrets/` for the application to read
5. Secrets are automatically rotated when they expire

4.4 Ansible Vault Role

The project includes an **Ansible role** (`ansible/roles/vault-config`) to set up Vault:

Tasks Performed:
- Create Vault namespace in Kubernetes
- Display Vault installation instructions

- Configure Kubernetes authentication backend
- Create policies and roles

4.5 Security Benefits

No Hardcoded Secrets: Credentials never stored in Git or configuration files

Least Privilege Access: Each application only accesses its required secrets

Audit Logging: All secret access is logged for compliance

Automatic Rotation: Secrets can be rotated without redeploying applications

Encrypted Storage: All secrets encrypted at rest in Vault

Dynamic Credentials: Short-lived database credentials generated on demand

---

5. Key Features & Achievements

5.1 Complete CI/CD Pipeline

12-Stage Automated Pipeline: From code commit to production deployment

GitHub Webhook Integration: Automatic trigger on every push

Automated Testing: pytest

Docker Image Management: Multi-stage builds, versioned tags

Zero-Downtime Deployments: Rolling updates with health checks

Deployment Verification: Automated health checks and rollout validation

5.2 Kubernetes Orchestration

Auto-Scaling: HPA scales from 2 to 10 pods based on CPU/memory

Health Monitoring: Liveness and readiness probes

Resource Management: CPU/memory requests and limits

ConfigMaps & Secrets: Externalized configuration

Service Discovery: LoadBalancer service for external access

Graceful Shutdown: PreStop hooks for clean termination

5.3 MLOps Best Practices

Experiment Tracking: All training runs logged in MLflow

Model Versioning: Git commit SHA linked to model versions

Artifact Management: Models, vectorizers, metrics stored

Automated Training Pipeline: Separate Jenkins job for retraining

Model Registry: Centralized model storage and promotion

Prediction Logging: All inferences logged with confidence scores

5.4 Observability & Monitoring

Centralized Logging: ELK Stack with Elasticsearch, Logstash, Kibana, Filebeat

Log Visualization: Kibana dashboards for real-time analysis

Structured Logging: JSON logs with prediction metadata

Search Capabilities: Full-text search across all logs

Alerting: (Can be configured for anomaly detection)

5.5 Security & Best Practices

Vault Integration: Secure secrets management

Non-Root Containers: All containers run as non-root users

RBAC: Kubernetes role-based access control

Network Policies: (Can be configured for pod-to-pod security)

Image Scanning: Multi-stage builds reduce attack surface

Audit Trail: All deployments tracked with Git commits

Conclusion

This project successfully demonstrates a **production-ready MLOps pipeline** for deploying machine learning models. The implementation follows industry best practices and showcases:

- **Complete Automation**: Every step from code commit to deployment is automated
- **Scalability**: Kubernetes HPA ensures the system handles varying loads
- **Reliability**: Health checks, rolling updates, and zero-downtime deployments
- **Observability**: Comprehensive logging and monitoring with ELK Stack
- **Security**: Vault integration for secrets management
- **ML Excellence**: Complete ML lifecycle with MLflow tracking

The step-by-step implementation flow documented here enables anyone with the project files to recreate this system and understand how modern MLOps pipelines work in production environments.
