Databricks MLOps

1. Introduction

This document presents a production-ready MLOps framework for Databricks on AWS that integrates best practices from the official Databricks MLOps Stacks, enhanced with enterprise-grade features. The framework follows a cookie-cutter pattern to enforce consistency, providing a flexible blueprint for organizations to standardize their machine learning workflows. By focusing on comprehensive governance, pluggable data integration, robust model management, and automated workflows, this framework delivers reliable machine learning solutions at scale.

2. Architecture Overview

2.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                Enterprise MLOps Framework on AWS            │
└──────────────────────────┬──────────────────────────────────┘
                           │
              ┌────────────▼────────────┐
              │ Databricks Asset Bundles │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │  Unity Catalog Governance │
              └──┬─────────────────────┬─┘
                 │                     │
      ┌──────────▼───────┐   ┌─────────▼─────────┐
      │ Development      │   │ Staging           │
      │ Catalog          │   │ Catalog           │
      └────────┬─────────┘   └─────────┬─────────┘
               │                       │
   ┌───────────▼──────────┐   ┌────────▼───────────┐
   │ Data Integration     │   │ Model Registry     │
   │ - Delta Tables       │   │ - Model Versioning │
   │ - Snowflake          │   │ - Model Cards      │
   │ - S3                 │   │ - Lineage Tracking │
   └───────────┬──────────┘   └────────┬───────────┘
```

```
         │              │
┌─────────────▼─────────────┐  ┌─────────────▼─────────────┐
│ Feature Store        │    │ Model Deployment  │
│ - Feature Registry   │    │ - Serving Endpoints│
│ - Feature Versioning │    │ - Batch Inference │
└────────────┬───────────────┘  └─────────────┬─────────────┘
         │              │
         │     ┌─────────────▼─────────────┐
         │     │ Production      │
         │     │ Catalog         │
         │     └─────────────┬─────────────┘
         │              │
┌─────────────▼──────────────────────────────▼─────────────┐
│          CI/CD Automation          │
└───────────────────────────────────────────────────────────┘
```

2.2 Key Components

1. **Unity Catalog Governance**: Centralized governance layer providing:
   - Unified security model across all environments
   - Fine-grained access control for data, models, and features
   - Lineage tracking for end-to-end reproducibility
   - Audit logging for compliance requirements
2. **Flexible Data Integration**: Support for multiple data sources:
   - Delta tables in Unity Catalog
   - External sources (Snowflake, Redshift, etc.)
   - Cloud storage (S3, ADLS, etc.)
   - Streaming data sources
3. **Comprehensive Feature Store**:
   - Centralized feature registry
   - Feature versioning and lineage
   - Online/offline feature serving
   - Feature sharing across teams
4. **Enhanced Model Registry**:
   - Model versioning with Git integration
   - Model cards with business and technical metadata
   - Model approval workflows
   - Compliance documentation
5. **Robust CI/CD Automation**:
   - Infrastructure as code via Databricks Asset Bundles
   - Pipeline testing and validation
   - Environment promotion workflows
   - Automated deployment

3. Directory Structure

This cookie-cutter template provides a consistent project structure:

```
{{ cookiecutter.project_name }}/
├── .github/                    # CI/CD workflows
│   └── workflows/
│       ├── ci.yml              # CI pipeline
│       └── cd.yml              # CD pipeline
├── conf/                       # Configuration files
│   ├── catalog/                # Unity Catalog configurations
│   ├── data_sources/           # Data source configurations
│   │   ├── delta.yml           # Delta table configs
│   │   ├── snowflake.yml       # Snowflake connection configs
│   │   └── s3.yml              # S3 configs
│   └── deployment/             # Environment-specific configs
│       ├── dev.yml             # Development environment
│       ├── staging.yml         # Staging environment
│       └── prod.yml            # Production environment
├── databricks.yml              # Root DAB configuration
├── pyproject.toml              # Project dependencies
├── src/
│   └── {{ cookiecutter.package_name }}/
│       ├── common/             # Shared utilities
│       │   ├── config.py       # Configuration management
│       │   └── utils.py        # Helper functions
│       ├── data/               # Data processing modules
│       │   ├── connectors/     # Data source connectors
│       │   │   ├── base.py     # Base connector interface
│       │   │   ├── delta.py    # Delta connector
│       │   │   ├── snowflake.py # Snowflake connector
│       │   │   └── s3.py       # S3 connector
│       │   ├── data_loader.py  # Generic data loading interface
│       │   └── transformations.py # Data transformation utilities
│       ├── features/           # Feature Store integration
│       │   ├── feature_store.py # Feature store operations
│       │   ├── feature_registry.py # Feature registry management
│       │   └── feature_specs.py # Feature definitions
│       ├── governance/         # Governance modules
│       │   ├── model_cards.py  # Model card management
│       │   ├── documentation.py # Documentation utilities
│       │   └── lineage.py      # Lineage tracking
│       ├── models/             # Model definitions
│       │   ├── model_factory.py # Generic model factory
```

```
│       │   ├── evaluation.py        # Model validation utilities
│       │   └── registry.py          # Model registry operations
│       ├── pipelines/               # End-to-end pipelines
│       │   ├── training_pipeline.py  # Model training pipeline
│       │   ├── inference_pipeline.py # Batch inference pipeline
│       │   └── monitoring_pipeline.py # Model monitoring
│       └── serving/                 # Model serving utilities
│           ├── endpoints.py         # Serving endpoint management
│           └── scoring.py           # Model scoring logic
├── resources/                       # Databricks Asset Definitions
│   ├── jobs/                        # Job definitions
│   │   ├── feature_engineering_job.yml  # Feature engineering job
│   │   ├── model_training_job.yml   # Training job
│   │   └── inference_job.yml        # Inference job
│   ├── experiments/                 # MLflow experiments
│   │   └── experiment.yml           # Experiment definition
│   ├── models/                      # Registered models
│   │   └── model.yml                # Model definition
│   └── endpoints/                   # Serving endpoints
│       └── endpoint.yml             # Endpoint definition
└── tests/                           # Test suite
    ├── unit/                        # Unit tests
    │   ├── test_data.py
    │   ├── test_features.py
    │   └── test_models.py
    └── integration/                 # Integration tests
        ├── test_pipelines.py
        └── test_end_to_end.py
```

4. Project Configuration

4.1 Project Dependencies (pyproject.toml)

The `pyproject.toml` file defines all project dependencies:

[build-system]
requires = ["setuptools>=42", "wheel"]
build-backend = "setuptools.build_meta"

[project]
name = "{{ cookiecutter.package_name }}"
version = "0.1.0"

```toml
description = "Enterprise MLOps Framework for Databricks on AWS"
requires-python = ">=3.9"
authors = [
    {name = "{{ cookiecutter.author_name }}", email = "{{ cookiecutter.author_email }}"}
]
dependencies = [
    # Core dependencies
    "pyspark>=3.4.0",
    "delta-spark>=2.4.0",
    "databricks-sdk>=0.12.0",
    "databricks-cli>=0.100.0",

    # MLOps dependencies
    "mlflow>=2.7.0",
    "scikit-learn>=1.3.0",
    "xgboost>=1.7.6",
    "lightgbm>=4.0.0",

    # Data processing
    "pandas>=2.0.0",
    "numpy>=1.24.0",
    "pydantic>=2.0.0",

    # Utilities
    "PyYAML>=6.0",
    "python-dotenv>=1.0.0",
    "boto3>=1.28.0",
    "great-expectations>=0.17.0",
]

[project.optional-dependencies]
dev = [
    "pytest>=7.0.0",
    "pytest-cov>=4.1.0",
    "black>=23.1.0",
    "isort>=5.12.0",
    "mypy>=1.0.0",
    "flake8>=6.0.0",
    "pre-commit>=3.3.0",
    "uv>=0.1.0",
]

snowflake = [
    "snowflake-connector-python>=3.0.0",
```

```
]

monitoring = [
    "evidently>=0.3.0",
    "prometheus-client>=0.16.0",
]

[tool.black]
line-length = 100
target-version = ["py39"]

[tool.isort]
profile = "black"
line_length = 100

[tool.mypy]
python_version = "3.9"
warn_return_any = true
warn_unused_configs = true
disallow_untyped_defs = true
disallow_incomplete_defs = true

[tool.pytest.ini_options]
testpaths = ["tests"]
```

4.2 Databricks Asset Bundle Configuration

The root `databricks.yml` file configures all Databricks resources:

```
# databricks.yml
bundle:
  name: "{{ cookiecutter.project_name }}"

environments:
  development:
    profile: development
    resources:
      workspace_directory: "/Shared/{{ cookiecutter.project_name }}/dev"
      catalog: "development"
      schema: "{{ cookiecutter.project_name }}"
      tags:
        environment: "development"
    targets:
```

```yaml
      dev:
        workspace_host: "${env:DEV_WORKSPACE_HOST}"
        aws_attributes:
          instance_profile_arn: "${env:DEV_INSTANCE_PROFILE_ARN}"

  staging:
    profile: staging
    resources:
      workspace_directory: "/Shared/{{ cookiecutter.project_name }}/staging"
      catalog: "non_published_"
      schema: "{{ cookiecutter.project_name }}"
      tags:
        environment: "staging"
    targets:
      staging:
        workspace_host: "${env:STAGING_WORKSPACE_HOST}"
        aws_attributes:
          instance_profile_arn: "${env:STAGING_INSTANCE_PROFILE_ARN}"

  production:
    profile: production
    resources:
      workspace_directory: "/Shared/{{ cookiecutter.project_name }}/prod"
      catalog: "published_"
      schema: "{{ cookiecutter.project_name }}"
      tags:
        environment: "production"
    targets:
      prod:
        workspace_host: "${env:PROD_WORKSPACE_HOST}"
        aws_attributes:
          instance_profile_arn: "${env:PROD_INSTANCE_PROFILE_ARN}"

resources:
  jobs:
    feature_engineering_job: "resources/jobs/feature_engineering_job.yml"
    model_training_job: "resources/jobs/model_training_job.yml"
    model_inference_job: "resources/jobs/model_inference_job.yml"
    model_monitoring_job: "resources/jobs/model_monitoring_job.yml"

  experiments:
    mlflow_experiment: "resources/experiments/experiment.yml"

  models:
```

```
    registered_model: "resources/models/model.yml"

  endpoints:
    serving_endpoint: "resources/endpoints/endpoint.yml"
```

10. Example Usage

10.1 Project Initialization

```
# Create a new project from the cookie-cutter template
cookiecutter gh:your-org/enterprise-mlops-template

# Initialize git repository
cd your-new-project
git init
git add .
git commit -m "Initial commit"

# Create virtual environment and install dependencies
pip install uv
uv venv create
source .venv/bin/activate  # On Windows: .venv\Scripts\activate

# Install the project with development dependencies
uv pip install -e ".[dev]"

# Configure AWS credentials
aws configure

# Set up environment variables for Databricks workspaces
export DEV_WORKSPACE_HOST="https://your-dev-workspace.cloud.databricks.com"
export
DEV_INSTANCE_PROFILE_ARN="arn:aws:iam::123456789012:instance-profile/dev-profile"
export STAGING_WORKSPACE_HOST="https://your-staging-workspace.cloud.databricks.com"
export
STAGING_INSTANCE_PROFILE_ARN="arn:aws:iam::123456789012:instance-profile/staging-p
rofile"
```

10.2 Data Source Configuration

Create configuration for data sources:

```yaml
# conf/data_sources/dev.yml
data_sources:
  customer_data:
    type: "delta"
    config:
      # No specific config needed for Delta tables

  transactions:
    type: "snowflake"
    config:
      url: "${env:SNOWFLAKE_URL}"
      user: "${env:SNOWFLAKE_USER}"
      password: "${env:SNOWFLAKE_PASSWORD}"
      warehouse: "COMPUTE_WH"
      database: "SALES"
      schema: "PUBLIC"

  product_catalog:
    type: "s3"
    config:
      bucket: "enterprise-data-lake"
      region: "us-west-2"
      role: "${env:AWS_ROLE_ARN}"
```

10.3 Feature Engineering Example

```python
# Example feature engineering script
from churn_prediction.common.config import EnvironmentConfig
from churn_prediction.data.data_loader import DataLoader
from churn_prediction.features.feature_store import FeatureStoreManager
from pyspark.sql import SparkSession
from pyspark.sql.functions import datediff, current_date, col, when
from pyspark.sql.functions import sum as spark_sum, count, avg, max as spark_max

# Initialize Spark
spark = SparkSession.builder.getOrCreate()

# Load configuration
config = EnvironmentConfig.from_file("conf/deployment/dev.yml")

# Initialize components
data_loader = DataLoader("conf/data_sources/dev.yml", spark)
feature_store = FeatureStoreManager(config, spark)
```

```
# Load customer data
customer_df = data_loader.load_data(
    source_name="customer_data",
    object_name="customers"
)

# Load transaction data
transactions_df = data_loader.load_data(
    source_name="transactions",
    object_name="transactions",
    filters=["transaction_date > '2023-01-01'"]
)

# Create features
# Calculate customer tenure feature
customer_features = customer_df.withColumn(
    "tenure_days",
    datediff(current_date(), col("signup_date"))
)

# Calculate transaction aggregates
transaction_aggs = transactions_df.groupBy("customer_id").agg(
    count("transaction_id").alias("transaction_count"),
    spark_sum("amount").alias("total_spend"),
    avg("amount").alias("avg_transaction_value"),
    spark_max("transaction_date").alias("last_transaction_date")
)

# Join with customer data
customer_features = customer_features.join(
    transaction_aggs,
    on="customer_id",
    how="left"
)

# Calculate days since last transaction
customer_features = customer_features.withColumn(
    "days_since_last_transaction",
    datediff(current_date(), col("last_transaction_date"))
)

# Register in Feature Store
feature_store.create_feature_table(
    df=customer_features,
```

```
    name="customer_churn_features",
    primary_keys=["customer_id"],
    description="Features for customer churn prediction",
    tags={"domain": "customer", "usecase": "churn_prediction"}
)

print("Feature engineering completed successfully")
```

10.4 Model Training with Governance

```
# Example model training script with model cards and governance
from churn_prediction.common.config import EnvironmentConfig
from churn_prediction.pipelines.training_pipeline import EnhancedTrainingPipeline

# Load configuration
config = EnvironmentConfig.from_file("conf/deployment/dev.yml")

# Initialize training pipeline
pipeline = EnhancedTrainingPipeline(config)

# Define model configuration (generic, not hardcoded to specific tables)
model_config = {
    "data_source": {
        "source_name": "customer_data",
        "object_name": "customers",
        "label_column": "churn",
        "filters": ["active = true"]
    },
    "feature_store": {
        "feature_table": "customer_churn_features",
        "entity_keys": ["customer_id"],
        "exclude_columns": []
    },
    "model": {
        "name": "customer_churn_predictor",
        "type": "RandomForestClassifier",
        "hyperparameters": {
            "n_estimators": 100,
            "max_depth": 10,
            "min_samples_split": 5,
            "random_state": 42
        },
        "tags": {
            "domain": "customer_analytics",
```

```
            "criticality": "high"
        }
    },
    "governance": {
        "owner": "analytics_team",
        "use_case": "Predict customer churn to enable targeted retention campaigns",
        "limitations": [
            "Model does not account for seasonal patterns",
            "Limited to customers with transaction history"
        ],
        "ethical_considerations": [
            "Ensure retention offers are not exploitative",
            "Monitor demographic fairness metrics"
        ]
    }
}

# Train model with governance
result = pipeline.train(model_config)

print(f"Model {result['model_name']} trained and registered with version {result['version']}")
print(f"Metrics: {result['metrics']}")
print(f"Model card created: {result['model_card_created']}")
```

10.5 Deployment with Databricks Asset Bundles

Define model training job:

```
# resources/jobs/model_training_job.yml
job_clusters:
  - job_cluster_key: "default"
    new_cluster:
      spark_version: "13.0.x-gpu-ml-scala2.12"
      node_type_id: "i3.xlarge"
      aws_attributes:
        availability: "SPOT_WITH_FALLBACK"
        zone_id: "auto"
        spot_bid_price_percent: 100
      autoscale:
        min_workers: 1
        max_workers: 4
      spark_conf:
        "spark.databricks.cluster.profile": "singleNode"
```

```
        "spark.master": "local[*]"

tasks:
  - task_key: "training"
    job_cluster_key: "default"
    python_wheel_task:
      package_name: "churn_prediction"
      entry_point: "train_model"
      parameters: ["--config", "${bundle.environment}"]
    libraries:
      - whl: "dbfs:/FileStore/wheels/churn_prediction-0.1.0-py3-none-any.whl"

schedule:
  quartz_cron_expression: "0 0 0 * * ?"
  timezone_id: "UTC"
  pause_status: "UNPAUSED"
```

Deploy to development environment:

```
# Build the package
python -m build

# Upload wheel file to DBFS
databricks fs cp dist/churn_prediction-0.1.0-py3-none-any.whl dbfs:/FileStore/wheels/

# Validate bundle
databricks bundle validate -t dev

# Deploy to development environment
databricks bundle deploy -t dev

# Run the training job
databricks jobs run-now --job-id $(databricks jobs list --output json | jq -r '.jobs[] |
select(.settings.name=="model_training_job") | .job_id')
```

10.6 Model Promotion and Deployment

```
# Example model promotion script
from churn_prediction.common.config import EnvironmentConfig
from churn_prediction.models.registry import EnhancedModelRegistry
from churn_prediction.serving.endpoints import ModelServingManager

# Load configurations
```

```
dev_config = EnvironmentConfig.from_file("conf/deployment/dev.yml")
staging_config = EnvironmentConfig.from_file("conf/deployment/staging.yml")

# Initialize registry and serving managers
dev_registry = EnhancedModelRegistry(dev_config)
staging_registry = EnhancedModelRegistry(staging_config)
serving_manager = ModelServingManager(staging_config)

# Get the model to promote
models = dev_registry.list_models()
churn_model = next(m for m in models if m["name"] == "customer_churn_predictor")
latest_version = churn_model["versions"][0]["version"]

# Transition to staging in dev
dev_registry.transition_stage(
    model_name="customer_churn_predictor",
    version=latest_version,
    stage="Staging"
)

# Get model details with run ID
model_details = dev_registry.client.get_model_version(
    name=f"{dev_config.catalog}.{dev_config.schema}.customer_churn_predictor",
    version=latest_version
)

# Register in staging environment
staging_version = staging_registry.register_model(
    run_id=model_details.run_id,
    model_name="customer_churn_predictor",
    description=f"Promoted from development version {latest_version}"
)

# Deploy to endpoint
endpoint_name = serving_manager.deploy_endpoint(
    model_name="customer_churn_predictor",
    version=staging_version,
    workload_size="Small",
    scale_to_zero=True
)

print(f"Model deployed to endpoint: {endpoint_name}")
```

11. Conclusion

This enterprise-grade MLOps framework for Databricks on AWS provides a comprehensive solution for managing machine learning lifecycles with:

1. **Flexible Data Integration**: Generic connectors for any data source including Snowflake, Delta tables, and S3
2. **Enhanced Governance**: Complete model cards, lineage tracking, and documentation
3. **Feature Store Integration**: Centralized feature management with versioning
4. **Model Registry**: Advanced model versioning with approval workflows
5. **Infrastructure as Code**: Everything defined in Databricks Asset Bundles
6. **Python-First Approach**: Pure Python modules for better testing and version control
7. **Cookie-Cutter Pattern**: Consistent project structure across all ML initiatives

By implementing this framework using the cookie-cutter pattern, organizations can standardize their ML development practices while ensuring reliability, compliance, and governance across all environments. The framework's pluggable architecture allows teams to easily extend its capabilities by adding new data connectors, model types, or governance features while maintaining a consistent structure.