Project MLE-004: SageMaker Model Registry with CI/CD Pipeline

1. Project Overview

This project demonstrates the implementation of a secure and automated end-to-end Machine Learning (ML) workflow on AWS. It establishes a CI/CD pipeline using AWS developer tools to train, register, approve, and deploy a scikit-learn model using Amazon SageMaker. The pipeline automates the process of moving a model from development to production, ensuring consistency, reliability, and governance.

1.1 Business Context

In many organizations, the process of deploying machine learning models is manual, slow, and prone to errors. This can lead to significant delays in delivering value to the business and introduces operational risks. This project addresses these challenges by creating a robust MLOps foundation that enables data science and engineering teams to:

- **Accelerate Time-to-Market** — Automate the entire model lifecycle, from code commit to production deployment.
- **Improve Model Quality** — Enforce automated checks and approval gates to ensure only high-performing models are deployed.
- **Enhance Governance and Compliance** — Maintain a central, version-controlled repository of all models, tracking their lineage and approval status.
- **Reduce Operational Overhead** — Minimize manual intervention, freeing up teams to focus on developing and improving models.

The model in this project is a placeholder (a churn prediction classifier), but the infrastructure is designed to be a reusable pattern for a wide range of real-world ML applications, such as fraud detection, customer segmentation, or predictive maintenance.

1.2 GRC (Governance, Risk, and Compliance) Mapping

This project directly supports several key GRC objectives by implementing controls that align with common industry frameworks (NIST, ISO 27001, SOC 2).

GRC Domain	Control Objective	Implementation
Change Management	Ensure all changes to production systems are authorized, tested, and documented.	AWS CodePipeline orchestrates the entire workflow. Changes are triggered by commits to a CodeCommit repository, providing a full audit trail. The Manual Approval stage acts as a formal gate before production deployment.
Access Control	Implement the principle of least privilege for all system components.	IAM Roles are created with narrowly defined permissions for CodePipeline, CodeBuild, and SageMaker. Each component has only the permissions it needs to perform its specific function.
Asset Management	Maintain a centralized inventory of all approved ML models.	SageMaker Model Registry acts as a central catalog for all registered model versions. Each model has a unique ARN, metadata, and an explicit approval status (Pending, Approved, Rejected).
Secure Configuration	Ensure that infrastructure is configured securely and consistently.	CodeBuild uses `buildspec.yml` files to define the build and deployment environment as code. This ensures the process is repeatable and auditable.
Audit & Accountability	Log all actions and provide a clear audit trail for security and compliance reviews.	AWS CloudTrail logs every API call. CodePipeline provides a visual history of every execution, including who approved changes and when deployments occurred.

---

2. Architecture

The diagram below illustrates the CI/CD pipeline architecture.

Workflow:

1. **Source** — A developer pushes code changes (e.g., `train.py`, `buildspec.yml`) to the `main` branch of the AWS CodeCommit repository.
2. **Pipeline Trigger** — The commit automatically triggers AWS CodePipeline.
3. **Build Stage** — CodePipeline starts a CodeBuild project. The build environment installs necessary Python libraries, executes `train.py` to train a new model, and registers the trained model to the SageMaker Model Registry with the status `PendingManualApproval`. The ARN of the new model package is saved as an artifact.
4. **Approval Stage** — The pipeline pauses for approval.
    - **Manual Approval:** An operator reviews the model's performance metrics in the SageMaker console and manually approves or rejects the deployment.
    - **Automated Approval:** An EventBridge rule, triggered by the new model package, invokes a Lambda function. The Lambda function evaluates the model's metrics against predefined thresholds (e.g., accuracy > 85%) and automatically updates the model package status to `Approved` or `Rejected`.
5. **Deploy Stage** — If the model is approved, a second CodeBuild project reads the model package ARN from the build artifact and deploys the model to a real-time SageMaker Endpoint.
6. **Production** — The model is live and ready to serve predictions.

---

3. Prerequisites

Before you begin, ensure you have the following:

- An AWS account with administrator privileges.
- The AWS CLI installed and configured on your local machine.
- A basic understanding of AWS IAM, S3, CodeCommit, CodeBuild, CodePipeline, and SageMaker.
- All resources should be created in the **us-east-1** region for consistency.

---

4. IAM Roles and Policies

This project requires five IAM roles to delegate permissions securely to the AWS services involved. Create each role with the trust relationship and permissions policy specified below before creating the pipeline.

4.1 CodePipeline Service Role — `CodePipeline-MLE-004-Role`

This role allows CodePipeline to interact with CodeCommit, CodeBuild, and S3.

Trust Relationship:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "codepipeline.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Permissions Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject", "s3:GetObjectVersion",
                "s3:GetBucketVersioning", "s3:PutObjectAcl", "s3:PutObject"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:GetBranch", "codecommit:GetCommit",
                "codecommit:UploadArchive",
"codecommit:GetUploadArchiveStatus",
                "codecommit:CancelUploadArchive"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["codebuild:BatchGetBuilds", "codebuild:StartBuild"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["lambda:InvokeFunction", "lambda:ListFunctions"],
            "Resource": "*"
        }
    ]
}
```

4.2 CodeBuild Training Role — `CodeBuildTrainRole`

This role is assumed by the CodeBuild project in the **Build** stage. It needs permissions to interact with SageMaker, S3, and CloudWatch Logs.

Trust Relationship:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "codebuild.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Permissions Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        { "Effect": "Allow", "Action": ["sagemaker:*"], "Resource": "*" },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject", "s3:PutObject",
                "s3:GetBucketLocation", "s3:CreateBucket", "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup", "logs:CreateLogStream",
"logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": { "iam:PassedToService":
"sagemaker.amazonaws.com" }
            }
        }
    ]
}
```

4.3 CodeBuild Deploy Role — `CodeBuildDeployRole`

This role is assumed by the CodeBuild project in the **Deploy** stage. It requires permissions to create SageMaker models and endpoints.

Trust Relationship:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "codebuild.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Permissions Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateModel", "sagemaker:CreateEndpoint",
                "sagemaker:CreateEndpointConfig",
"sagemaker:DescribeModelPackage"
            ],
            "Resource": "*"
        },
        { "Effect": "Allow", "Action": ["s3:GetObject"], "Resource": "*" },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup", "logs:CreateLogStream",
"logs:PutLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": { "iam:PassedToService":
"sagemaker.amazonaws.com" }
            }
        }
    ]
}
```

4.4 SageMaker Execution Role — `SageMaker-MLE-004-Role`

This role is assumed by SageMaker training jobs and endpoints. It needs access to S3 to read data and write model artifacts.

Trust Relationship:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "sagemaker.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Permissions Policy: Attach the `AmazonSageMakerFullAccess` managed policy. For a production environment, create a more restrictive custom policy.

4.5 Lambda Auto-Approval Role — `Lambda-AutoApprove-Role`

This role is assumed by the automated approval Lambda function.

Trust Relationship:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Permissions Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeModelPackage",
                "sagemaker:ListModelPackages",
                "sagemaker:UpdateModelPackage"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutJobSuccessResult",
                "codepipeline:PutJobFailureResult"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup", "logs:CreateLogStream",
"logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

---

5. Step-by-Step Deployment Guide

Step 1: Set Up the CodeCommit Repository

1. Create a CodeCommit repository named `model-registry-pipeline-repo`.
2. Clone the repository to your local machine.
3. Add the following files to the repository: `create_model_package.py`, `buildspec-train.yml`, `buildspec-deploy.yml`, and

`lambda_auto_approve.py`.

4. Commit and push the files to the `main` branch.

Step 2: Create the IAM Roles

Create the five IAM roles with the trust relationships and permissions policies detailed in Section 4 above.

Step 3: Create the CodeBuild Projects

Create two CodeBuild projects with the following settings:

Setting	`train-model` Project	`deploy-model` Project
Source	AWS CodeCommit, `model-registry-pipeline-repo`, `main`	AWS CodeCommit, `model-registry-pipeline-repo`, `main`
Environment	Managed image, Ubuntu, Standard, `aws/codebuild/standard:5.0`	Managed image, Ubuntu, Standard, `aws/codebuild/standard:5.0`
Service Role	`CodeBuildTrainRole`	`CodeBuildDeployRole`
Buildspec	`buildspec-train.yml`	`buildspec-deploy.yml`

Step 4: Create the CodePipeline

Part 1 — Create the Pipeline Using the Wizard

1. Select **"Build custom pipeline"**.
2. Set the **Pipeline name** to `mle-004-model-registry-pipeline`, **Pipeline type** to V2, **Execution mode** to Superseded, and **Service role** to `CodePipeline-MLE-004-Role`. Click **Next**.
3. Set **Source provider** to AWS CodeCommit, **Repository name** to `model-registry-pipeline-repo`, and **Branch name** to `main`. Click **Next**.

4. Set **Build provider** to AWS CodeBuild, **Region** to US East (N. Virginia), and **Project name** to `train-model`. Click **Next**.
5. Click **"Skip test stage"** and **"Skip deploy stage"**.
6. Review the settings and click **"Create pipeline"**.

> *The pipeline will be created with only Source → Build stages. Allow it to complete its first run before proceeding to Part 2.*

Part 2 — Add the Approval and Deploy Stages via Edit

After the pipeline is created, click the **"Edit"** button at the top right of the pipeline page.

Add the Approval Stage:

1. Click **"+ Add stage"** below the Build stage.
2. Set **Stage name** to `Approval` and click **"Add stage"**.
3. Inside the new stage, click **"+ Add action group"**.
4. Set **Action name** to `ManualApproval` and **Action provider** to Manual approval. Optionally add an SNS topic ARN for email notifications.
5. Click **"Done"**.

Add the Deploy Stage:

1. Click **"+ Add stage"** below the Approval stage.
2. Set **Stage name** to `Deploy` and click **"Add stage"**.
3. Inside the new stage, click **"+ Add action group"**.
4. Configure the action as follows:
   - **Action name:** `DeployModel`
   - **Action provider:** AWS CodeBuild
   - **Region:** US East (N. Virginia)
   - **Input artifacts:** Add two artifacts — `SourceArtifact` (primary, contains `buildspec-deploy.yml`) and `BuildArtifact` (secondary, contains `model_package_arn.txt`)
   - **Project name:** `deploy-model`

- **Build type:** Single build
- **Output artifacts:** Leave blank

5. Click **"Done"**, then click **"Save"** at the top of the Edit screen and confirm.

> ***Why two input artifacts?*** *The `buildspec-deploy.yml` file lives in your CodeCommit repo (delivered as `SourceArtifact`). The `model_package_arn.txt` file is produced by the Build stage (delivered as `BuildArtifact`). CodeBuild needs both to complete the deployment.*

Final Pipeline Structure

After completing both parts, your pipeline should look like this:

```
Source  (CodeCommit → model-registry-pipeline-repo)
   ↓
Build   (CodeBuild → train-model)
   ↓
Approval (Manual Approval)
   ↓
Deploy  (CodeBuild → deploy-model)
```

Stage	Action Provider	Input Artifact(s)	Output Artifact
Source	AWS CodeCommit	—	`SourceArtifact`
Build	AWS CodeBuild (`train-model`)	`SourceArtifact`	`BuildArtifact`
Approval	Manual approval	—	—
Deploy	AWS CodeBuild (`deploy-model`)	`SourceArtifact` + `BuildArtifact`	—

Step 5 (Option B): Automated Approval Workflow

1. Create a Lambda function named `mle-004-auto-approve-model` with Runtime Python 3.11, Execution role `Lambda-AutoApprove-Role`, and code from

`lambda_auto_approve.py`.

2. Create an EventBridge rule named `mle-004-model-registry-rule` with the following event pattern:

```
{
    "source": ["aws.sagemaker"],
    "detail-type": ["SageMaker Model Package State Change"],
    "detail": {
        "ModelPackageGroupName": ["ChurnPredictionModels"],
        "ModelApprovalStatus": ["PendingManualApproval"]
    }
}
```

1. Set the target to the `mle-004-auto-approve-model` Lambda function.
2. Modify the CodePipeline: remove the `ManualApproval` stage and add an `AutoApproval` stage with an AWS Lambda action pointing to `mle-004-auto-approve-model`.

Step 6: Trigger and Verify

1. Push a change to the CodeCommit repository to trigger the pipeline.
2. Monitor the pipeline execution in the CodePipeline console.
3. Verify that the model is approved (either manually or automatically).
4. Verify that the SageMaker endpoint is created successfully.

---

6. Code Explanation

6.1 `create_model_package.py`

This script is the core of the **Build** stage. It performs the following actions:

- **Creates a Dummy Model** — Generates synthetic data using `sklearn.datasets.make_classification` and trains a

`RandomForestClassifier`. In a real-world scenario, this would be replaced with actual model training code.

- **Packages the Model** — The trained model is saved as `model.pkl` and packaged into a `model.tar.gz` archive, the standard format for deploying custom models on SageMaker.
- **Uploads to S3** — The model artifact is uploaded to the default SageMaker S3 bucket with a unique timestamp to ensure versioning.
- **Creates Model Package Group** — Ensures that a `ModelPackageGroup` named `ChurnPredictionModels` exists as a container for different versions of the same model.
- **Creates Model Package** — Creates a new model package within the group, including metadata about the model, the S3 location of the artifact, the container image for inference, and supported instance types. The `ModelApprovalStatus` is set to `PendingManualApproval`.
- **Saves Model Package ARN** — The ARN of the newly created model package is saved to `model_package_arn.txt` and passed as an artifact to the Deploy stage.

6.2 `buildspec-train.yml`

This file defines the build commands for the **Build** stage in CodeBuild.

- **`install` phase** — Installs the necessary Python libraries: `boto3`, `sagemaker`, and `scikit-learn`.
- **`build` phase** — Executes `create_model_package.py`, which trains the model and registers it in the SageMaker Model Registry.
- **`artifacts` section** — Specifies that `model_package_arn.txt` should be saved as a build artifact, making the model package ARN available to subsequent pipeline stages.

6.3 `buildspec-deploy.yml`

This file defines the build commands for the **Deploy** stage.

- **`install` phase** — Installs `boto3`, the only dependency needed for this stage.

- **`build` phase** — Contains an embedded Python script that reads `model_package_arn.txt` to get the ARN of the approved model package, calls `sm_client.create_model()` to create a SageMaker Model, calls `sm_client.create_endpoint_config()` to define the production endpoint configuration, and finally calls `sm_client.create_endpoint()` to deploy the model to a real-time inference endpoint.

6.4 `lambda_auto_approve.py`

This Lambda function automates the model approval process.

- **Event-Driven Logic** — Can be triggered by EventBridge (when a new model package enters `PendingManualApproval`) or by CodePipeline directly (receiving a `jobId` to report success or failure).
- **Evaluation Logic** — The `evaluate_and_approve_model` function retrieves the model package details, including performance metrics attached during registration. In this demo it automatically approves any model, but in production you would add logic to check if metrics (e.g., accuracy, F1-score) meet predefined thresholds.
- **Update Model Status** — Based on the evaluation, the function calls `sagemaker_client.update_model_package()` to set the `ModelApprovalStatus` to either `Approved` or `Rejected`.

End of Document — Project MLE-004