Comprehensive Deployment Guide: Intelligent Document Classification Pipeline

Project: PRJ-MLE-001

---

1. Introduction

This guide provides a detailed, step-by-step walkthrough for deploying the **Intelligent Document Classification Pipeline** on AWS. This project is an excellent demonstration of building a serverless, event-driven machine learning workflow, suitable for an AWS Certified Machine Learning Engineer portfolio.

1.1. Project Goal

The primary goal is to create an automated system that ingests documents (PDFs, images), extracts their text content, classifies them into predefined categories using a custom-trained machine learning model, and stores the results for analysis. This entire process is orchestrated using serverless technologies to ensure scalability, resilience, and cost-effectiveness.

1.2. Architecture Overview

The pipeline is composed of several managed AWS services integrated to form a seamless workflow:

- **Amazon S3:** Acts as the entry point for documents and storage for model training data.
- **AWS Step Functions:** Orchestrates the entire multi-step classification process, providing state management and error handling.
- **AWS Lambda:** Provides the serverless compute for individual tasks like initiating jobs and processing results.
- **Amazon Textract:** Extracts text from uploaded documents.
- **Amazon SageMaker:** Used to train, host, and serve predictions from our custom BlazingText classification model.
- **Amazon DynamoDB:** Stores the final classification results.
- **Amazon EventBridge:** Triggers the pipeline when a new document is uploaded to S3.
- **AWS IAM:** Manages the permissions for all services to interact securely.

Architecture Diagram

---

2. Prerequisites

Before you begin, ensure you have the following set up and configured.

2.1. AWS Account and Permissions

- An active AWS account.
- An IAM user with administrative privileges (for ease of setup in a personal account) or specific permissions to create and manage S3, IAM, Lambda, SageMaker, Step Functions, DynamoDB, and EventBridge resources.

2.2. Local Environment

- **AWS CLI:** The AWS Command Line Interface installed and configured. You can verify the installation by running `aws --version`. To configure it, run `aws configure` and provide your AWS Access Key ID, Secret Access Key, and default region.
    - *Reference:* Installing the AWS CLI
- **Python 3.8+:** A local Python installation for running data preparation scripts.
- **wget:** A command-line utility to download the dataset. It is pre-installed on most Linux distributions and macOS.

2.3. AWS Environment

- **SageMaker Studio Domain:** This project uses a SageMaker Studio Notebook for model training. If you do not have a domain, you must create one. This can take 5-10 minutes.
    - *Reference:* Onboard to Amazon SageMaker Domain

---

3. Step-by-Step Deployment

This deployment is divided into five major parts: setting up the environment, training and deploying the model, creating the serverless backend, orchestrating the workflow, and finally, triggering the pipeline.

Part 1: Environment Setup

In this part, we will prepare our dataset, create the necessary S3 bucket, and set up the IAM roles that grant our AWS services the permissions they need to interact with each other.

Step 1.1: Prepare the Dataset

We will use a subset of the 20 Newsgroups dataset to train our text classifier.

1. **Download the Dataset:** Open your local terminal and run the following commands.

   ```
   bash wget http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz
   tar -xzvf 20news-bydate.tar.gz
   ```

2. **Create the Data Preparation Script:** Create a file named `prepare_data.py` and paste the code below. This script will read the raw text files, assign labels, and format the data as required by SageMaker BlazingText.

   ```python
   import os
   import random
   import re

# Define the categories we want to use

   selected_categories = {
   'rec.sport.hockey': 'sports',
   'sci.med': 'science',
   'comp.graphics': 'technology',
   'talk.politics.misc': 'politics'
   }

   data = []

   print("Starting data preparation...")
   ```

Walk through the directories

for category, label in selected_categories.items():
# FIX: Don't replace dots - the directory names have dots in them
dir_path = os.path.join("20news-bydate-train", category)

```
if not os.path.isdir(dir_path):
    print(f"Warning: Directory not found - {dir_path}")
    continue

print(f"Processing category: {category}")
file_count = 0

for filename in os.listdir(dir_path):
    file_path = os.path.join(dir_path, filename)
    if not os.path.isfile(file_path):
        continue

    try:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
            content = f.read()
            # Remove headers and clean up the text
            lines = content.split("\n")
            body_start_index = lines.index("") if "" in lines else 0
            body = " ".join(lines[body_start_index:])
            body = re.sub(r'\s+', ' ', body).strip()
            if body:
                data.append(f"__label__{label} {body}")
                file_count += 1
    except Exception as e:
        print(f"Could not read {file_path}: {e}")

print(f"  Processed {file_count} files from {category}")
```

print(f"\nTotal documents collected: {len(data)}")

```
if len(data) == 0:
print("\nERROR: No data was collected. Please check:")
print("1. The 20news-bydate-train directory exists")
print("2. Run: ls -la 20news-bydate-train/ to see the actual directory names")
exit(1)
```

Shuffle and split the data

```
random.shuffle(data)
split_ratio = 0.8
split_index = int(len(data) * split_ratio)

train_data = data[:split_index]
validation_data = data[split_index:]
```

Save to files

```
with open('documents.train', 'w', encoding='utf-8') as f:
for line in train_data:
f.write(line + "\n")

with open('documents.validation', 'w', encoding='utf-8') as f:
for line in validation_data:
f.write(line + "\n")

print(f"\nCreated documents.train with {len(train_data)} lines.")
print(f"Created documents.validation with {len(validation_data)} lines.")
```

3. **Run the Script:** Execute the script from your terminal.

```
bash python3 prepare_data.py
```

Step 1.2: Create S3 Bucket and IAM Roles

1. **Create S3 Bucket:** Run the following commands in your terminal to create a uniquely named S3 bucket and upload your training data.

   ```bash
   export BUCKET_NAME="mle-doc-classification-$(date +%s)"
   export AWS_REGION=$(aws configure get region)
   aws s3api create-bucket --bucket $BUCKET_NAME --region $AWS_REGION --
   create-bucket-configuration LocationConstraint=$AWS_REGION
   echo "Bucket $BUCKET_NAME created."

   aws s3 cp documents.train s3://$BUCKET_NAME/data/documents.train
   aws s3 cp documents.validation s3://$BUCKET_NAME/data/documents.validation
   echo "Training data uploaded to s3://$BUCKET_NAME/data/"
   `` **Note:** Remember your $BUCKET_NAME` for later steps.

2. **Create IAM Role for SageMaker:**
   - Navigate to the **IAM Console** in AWS.
   - Go to **Roles** and click **Create role**.
   - **Trusted entity type:** Select **AWS service**.
   - **Use case:** Choose **SageMaker**.
   - Click **Next**.
   - In the permissions search bar, find and select `AmazonSageMakerFullAccess` `AmazonS3FullAccess`.
   - Click **Next**.
   - **Role name:** `SageMaker-DocClass-Role`.
   - Click **Create role**.
   - After creation, find the role and copy its **ARN**. You will need it soon.

3. **Create IAM Role for Lambda:**
   - In the IAM Console, create another role.
   - **Trusted entity type: AWS service**.
   - **Use case: Lambda**.
   ```

- Click **Next**.
- Attach the following AWS managed policies:
  - `AWSLambda_FullAccess`
  - `AmazonS3FullAccess`
  - `AmazonDynamoDBFullAccess`
  - `AmazonSageMakerFullAccess`
  - `AmazonTextractFullAccess`
- **Note:** For a production environment, you should create a more restrictive, custom policy with only the necessary permissions.
- **Role name:** `DocClassificationLambdaRole`.
- Click **Create role**.

Part 2: Model Training and Deployment

This part is performed within a SageMaker Studio Notebook.

1. **Launch SageMaker Studio** and create a new Notebook using the `Python 3 (Data Science)` kernel.
2. **Execute the following code in separate notebook cells.**

```python

# Cell 1: Setup

import sagemaker
import boto3
from sagemaker import get_execution_role

sess = sagemaker.Session()
bucket = 'S3 Bucket' # Or replace with your BUCKET_NAME from the CLI
prefix = 'blazingtext-doc-classification'
```

IMPORTANT: Paste the SageMaker Role ARN you copied earlier

role = ""

print(f"Using bucket: {bucket}")
print(f"Using role: {role}")
```

```python

# Cell 2: Get BlazingText Container

container = sagemaker.image_uris.retrieve("blazingtext", sess.boto_region_name, "1")
print(f"Using BlazingText container: {container}")
```

```python

# Cell 3: Define Data Channels

train_data_path = f's3://{bucket}/documents.train'
validation_data_path = f's3://{bucket}/documents.validation'
s3_output_location = f's3://{bucket}/trained/output'

train_channel = sagemaker.inputs.TrainingInput(train_data_path,
content_type='text/plain')
validation_channel = sagemaker.inputs.TrainingInput(validation_data_path,
content_type='text/plain')
```

```python

# Cell 4: Configure and Run Training Job

bt_estimator = sagemaker.estimator.Estimator(container,
role,
instance_count=1,
instance_type='ml.c5.xlarge',
output_path=s3_output_location,
sagemaker_session=sess)

bt_estimator.set_hyperparameters(mode="supervised",
epochs=10,
min_count=2,
learning_rate=0.05,
vector_dim=10,
early_stopping=True,
patience=3,

```
min_epochs=5,
word_ngrams=2)

bt_estimator.fit({'train': train_channel, 'validation': validation_channel})
```

```python
# Cell 5: Deploy the Model

text_classifier = bt_estimator.deploy(initial_instance_count=1,
instance_type='ml.t2.medium')
print(f"Endpoint '{text_classifier.endpoint_name}' deployed successfully.")

# IMPORTANT: Note down this endpoint name!

endpoint_name = text_classifier.endpoint_name
```

Part 3: Create Serverless Backend

1. **Create DynamoDB Table:**
    - Navigate to the **DynamoDB Console**.
    - Click **Create table**.
    - **Table name:** `DocumentClassifications`.
    - **Partition key:** `documentId` (Type: String).
    - Leave other settings as default and click **Create table**.

2. **Create Lambda Functions:** Navigate to the **Lambda Console** and create the following three functions. For each, use the **Python 3.9** runtime and select the `DocClassificationLambdaRole` you created.

   Function 1: `start-textract`
   - **Code:**
   ```python
   import boto3

   textract = boto3.client('textract')

   def lambda_handler(event, context):
       bucket = event['Records'][0]['s3']['bucket']['name']
       key = event['Records'][0]['s3']['object']['key']

       response = textract.start_document_text_detection(
           DocumentLocation={'S3Object': {'Bucket': bucket, 'Name': key}}
       )

       return {
           'jobId': response['JobId'],
           'bucket': bucket,
           'key': key
       }
   ```

   Function 2: `process-textract`
   - **Code:**
   ```python
   import boto3
   import os
   import json

   textract = boto3.client('textract')
   sagemaker_runtime = boto3.client('sagemaker-runtime')

   ENDPOINT_NAME = os.environ.get('SAGEMAKER_ENDPOINT')
   ```

```
def lambda_handler(event, context):
    job_id = event['jobId']
    response = textract.get_document_text_detection(JobId=job_id)

    status = response['JobStatus']
    if status != 'SUCCEEDED':
        return {'status': status}

    text = ''
    for item in response['Blocks']:
        if item['BlockType'] == 'LINE':
            text += item['Text'] + ' '

    sm_response = sagemaker_runtime.invoke_endpoint(
        EndpointName=ENDPOINT_NAME,
        ContentType='application/json',
        Body=json.dumps({'instances': [text]})
    )

    predictions = json.loads(sm_response['Body'].read().decode())
    label = predictions[0]['label'][0].replace('__label__', '')

    return {
        'status': 'SUCCEEDED',
        'classification': label,
        'documentId': event['key']
    }
```

- **Configuration:** Go to the function's **Configuration > Environment variables** tab. Add a variable:
  - **Key:** `SAGEMAKER_ENDPOINT`
  - **Value:** The SageMaker endpoint name you noted down earlier.

Function 3: **`store-classification`**

- **Code:**

```python
import boto3
import os
from datetime import datetime

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table(os.environ.get('DYNAMODB_TABLE'))

def lambda_handler(event, context):
    table.put_item(
        Item={
            'documentId': event['documentId'],
            'classification': event['classification'],
            'timestamp': datetime.utcnow().isoformat()
        }
    )
    return {'status': 'SUCCESS'}
```

- **Configuration:** Add an environment variable:
  - **Key:** `DYNAMODB_TABLE`
  - **Value:** `DocumentClassifications`

Part 4: Orchestrate with Step Functions

1. Navigate to the **Step Functions Console**.
2. Click **Create state machine**.
3. Choose **Write your workflow in code**.
4. Paste the following Amazon States Language (ASL) definition. You must **replace the placeholder ARNs** with the actual ARNs of your Lambda functions.

`json { "Comment": "Document Classification Pipeline", "StartAt": "StartTextractJob", "States": { "StartTextractJob": { "Type": "Task", "Resource": "arn:aws:states:::lambda:invoke", "Parameters": { "FunctionName": "<ARN_OF_start-textract_LAMBDA>", "Payload.$":`

```
"$" }, "ResultPath": "$.textractResult", "Next":
"WaitForTextract" }, "WaitForTextract": { "Type": "Wait", "Seconds":
15, "Next": "GetTextractResults" }, "GetTextractResults": { "Type":
"Task", "Resource": "arn:aws:states:::lambda:invoke", "Parameters":
{ "FunctionName": "<ARN_OF_process-textract_LAMBDA>", "Payload":
{ "jobId.$": "$.textractResult.Payload.jobId", "key.$":
"$.textractResult.Payload.key" } }, "ResultPath":
"$.classificationResult", "Next": "CheckClassificationStatus" },
"CheckClassificationStatus": { "Type": "Choice", "Choices":
[ { "Variable": "$.classificationResult.Payload.status",
"StringEquals": "SUCCEEDED", "Next": "StoreClassification" } ],
"Default": "WaitForTextract" }, "StoreClassification": { "Type":
"Task", "Resource": "arn:aws:states:::lambda:invoke", "Parameters":
{ "FunctionName": "<ARN_OF_store-classification_LAMBDA>", "Payload":
{ "documentId.$": "$.classificationResult.Payload.documentId",
"classification.$":
"$.classificationResult.Payload.classification" } }, "End":
true } } }
```

5. Click **Next**. Name your state machine `DocumentClassificationPipeline` and create it.

Part 5: Trigger the Pipeline

1. Navigate to the **Amazon EventBridge Console**.
2. Click **Create rule**.
3. **Name:** `TriggerDocClassificationPipeline`.
4. **Event bus:** `default`.
5. **Rule type:** Select **Rule with an event pattern**.
6. **Event pattern:**
    - **Event source:** AWS services
    - **AWS service:** Simple Storage Service (S3)
    - **Event type:** Object Created
    - Click **Specific bucket(s) by name** and enter your `$BUCKET_NAME`.

7. **Target:**
    - **Target type:** AWS service
    - **Select a target:** Step Functions state machine
    - **State machine:** `DocumentClassificationPipeline`.
8. Click **Create rule**.

---

4. Testing and Monitoring

4.1. How to Use

1. **Upload a Document:** Upload a text-based image (PNG, JPG) or a PDF file to your S3 bucket.

    ```bash

# Create a dummy text file to test

    echo "This is a test document about the latest science and technology." > test.txt
    aws s3 cp test.txt s3://$BUCKET_NAME/test.txt
    ```

2. **Monitor Execution:** Go to the Step Functions console. You will see your `DocumentClassificationPipeline` state machine running. The visual workflow will show the progress of each step.
3. **Check Result:** Once the execution succeeds, navigate to the DynamoDB console and view the items in the `DocumentClassifications` table. You will find a new entry with the classification result.

4.2. Monitoring

- **Step Functions:** The visual workflow is the best place for high-level monitoring.
- **CloudWatch Logs:** Each Lambda function has a log group in CloudWatch (`/aws/lambda/<function-name>`) for detailed debugging.
- **SageMaker:** Your endpoint metrics (invocations, latency, errors) are available in CloudWatch under the `/aws/sagemaker/Endpoints` namespace.

---

5. Cleanup

To avoid incurring future charges, it is crucial to delete all the resources you have created.

1. **Delete SageMaker Endpoint:** In your SageMaker Studio notebook, run `text_classifier.delete_endpoint()`.
2. **Delete EventBridge Rule:** Go to the EventBridge console and delete the `TriggerDocClassificationPipeline` rule.
3. **Delete Step Functions State Machine:** Delete the `DocumentClassificationPipeline` state machine.
4. **Delete Lambda Functions:** Delete the `start-textract`, `process-textract`, and `store-classification` functions.
5. **Delete DynamoDB Table:** Delete the `DocumentClassifications` table.
6. **Delete IAM Roles:** Delete the `SageMaker-DocClass-Role` and `DocClassificationLambdaRole`.
7. **Empty and Delete S3 Bucket:**

   ```
   bash aws s3 rb s3://$BUCKET_NAME --force
   ```