Service Rationale Document

**AWS Service Selection and Justification**

|  |  |  |
| --- | --- | --- |
| **Component** | **Selected AWS Service** | **Justification** |
| Model Training Job | AWS Glue | supports PySpark for ETL |
| Fraud Detection API | Amazon ECS (Fargate) + API Gateway | To deploy the container and integrate the api with it |
| Model Retraining Function | AWS Lambda | Trigger any updates pushed on S3 bucket |
| Data Storage | Amazon S3 | To integrate with the other services and t push the data on it. |
| Authentication and Authorization | AWS IAM |  |
| Monitoring & Logging | Amazon CloudWatch | Provides metrics, logs, and alerting |

IAM rule

S3 bucket details:

s3://fraud-detector-bucker-123

Folders and content

glue-scripts

* fraud\_detector\_model\_trainer.py

glue-data

* credit\_card\_transaction\_data\_labeled.csv

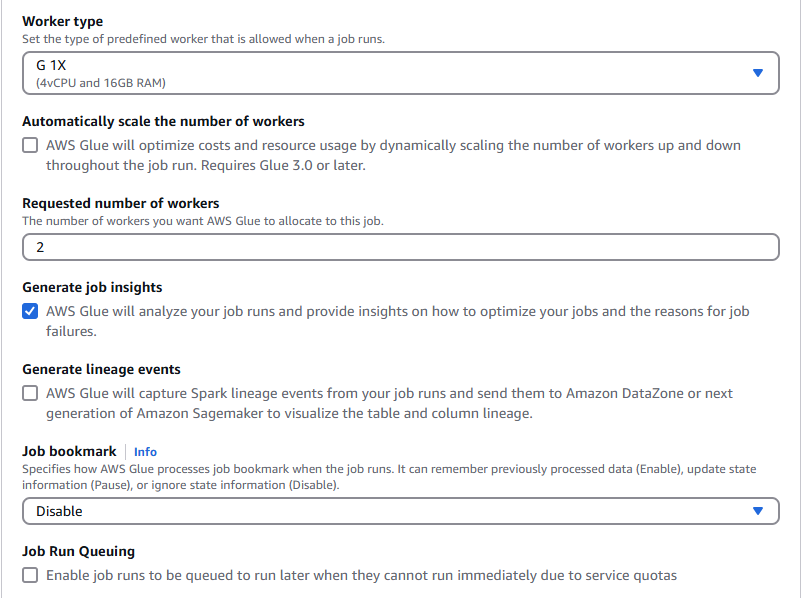
Create “GlueFraudDetectionRole” with the following policies:

* AWSGlueServiceRole
* AmazonS3FullAccess

Creating an AWS Glue Job

Job name: FraudDetectionTrainingJob

A screenshot of a computer

AI-generated content may be incorrect.  A screenshot of a computer

AI-generated content may be incorrect.Modified Script

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from pyspark.sql import SparkSession, DataFrame

from awsglue.dynamicframe import DynamicFrame

from pyspark.ml.feature import VectorAssembler, StandardScaler, StringIndexer

from pyspark.ml.classification import RandomForestClassifier

from pyspark.ml import Pipeline

from pyspark.sql.functions import col, hour, when

from pyspark.sql.types import DoubleType

glueContext = GlueContext(SparkSession.builder.appName("CreditCardFraudDetection").getOrCreate())

spark = glueContext.spark\_session

def MyTransform(glueContext, dynamic\_frame: DynamicFrame) -> DynamicFrame:

try:

df = dynamic\_frame.toDF()

numeric\_columns = [col for col in df.columns if col != 'Class']

for column in numeric\_columns:

df = df.withColumn(column, col(column).cast(DoubleType()))

assembler = VectorAssembler(inputCols=numeric\_columns, outputCol="numericFeatures")

scaler = StandardScaler(inputCol="numericFeatures", outputCol="scaledFeatures", withStd=True, withMean=True)

finalAssembler = VectorAssembler(inputCols=["scaledFeatures"], outputCol="features")

class\_counts = df.groupBy("Class").count().collect()

total\_count = sum([row['count'] for row in class\_counts])

weight\_dict = {row['Class']: total\_count / row['count'] for row in class\_counts}

df = df.withColumn("weight", when(col("Class") == 0, weight\_dict[0]).otherwise(weight\_dict[1]))

indexer = StringIndexer(inputCol="Class", outputCol="label")

rf = RandomForestClassifier(labelCol="label", featuresCol="features", numTrees=100, maxDepth=10, weightCol="weight")

pipeline = Pipeline(stages=[assembler, scaler, finalAssembler, indexer, rf])

model = pipeline.fit(df)

model\_path = 's3://fraud-detector-bucker-123/model/fraud\_detection\_model\_latest' # corrected s3 path

model.write().mode("overwrite").save(model\_path)

print(f'Model trained and saved to {model\_path}')

dynamic\_frame\_output = DynamicFrame.fromDF(df, glueContext, "output\_dynamic\_frame")

return dynamic\_frame\_output

except Exception as e:

print(f"Error: {e}")

return None