### Set up a SageMaker Studio notebook

We'll need to set up access to the model artifacts stored in Amazon S3. To achieve this, you can copy and run the following code snippet in a new cell of your notebook. This code creates an S3 client object and defines the specific locations within your default S3 bucket where the trained models are saved.

One important point to note: the SageMaker session object (see line 16) automatically creates the bucket named sagemaker-<your-Region>-<your-account-id>. This bucket will store the trained model artifacts.

On the other hand, the training datasets reside in a public S3 bucket named sagemaker-sample-files (specified as the read bucket in line 29). The read prefix parameter further narrows down the location of the datasets within that bucket.

import pandas as pd

import numpy as np

import boto3

import sagemaker

import time

import json

import io

from io import StringIO

import base64

import pprint

import re

from sagemaker.image\_uris import retrieve

sess = sagemaker.Session()

write\_bucket = sess.default\_bucket()

write\_prefix = "fraud-detect-demo"

region = sess.boto\_region\_name

s3\_client = boto3.client("s3", region\_name=region)

sm\_client = boto3.client("sagemaker", region\_name=region)

sm\_runtime\_client = boto3.client("sagemaker-runtime")

sm\_autoscaling\_client = boto3.client("application-autoscaling")

sagemaker\_role = sagemaker.get\_execution\_role()

# S3 locations used for parameterizing the notebook run

read\_bucket = "sagemaker-sample-files"

read\_prefix = "datasets/tabular/synthetic\_automobile\_claims"

model\_prefix = "models/xgb-fraud"

data\_capture\_key = f"{write\_prefix}/data-capture"

# S3 location of trained model artifact

model\_uri = f"s3://{read\_bucket}/{model\_prefix}/fraud-det-xgb-model.tar.gz"

# S3 path where data captured at endpoint will be stored

data\_capture\_uri = f"s3://{write\_bucket}/{data\_capture\_key}"

# S3 location of test data

test\_data\_uri = f"s3://{read\_bucket}/{read\_prefix}/test.csv"

Create a Real-Time Inference endpoint

Within SageMaker, various approaches exist for deploying a trained model to a Real-Time Inference endpoint: the SageMaker SDK, the AWS SDK - Boto3, and the SageMaker console. The SageMaker SDK offers more abstractions, while the AWS SDK - Boto3 provides lower-level APIs for greater control over model deployment. This tutorial demonstrates how to deploy the model using the AWS SDK - Boto3. Follow these three steps sequentially to deploy a model:

1. Create a SageMaker model using the model artifact.
2. Set up an endpoint configuration, defining properties like instance type and count.
3. Create the endpoint using the endpoint configuration.

To create a SageMaker model using the trained model artifacts stored in Amazon S3, use the following code. The create\_model method requires the Docker container with the training image (in this case, the XGBoost container) and the Amazon S3 location of the model artifacts as parameters.

# Retrieve the SageMaker managed XGBoost image

training\_image = retrieve(framework="xgboost", region=region, version="1.3-1")

# Specify a unique model name that does not exist

model\_name = "fraud-detect-xgb"

primary\_container = {

"Image": training\_image,

"ModelDataUrl": model\_uri

}

model\_matches = sm\_client.list\_models(NameContains=model\_name)["Models"]

if not model\_matches:

model = sm\_client.create\_model(ModelName=model\_name,

PrimaryContainer=primary\_container,

ExecutionRoleArn=sagemaker\_role)

else:

print(f"Model with name {model\_name} already exists! Change model name to create new")

You can view the created model in the SageMaker console under the Models section.

After creating the SageMaker model, use the following code to configure the endpoint with the Boto3 create\_endpoint\_config method. This method primarily requires the endpoint configuration name and variant information, such as the inference instance type and count, the name of the model to be deployed, and the traffic share the endpoint should handle. Additionally, you can enable data capture by specifying a DataCaptureConfig. This feature allows the real-time endpoint to capture and store requests and/or responses in Amazon S3. Data capture is a key step in setting up model monitoring. When combined with baseline metrics and monitoring jobs, it helps monitor model performance by comparing test data metrics with baselines. This monitoring is useful for scheduling model retraining based on model or data drift and for auditing purposes. In the current setup, both the input (incoming test data) and output (model predictions) are captured and stored in your default S3 bucket.

# Endpoint Config name

endpoint\_config\_name = f"{model\_name}-endpoint-config"

# Endpoint config parameters

production\_variant\_dict = {

"VariantName": "Alltraffic",

"ModelName": model\_name,

"InitialInstanceCount": 1,

"InstanceType": "ml.m5.xlarge",

"InitialVariantWeight": 1

}

# Data capture config parameters

data\_capture\_config\_dict = {

"EnableCapture": True,

"InitialSamplingPercentage": 100,

"DestinationS3Uri": data\_capture\_uri,

"CaptureOptions": [{"CaptureMode" : "Input"}, {"CaptureMode" : "Output"}]

}

# Create endpoint config if one with the same name does not exist

endpoint\_config\_matches = sm\_client.list\_endpoint\_configs(NameContains=endpoint\_config\_name)["EndpointConfigs"]

if not endpoint\_config\_matches:

endpoint\_config\_response = sm\_client.create\_endpoint\_config(

EndpointConfigName=endpoint\_config\_name,

ProductionVariants=[production\_variant\_dict],

DataCaptureConfig=data\_capture\_config\_dict

)

else:

print(f"Endpoint config with name {endpoint\_config\_name} already exists! Change endpoint config name to create new")

You can view the created endpoint configuration in the SageMaker console under the Endpoint configurations section.

Use the following code to create the endpoint. The create\_endpoint method uses the endpoint configuration as a parameter and deploys the model specified in this configuration to a compute instance. The deployment process takes approximately 6 minutes.

endpoint\_name = f"{model\_name}-endpoint"

endpoint\_matches = sm\_client.list\_endpoints(NameContains=endpoint\_name)["Endpoints"]

if not endpoint\_matches:

endpoint\_response = sm\_client.create\_endpoint(

EndpointName=endpoint\_name,

EndpointConfigName=endpoint\_config\_name

)

else:

print(f"Endpoint with name {endpoint\_name} already exists! Change endpoint name to create new")

resp = sm\_client.describe\_endpoint(EndpointName=endpoint\_name)

status = resp["EndpointStatus"]

while status == "Creating":

print(f"Endpoint Status: {status}...")

time.sleep(60)

resp = sm\_client.describe\_endpoint(EndpointName=endpoint\_name)

status = resp["EndpointStatus"]

print(f"Endpoint Status: {status}")

### Invoke the inference endpoint

After the endpoint status changes to InService, you can invoke it using the REST API, AWS SDK - Boto3, SageMaker Studio, AWS CLI, or SageMaker Python SDK. In this tutorial, you'll use the AWS SDK - Boto3. Before calling an endpoint, ensure that the test data is properly formatted using serialization and deserialization. Serialization converts raw data, such as a .csv file, into byte streams that the endpoint can process. Deserialization reverses this process, converting byte streams back into a human-readable format.

In this tutorial, you will invoke the endpoint by sending the first five samples from a test dataset. To do this and obtain prediction results, use the following code. The request to the endpoint (test dataset) is in .csv format, so a CSV serialization process is used to create the payload. The response is then deserialized into an array of predictions. Once the execution completes, the cell will return the model predictions and the true labels for the test samples. Note that the XGBoost model returns probabilities instead of actual class labels. The model predicts a very low likelihood of the test samples being fraudulent claims, and these predictions align with the true labels.

# Fetch test data to run predictions with the endpoint

test\_df = pd.read\_csv(test\_data\_uri)

# For content type text/csv, payload should be a string with commas separating the values for each feature

# This is the inference request serialization step

# CSV serialization

csv\_file = io.StringIO()

test\_sample = test\_df.drop(["fraud"], axis=1).iloc[:5]

test\_sample.to\_csv(csv\_file, sep=",", header=False, index=False)

payload = csv\_file.getvalue()

response = sm\_runtime\_client.invoke\_endpoint(

EndpointName=endpoint\_name,

Body=payload,

ContentType="text/csv",

Accept="text/csv"

)

# This is the inference response deserialization step

# This is a bytes object

result = response["Body"].read()

# Decoding bytes to a string

result = result.decode("utf-8")

# Converting to list of predictions

result = re.split(",|\n",result)

prediction\_df = pd.DataFrame()

prediction\_df["Prediction"] = result[:5]

prediction\_df["Label"] = test\_df["fraud"].iloc[:5].values

prediction\_df

To monitor the endpoint invocation metrics using Amazon CloudWatch, open the SageMaker console. Navigate to the Inference section and select Endpoints, then choose fraud-detect-xgb-endpoint.

On the Endpoint details page, under Monitor, select "View invocation metrics." At first, you might only notice a single dot on the metrics chart. However, after several invocations, a line will appear, resembling the one in the sample screenshot.

The Metrics page displays various endpoint performance metrics. You can select different time periods, such as 1 hour or 3 hours, to visualize the endpoint's performance. Click on any metric to view its trend over the selected time period. In the next step, you'll choose one of these metrics to define auto-scaling policies.

Since data capture was configured in the endpoint settings, you can review the payload sent to the endpoint along with its response. Note that it may take some time for the captured data to be fully uploaded to S3. Use the following code to verify if data capture is complete.

from sagemaker.s3 import S3Downloader

print("Waiting for captures to show up", end="")

for \_ in range(90):

capture\_files = sorted(S3Downloader.list(f"{data\_capture\_uri}/{endpoint\_name}"))

if capture\_files:

capture\_file = S3Downloader.read\_file(capture\_files[-1]).split("\n")

capture\_record = json.loads(capture\_file[0])

if "inferenceId" in capture\_record["eventMetadata"]:

break

print(".", end="", flush=True)

time.sleep(1)

print()

print(f"Found {len(capture\_files)} Data Capture Files:")

The captured data is stored in S3 as separate files for each endpoint invocation in JSON Lines format, which stores structured data with each line representing a JSON value. Use the following code to retrieve the data capture files.

capture\_files = sorted(S3Downloader.list(f"{data\_capture\_uri}/{endpoint\_name}"))

capture\_file = S3Downloader.read\_file(capture\_files[0]).split("\n")

capture\_record = json.loads(capture\_file[0])

capture\_record

Copy and paste the following code to decode the data in the captured files using base64. This code retrieves the five test samples that were sent as payload, along with their predictions. This feature is useful for inspecting endpoint loads with model responses and monitoring model performance.

input\_data = capture\_record["captureData"]["endpointInput"]["data"]

output\_data = capture\_record["captureData"]["endpointOutput"]["data"]

input\_data\_list = base64.b64decode(input\_data).decode("utf-8").split("\n")

print(input\_data\_list)

output\_data\_list = base64.b64decode(output\_data).decode("utf-8").split("\n")

print(output\_data\_list)

### Configure auto scaling for endpoint

Workloads using Real-Time Inference endpoints typically require low latency. During traffic spikes, these endpoints can face CPU overload, high latency, or timeouts. Therefore, it's crucial to scale capacity to manage traffic changes efficiently while maintaining low latency. SageMaker inference auto scaling monitors workloads and dynamically adjusts the instance count to ensure steady and predictable endpoint performance at a low cost. When the workload increases, auto scaling brings more instances online, and when the workload decreases, it removes unnecessary instances, helping to reduce compute costs. In this tutorial, you'll use the AWS SDK - Boto3 to set up auto scaling for your endpoint. SageMaker offers several types of auto scaling: target tracking scaling, step scaling, on-demand scaling, and scheduled scaling. This tutorial focuses on using a target tracking scaling policy, which is triggered when a chosen scaling metric exceeds a specified target threshold.

Auto scaling can be set up in two steps. First, configure a scaling policy with the details for the minimum, desired, and maximum number of instances per endpoint. Copy and paste the following code to configure a target tracking scaling policy. The specified maximum number of instances will be launched when traffic exceeds the chosen thresholds, which you will set in the next step.

resp = sm\_client.describe\_endpoint(EndpointName=endpoint\_name)

# SageMaker expects resource id to be provided with the following structure

resource\_id = f"endpoint/{endpoint\_name}/variant/{resp['ProductionVariants'][0]['VariantName']}"

# Scaling configuration

scaling\_config\_response = sm\_autoscaling\_client.register\_scalable\_target(

ServiceNamespace="sagemaker",

ResourceId=resource\_id,

ScalableDimension="sagemaker:variant:DesiredInstanceCount",

MinCapacity=1,

MaxCapacity=2

)

Copy and paste the following code to create the scaling policy. The selected scaling metric is SageMakerVariantInvocationsPerInstance, which measures the average number of invocations per minute for each inference instance of a model variant. Auto scaling is triggered when this metric exceeds the specified threshold of 5.

# Create Scaling Policy

policy\_name = f"scaling-policy-{endpoint\_name}"

scaling\_policy\_response = sm\_autoscaling\_client.put\_scaling\_policy(

PolicyName=policy\_name,

ServiceNamespace="sagemaker",

ResourceId=resource\_id,

ScalableDimension="sagemaker:variant:DesiredInstanceCount",

PolicyType="TargetTrackingScaling",

TargetTrackingScalingPolicyConfiguration={

"TargetValue": 5.0, # Target for avg invocations per minutes

"PredefinedMetricSpecification": {

"PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",

},

"ScaleInCooldown": 600, # Duration in seconds until scale in

"ScaleOutCooldown": 60 # Duration in seconds between scale out

}

)

Copy and paste the following code to retrieve the details of the scaling policy.

response = sm\_autoscaling\_client.describe\_scaling\_policies(ServiceNamespace="sagemaker")

pp = pprint.PrettyPrinter(indent=4, depth=4)

for i in response["ScalingPolicies"]:

pp.pprint(i["PolicyName"])

print("")

if("TargetTrackingScalingPolicyConfiguration" in i):

pp.pprint(i["TargetTrackingScalingPolicyConfiguration"])

Copy and paste the following code to stress-test the endpoint. This code will run for 250 seconds, repeatedly invoking the endpoint by sending randomly selected samples from the test dataset.

request\_duration = 250

end\_time = time.time() + request\_duration

print(f"Endpoint will be tested for {request\_duration} seconds")

while time.time() < end\_time:

csv\_file = io.StringIO()

test\_sample = test\_df.drop(["fraud"], axis=1).iloc[[np.random.randint(0, test\_df.shape[0])]]

test\_sample.to\_csv(csv\_file, sep=",", header=False, index=False)

payload = csv\_file.getvalue()

response = sm\_runtime\_client.invoke\_endpoint(

EndpointName=endpoint\_name,

Body=payload,

ContentType="text/csv"

)

You can monitor the endpoint metrics using Amazon CloudWatch. For a comprehensive list of available endpoint metrics, including invocations, refer to SageMaker's Endpoint Invocation Metrics. In the SageMaker console, navigate to Inference, select Endpoints, and choose the specific endpoint, such as fraud-detect-xgb-endpoint. Within the Endpoint details page, go to the Monitor section and select View invocation metrics. Then, on the Metrics page, choose InvocationsPerInstance (the monitoring metric selected during the scaling policy setup) and Invocations from the metrics list. Finally, navigate to the Graphed metrics tab to view the graphical representation of these metrics.

When viewing the Graphed metrics page, you'll have a visual representation of the traffic pattern received by the endpoint. You can modify the time granularity, for instance, changing it from the default 5 minutes to 1 minute. Auto scaling may take a few minutes to add the second instance. Once the new instance is incorporated, you'll notice that the invocations per instance amount to half of the total invocations.

Once the endpoint experiences an uptick in payload, you can verify the endpoint's status using the following code. This script monitors the transition of the endpoint's status from InService to Updating and tracks the instance counts. After a brief interval, you'll observe the status shifting from InService to Updating and then back to InService, albeit with an increased instance count.

# Check the instance counts after the endpoint gets more load

response = sm\_client.describe\_endpoint(EndpointName=endpoint\_name)

endpoint\_status = response["EndpointStatus"]

request\_duration = 250

end\_time = time.time() + request\_duration

print(f"Waiting for Instance count increase for a max of {request\_duration} seconds. Please re run this cell in case the count does not change")

while time.time() < end\_time:

response = sm\_client.describe\_endpoint(EndpointName=endpoint\_name)

endpoint\_status = response["EndpointStatus"]

instance\_count = response["ProductionVariants"][0]["CurrentInstanceCount"]

print(f"Status: {endpoint\_status}")

print(f"Current Instance count: {instance\_count}")

if (endpoint\_status=="InService") and (instance\_count>1):

break

else:

time.sleep(15)

### Clean up the resources

To avoid incurring unintended charges, it is best practice to delete any resources you are no longer using.

To prevent ongoing charges for the compute instance running at the endpoint, delete the model, endpoint configuration, and endpoint you created in this tutorial by running the following code block in your notebook.

# Delete model

sm\_client.delete\_model(ModelName=model\_name)

# Delete endpoint configuration

sm\_client.delete\_endpoint\_config(EndpointConfigName=endpoint\_config\_name)

# Delete endpoint

sm\_client.delete\_endpoint(EndpointName=endpoint\_name)

To delete the S3 bucket, follow these steps:

1. Open the Amazon S3 console.
2. In the navigation bar, select Buckets, then choose sagemaker-<your-Region>-<your-account-id>.
3. Select the checkbox next to fraud-detect-demo and click Delete.
4. In the Delete objects dialog box, confirm you have selected the correct object, then type permanently delete in the Permanently delete objects confirmation box.
5. Once the bucket is empty, you can delete the sagemaker-<your-Region>-<your-account-id> bucket by repeating the same procedure.

The Data Science kernel used to run the notebook image in this tutorial will continue to incur charges until you stop the kernel or delete the apps. For more information, refer to the "Shut Down Resources" section in the Amazon SageMaker Developer Guide.

To delete the SageMaker Studio apps, follow these steps:

1. Open the SageMaker Studio console.
2. Select studio-user.
3. Under Apps, choose Delete app for each listed app.
4. Wait for the Status to change to Deleted.