SageMaker shadow testing overview

This notebook's CI test result for us-west-2 is as follows. CI test results in other regions can be found at the end of the notebook.

This us-west-2 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

---

Amazon SageMaker now enables you to evaluate any changes to your model serving infrastructure, consisting of the ML model, the serving container, or the ML instance by shadow testing its performance against the currently deployed one. Shadow testing can help you catch potential configuration errors and performance issues before they impact end users. With SageMaker, you don't need to invest in building your own shadow testing infrastructure, allowing you to focus on model development.

You can use this to validate changes to any component to your production variant, namely the model, the container, or the instance, without any end user impact. It is useful in situations such as:

- You are considering promoting a new model that has been validated offline to production but want to evaluate operational performance metrics such as latency, error rate before making this decision
- You are considering changes to your serving infrastructure container, such as patching vulnerabilities or upgrading to newer versions, and want to assess the impact of these changes prior to promotion
- You are considering changing your ML instance and want to evaluate how the new instance would perform with live inference requests.

Just select a production variant you want to test against, and SageMaker automatically deploys the new variant in shadow mode and routes a copy of the inference requests to it in real time within the same endpoint. Only the responses of the production variant are returned to the calling application. You can choose to discard or log the responses of the shadow variant for offline comparison.

This notebook provides a walkthrough of the feature using the SageMaker Inference APIs.

SageMaker Background

title

A *production variant* consists of the ML model, Serving Container, and ML Instance. Since each variant is independent of others, you can have different models, containers, or instance types across variants. SageMaker lets you specify autoscaling policies on a per-variant basis so they can scale independently based on incoming load. SageMaker supports up to 10 production variants per endpoint. You can either configure a variant to receive a portion of the incoming

traffic by setting variant weights or specify the target variant in the incoming request. The response from the production variant is forwarded back to the invoker.

A *shadow variant (new)* has the same components as a production production variant. A user specified portion of the requests, known as the traffic sampling percentage (VariantWeight parameter in the ShadowProductionVariants object), is forwarded to the shadow variant. You can choose to log the response of the shadow variant in S3 or discard it. For an endpoint with a shadow variant, you can have a maximum of one production variant.

You can monitor the invocation metrics for both production and shadow variants in CloudWatch under the AWS/SageMaker namespace

Setup

Ensure that you have an updated version of boto3, which includes the latest SageMaker features:

```
!pip install sagemaker --quiet
!pip install -U awscli --quiet
```

The SageMaker role arn used to give training and hosting access to your data. The S3 bucket that you want to use for training and storing model objects.

```
import os
import boto3
import sagemaker
import time
from time import gmtime, strftime
from datetime import datetime, timedelta, timezone
from sagemaker import get_execution_role, session
from sagemaker.s3 import S3Downloader, S3Uploader

boto_session = boto3.session.Session()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name
sm_session = session.Session(boto3.Session())
sm = boto3.Session().client("sagemaker")
sm_runtime = boto3.Session().client("sagemaker-runtime")


# You can use a different bucket, but make sure the role you chose for
this notebook
# has the s3:PutObject permissions. This is the bucket into which the
model artifacts will be uploaded
bucket = "final-10lab"

prefix = "sagemaker/shadow-deployment"
resource_name = "ShadowDemo-{}-{}"
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/
pandas/core/computation/expressions.py:21: UserWarning: Pandas
```

```
requires version '2.8.0' or newer of 'numexpr' (version '2.7.3'
currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED

sagemaker.config INFO - Not applying SDK defaults from location:
/etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/home/ec2-user/.config/sagemaker/config.yaml
```

```
! mkdir model
! mkdir test_data

!aws s3 cp s3://final-10lab/model.tar.gz model/
!aws s3 cp s3://final-10lab/model2.tar.gz model/
```

```
mkdir: cannot create directory ‘model’: File exists
mkdir: cannot create directory ‘test_data’: File exists
download: s3://final-10lab/model.tar.gz to model/model.tar.gz
download: s3://final-10lab/model2.tar.gz to model/model2.tar.gz
```

Create models

First, we upload our pre-trained models to Amazon S3

This code uploads two pre-trained XGBoost models that are ready for you to deploy. These models were trained using the XGB Churn Prediction Notebook in SageMaker. You can also use your own pre-trained models in this step.

The models in this example are used to predict the probability of a mobile customer leaving their current mobile operator. The dataset we use is publicly available and was mentioned in the book Discovering Knowledge in Data by Daniel T. Larose. It is attributed by the author to the University of California Irvine Repository of Machine Learning Datasets.

To begin, let us upload these trained models to S3. Keep in mind that to use your pre-trained model, you just need to point `local_path` to your local pre-tained model file.

```
model_url = S3Uploader.upload(
    local_path="model/model.tar.gz",
    desired_s3_uri=f"s3://{bucket}/{prefix}",
)
model_url2 = S3Uploader.upload(
```

```
    local_path="model/model2.tar.gz",
    desired_s3_uri=f"s3://{bucket}/{prefix}",
)

print(f"Model URI 1: {model_url}")
print(f"Model URI 2: {model_url2}")
```

```
sagemaker.config INFO - Not applying SDK defaults from location:
/etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/home/ec2-user/.config/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location:
/home/ec2-user/.config/sagemaker/config.yaml
Model URI 1: s3://final-10lab/sagemaker/shadow-deployment/model.tar.gz
Model URI 2:
s3://final-10lab/sagemaker/shadow-deployment/model2.tar.gz
```

```
from sagemaker import image_uris

image_uri = image_uris.retrieve("xgboost",
boto3.Session().region_name, "0.90-1")
image_uri2 = image_uris.retrieve("xgboost",
boto3.Session().region_name, "0.90-2")

print(f"Model Image 1: {image_uri}")
print(f"Model Image 2: {image_uri2}")
```

```
Model Image 1: 683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-
xgboost:0.90-1-cpu-py3
Model Image 2: 683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-
xgboost:0.90-2-cpu-py3
```

Deploy the two models as production and shadow variants to a real-time Inference endpoint

The first step in deploying a trained model to SageMaker Inference is to created a SageMaker Model using the create_model API.

```
model_name = f"DEMO-xgb-stroke-pred-{datetime.now():%Y-%m-%d-%H-%M-
%S}"
model_name2 = f"DEMO-xgb-stroke-pred-{datetime.now():%Y-%m-%d-%H-%M-
%S}"

print(f"Model Name 1: {model_name}")
print(f"Model Name 2: {model_name2}")

resp = sm.create_model(
    ModelName=model_name,
```

```
    ExecutionRoleArn=role,
    Containers=[{"Image": image_uri, "ModelDataUrl": model_url}],
)
print(f"Created Model: {resp}")

#resp = sm.create_model(
 #   ModelName=model_name2,
#    ExecutionRoleArn=role,
 #   Containers=[{"Image": image_uri2, "ModelDataUrl": model_url2}],
#)
#print(f"Created Model: {resp}")
```

```
Model Name 1: DEMO-xgb-stroke-pred-2023-12-13-03-33-31
Model Name 2: DEMO-xgb-stroke-pred-2023-12-13-03-33-31
Created Model: {'ModelArn': 'arn:aws:sagemaker:us-east-
1:040700907151:model/demo-xgb-stroke-pred-2023-12-13-03-33-31',
'ResponseMetadata': {'RequestId': 'ee60b44c-a160-49b2-a4b6-
b744a19d8d76', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-
requestid': 'ee60b44c-a160-49b2-a4b6-b744a19d8d76', 'content-type':
'application/x-amz-json-1.1', 'content-length': '102', 'date': 'Wed,
13 Dec 2023 03:33:31 GMT'}, 'RetryAttempts': 0}}
```

The first step in deploying a trained model to SageMaker Inference is to created a SageMaker Model using the create_model API

The next step is to create an endpoint config with the production and shadow variants. The ProductionVariants and the ShadowProductionVariants are of particular interest. We set the InitialVariantWeight in the ShadowProductionVariants to sample and send 50% of the production variant requests to the shadow variant. The production variant receives 100% of the traffic.

Both these variants have ml.m5.xlarge instances with 4 vCPUs and 16 GiB of memory and the initial instance count is set to 1.

```
ep_config_name = f"Shadow-EpConfig-{datetime.now():%Y-%m-%d-%H-%M-%S}"
production_variant_name = "production"
shadow_variant_name = "shadow"

create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName=ep_config_name,
    ProductionVariants=[
        {
            "VariantName": production_variant_name,
            "ModelName": model_name,
            "InstanceType": "ml.m5.xlarge",
            "InitialInstanceCount": 2,
            "InitialVariantWeight": 1,
        }
    ],
    ShadowProductionVariants=[
```

```
        {
            "VariantName": shadow_variant_name,
            "ModelName": model_name2,
            "InstanceType": "ml.m5.xlarge",
            "InitialInstanceCount": 1,
            "InitialVariantWeight": 0.5,
        }
    ],
)
print(f"Created EndpointConfig:
{create_endpoint_config_response['EndpointConfigArn']}")
```

```
Created EndpointConfig: arn:aws:sagemaker:us-east-
1:040700907151:endpoint-config/shadow-epconfig-2023-12-13-03-33-38
```

```
endpoint_name = f"xgb-prod-shadow-{datetime.now():%Y-%m-%d-%H-%M-%S}"
create_endpoint_api_response = sm.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=ep_config_name,
)
```

Now, wait for the endpoint creation to complete. This should take 2-5 minutes, depending on your model artifact and serving container size.

```
def wait_for_endpoint_in_service(endpoint_name):
    print("Waiting for endpoint in service")
    while True:
        details = sm.describe_endpoint(EndpointName=endpoint_name)
        status = details["EndpointStatus"]
        if status in ["InService", "Failed"]:
            print("\nDone!")
            break
        print(".", end="", flush=True)
        time.sleep(30)


wait_for_endpoint_in_service(endpoint_name)

sm.describe_endpoint(EndpointName=endpoint_name)
```

```
Waiting for endpoint in service
.....
Done!

{'EndpointName': 'xgb-prod-shadow-2023-12-13-03-34-11',
 'EndpointArn':
'arn:aws:sagemaker:us-east-1:040700907151:endpoint/xgb-prod-shadow-
2023-12-13-03-34-11',
 'EndpointConfigName': 'Shadow-EpConfig-2023-12-13-03-33-38',
 'ProductionVariants': [{'VariantName': 'production',
```

```
   'DeployedImages': [{'SpecifiedImage': '683313688378.dkr.ecr.us-
east-1.amazonaws.com/sagemaker-xgboost:0.90-1-cpu-py3',
     'ResolvedImage':
'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-
xgboost@sha256:4814427c3e0a6cf99e637704da3ada04219ac7cd5727ff622841537
61d36d7d3',
     'ResolutionTime': datetime.datetime(2023, 12, 13, 3, 34, 12,
888000, tzinfo=tzlocal())}],
   'CurrentWeight': 1.0,
   'DesiredWeight': 1.0,
   'CurrentInstanceCount': 2,
   'DesiredInstanceCount': 2}],
 'EndpointStatus': 'InService',
 'CreationTime': datetime.datetime(2023, 12, 13, 3, 34, 12, 28000,
tzinfo=tzlocal()),
 'LastModifiedTime': datetime.datetime(2023, 12, 13, 3, 36, 37,
101000, tzinfo=tzlocal()),
 'ShadowProductionVariants': [{'VariantName': 'shadow',
   'DeployedImages': [{'SpecifiedImage': '683313688378.dkr.ecr.us-
east-1.amazonaws.com/sagemaker-xgboost:0.90-1-cpu-py3',
     'ResolvedImage':
'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-
xgboost@sha256:4814427c3e0a6cf99e637704da3ada04219ac7cd5727ff622841537
61d36d7d3',
     'ResolutionTime': datetime.datetime(2023, 12, 13, 3, 34, 12,
959000, tzinfo=tzlocal())}],
   'CurrentWeight': 0.5,
   'DesiredWeight': 0.5,
   'CurrentInstanceCount': 1,
   'DesiredInstanceCount': 1}],
 'ResponseMetadata': {'RequestId': '26b8ffb0-1d39-4141-b63a-
53f4f83fbbdd',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '26b8ffb0-1d39-4141-b63a-
53f4f83fbbdd',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '1208',
   'date': 'Wed, 13 Dec 2023 03:36:48 GMT'},
  'RetryAttempts': 0}}
```

Invoke Endpoint

Once the endpoint has been successfully created, you can begin invoking it. To learn more about endpoint, please check out our documentation.

```
def invoke_endpoint(endpoint_name, should_raise_exp=False):
    with open("test_data/test2.csv", "r") as f:
        for row in f:
            payload = row.rstrip("\n")
```

```
            try:
                for i in range(10):  # send the same payload 10 times
for testing purpose
                    response = sm_runtime.invoke_endpoint(
                        EndpointName=endpoint_name,
ContentType="text/csv", Body=payload
                    )
            except Exception as e:
                print("E", end="", flush=True)
                if should_raise_exp:
                    raise e


invoke_endpoint(endpoint_name)
```

Now that the endpoint is InService and has been invoked, the following cells help collect CloudWatch metrics between the production and shadow variants for metrics comparison.

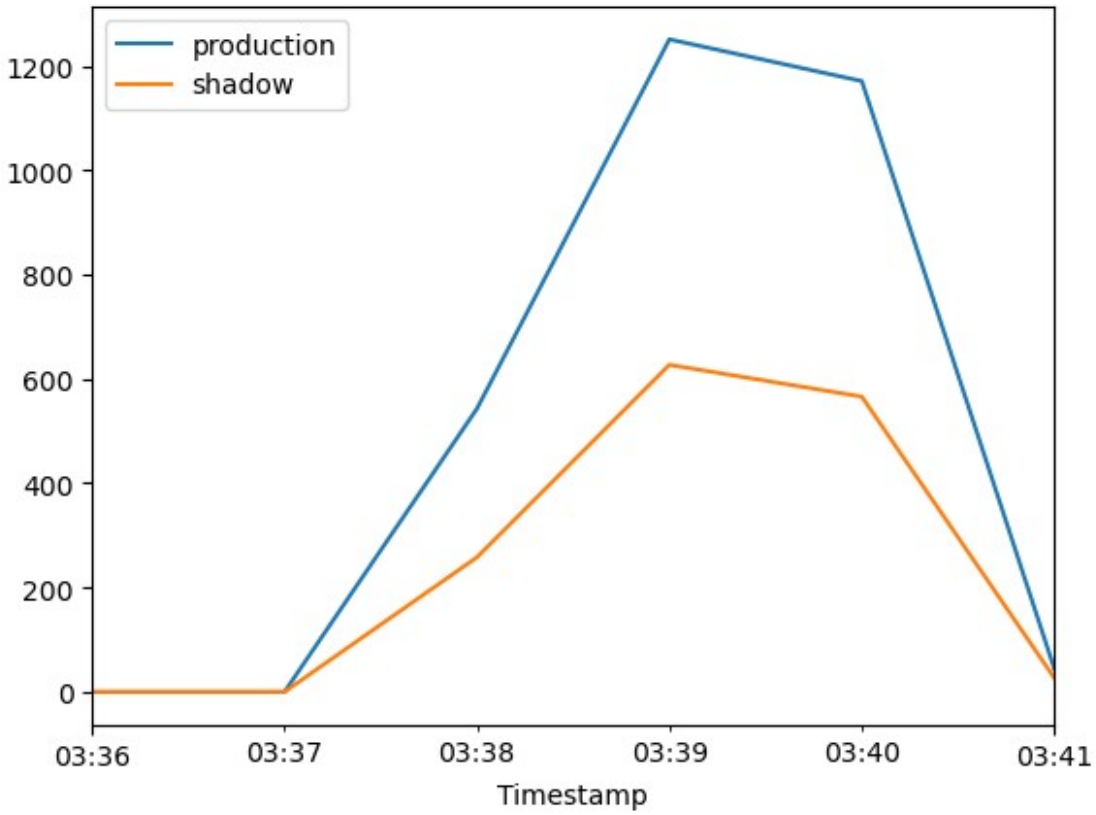
```
%matplotlib inline

import pandas as pd

cw = boto3.Session().client("cloudwatch", region_name=region)


def get_sagemaker_metrics(
    endpoint_name,
    variant_name,
    metric_name,
    statistic,
    start_time,
    end_time,
):
    dimensions = [
        {"Name": "EndpointName", "Value": endpoint_name},
        {"Name": "VariantName", "Value": variant_name},
    ]
    namespace = "AWS/SageMaker"
    if metric_name in ["CPUUtilization", "MemoryUtilization",
"DiskUtilization"]:
        namespace = "/aws/sagemaker/Endpoints"

    metrics = cw.get_metric_statistics(
        Namespace=namespace,
        MetricName=metric_name,
        StartTime=start_time,
        EndTime=end_time,
        Period=1,
        Statistics=[statistic],
        Dimensions=dimensions,
```

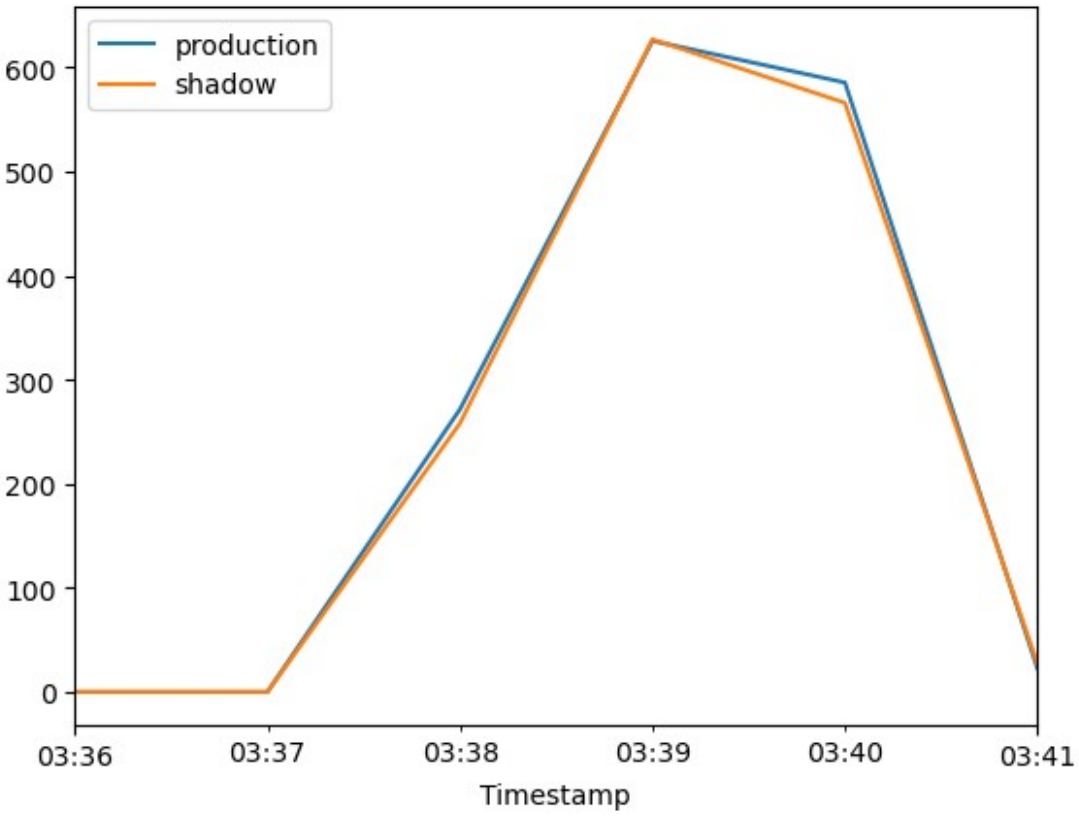
```
    )

    if len(metrics["Datapoints"]) == 0:
        return
    return (
        pd.DataFrame(metrics["Datapoints"])
        .sort_values("Timestamp")
        .set_index("Timestamp")
        .drop(["Unit"], axis=1)
        .rename(columns={statistic: variant_name})
    )


def plot_endpoint_invocation_metrics(
    endpoint_name,
    metric_name,
    statistic,
    start_time=None,
):
    start_time = start_time or datetime.now(timezone.utc) -
timedelta(minutes=10)
    end_time = datetime.now(timezone.utc)
    metrics_production = get_sagemaker_metrics(
        endpoint_name,
        production_variant_name,
        metric_name,
        statistic,
        start_time,
        end_time,
    )
    metrics_shadow = get_sagemaker_metrics(
        endpoint_name,
        shadow_variant_name,
        metric_name,
        statistic,
        start_time,
        end_time,
    )
    try:
        metrics_variants = pd.merge(metrics_production,
metrics_shadow, on="Timestamp")
        return metrics_variants.plot(y=["production", "shadow"])
    except Exception as e:
        print(e)
```

Metric Comparison

Now that we have deployed both the production and shadow models, let us compare the invocation metrics. Here is a list of invocation metrics available for comparison. Let us start by comparing invocations between the production and shadow variants

```
invocations = plot_endpoint_invocation_metrics(endpoint_name,
"Invocations", "Sum")
invocations_per_instance = plot_endpoint_invocation_metrics(
    endpoint_name, "InvocationsPerInstance", "Sum"
)
```

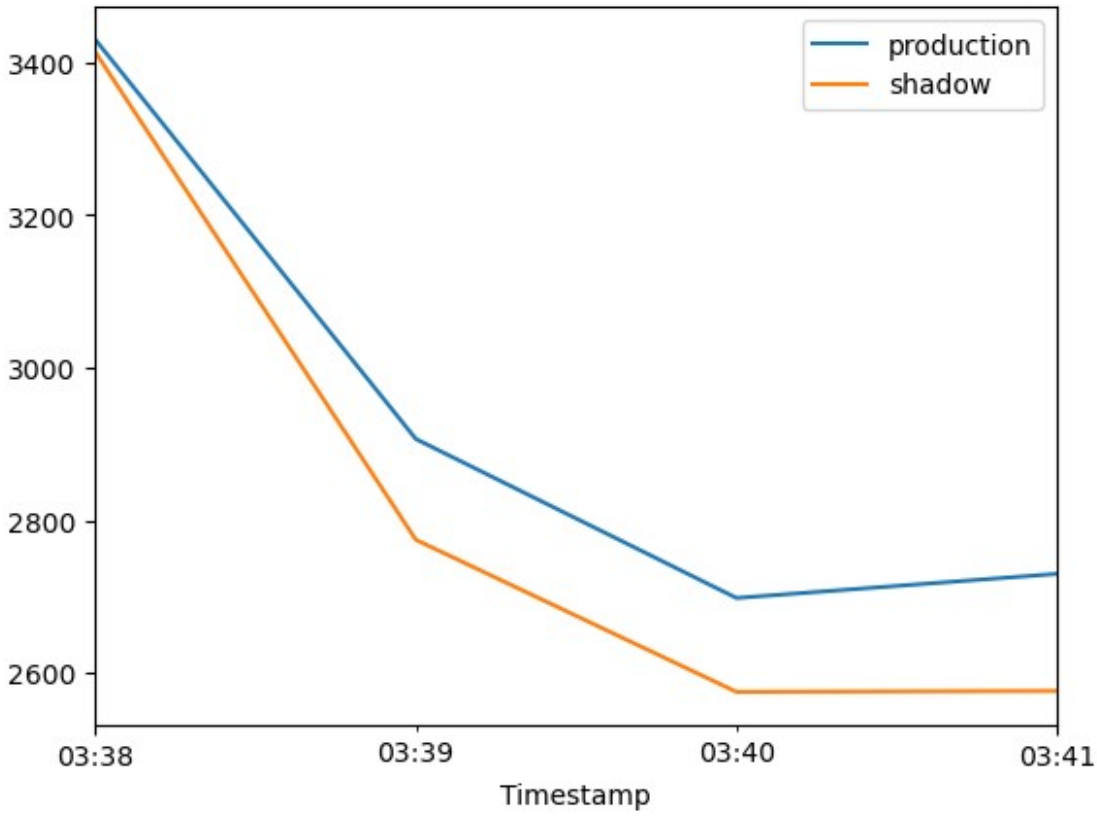




The Invocation metric refers to the number of invocations sent to the production variant. A fraction of these invocations, specified in the variant weight, are sent to the shadow variant. The invocation per instance is calculated by dividing the total number of invocations by the number of instances in a variant. From the chart above, we can confirm that both the production and shadow variants are receiving invocation requests according to the weights specified in the endpoint config.

Next let us compare the model latency between the production and shadow variants. Model latency is the time taken by a model to respond as viewed from SageMaker.

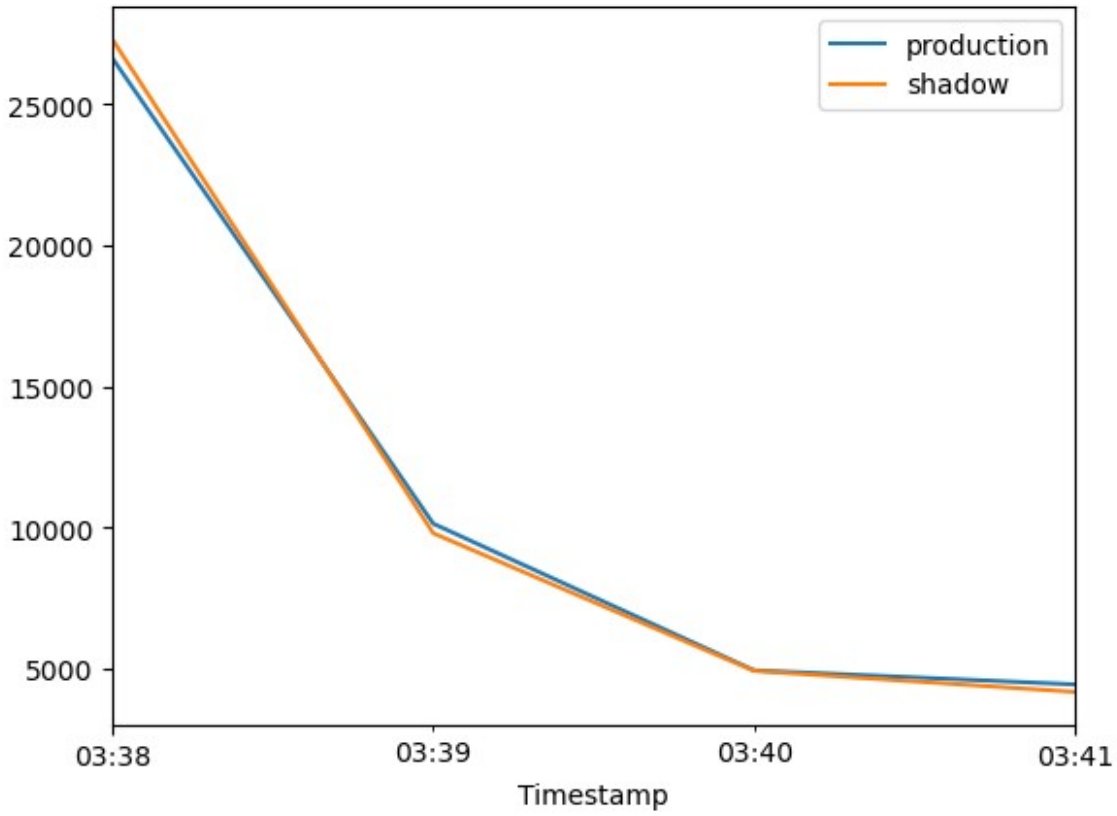
```
model_latency = plot_endpoint_invocation_metrics(endpoint_name,
"ModelLatency", "Average")
```



Using the chart above, we can observe how the model latency of the shadow variant compares with the production variant without exposing end users to the shadow variant.

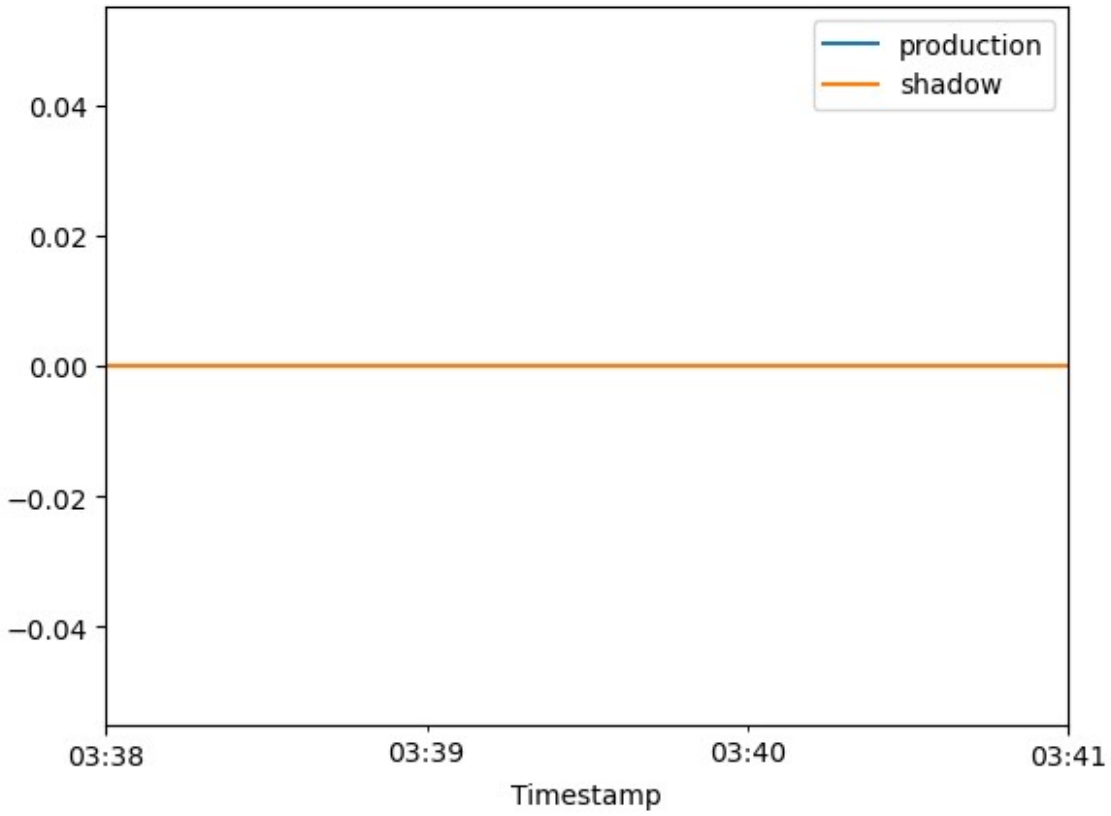
We expect the overhead latency to be comparable across production and shadow variants. Overhead latency is the interval measured from the time SageMaker receives the request until it returns a response to the client, minus the model Latency.

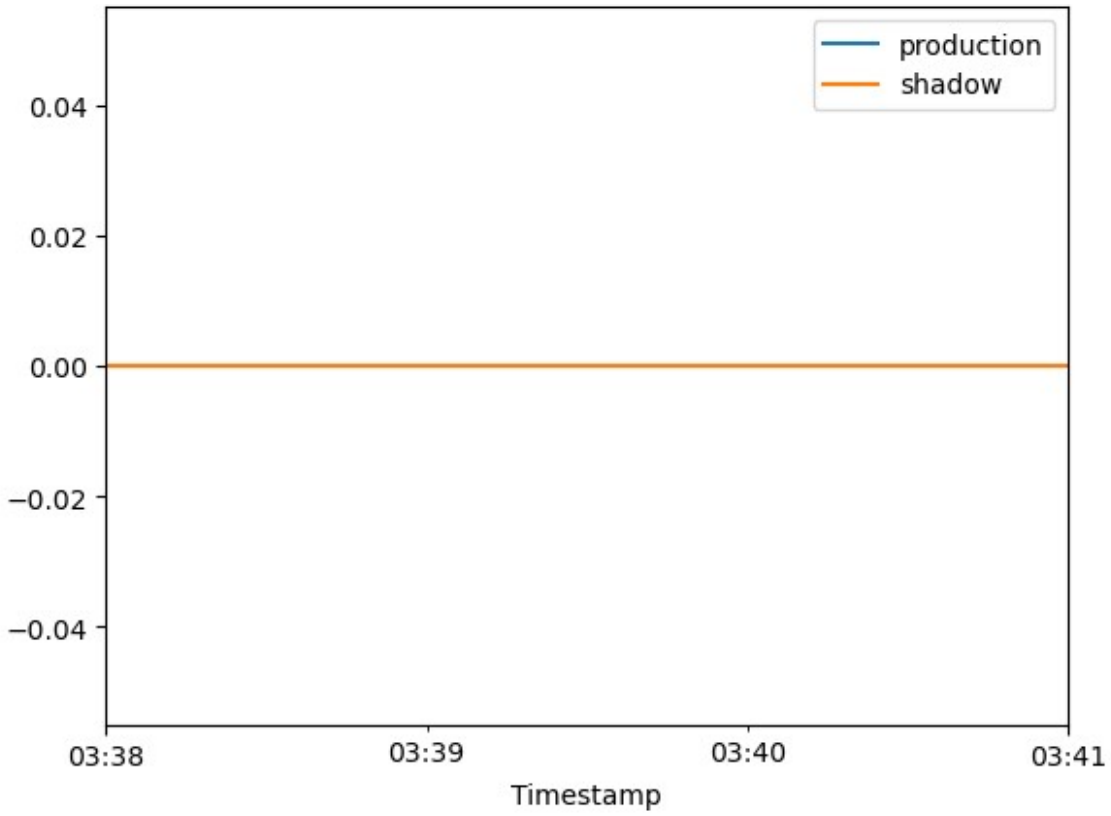
```
overhead_latency = plot_endpoint_invocation_metrics(endpoint_name,
"OverheadLatency", "Average")
```

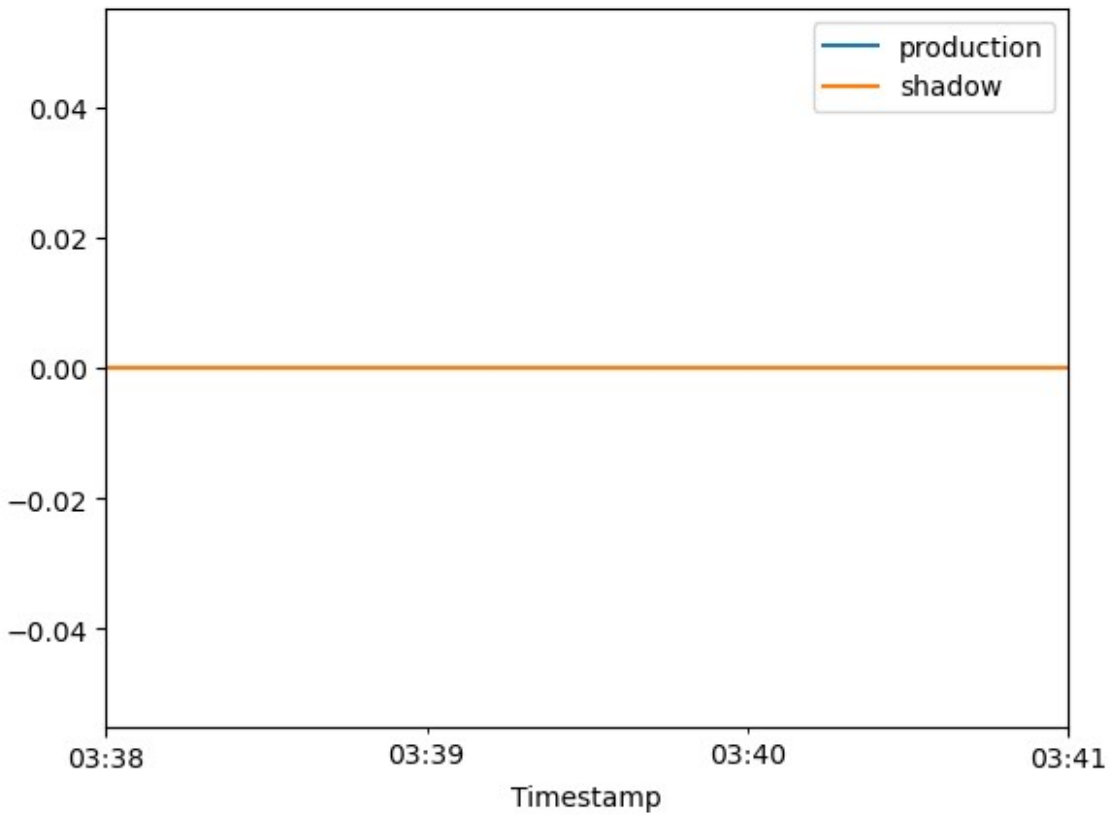


Finally, let us review the 4xx, 5xx and total model errors returned by the model serving container.

```
Invocation4xxErrors = plot_endpoint_invocation_metrics(endpoint_name,
"Invocation4XXErrors", "Sum")
Invocation5xxErrors = plot_endpoint_invocation_metrics(endpoint_name,
"Invocation5XXErrors", "Sum")
Invocation5xxErrors = plot_endpoint_invocation_metrics(
    endpoint_name, "InvocationModelErrors", "Sum"
)
```


production
shadow
0.04
0.02
0.00
−0.02
−0.04
03:38
03:39
03:40
03:41
Timestamp


production
shadow
0.04
0.02
0.00
−0.02
−0.04
03:38
03:39
03:40
03:41
Timestamp



We can consider promoting the shadow model if we do not see any differences in 4xx and 5xx errors between the production shadow variants.

To promote the shadow model to production, create a new endpoint configuration with current ShadowProductionVariant as the new ProductionVariant and removing the ShadowProductionVariant. This will remove the current ProductionVariant and promote the shadow variant to become the new production variant. As always, all SageMaker updates are orchestrated as blue/green deployments under the hood and there is no loss of availability while performing the update. Optionally, you can leverage Deployment Guardrails if you want to use all-at-once traffic shifting and auto rollbacks during your update.

```
promote_ep_config_name = f"PromoteShadow-EpConfig-{datetime.now():%Y-
%m-%d-%H-%M-%S}"

create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName=promote_ep_config_name,
    ProductionVariants=[
        {
            "VariantName": shadow_variant_name,
            "ModelName": model_name2,
            "InstanceType": "ml.m5.xlarge",
            "InitialInstanceCount": 2,
            "InitialVariantWeight": 1.0,
        }
```

```
    ],
)
print(f"Created EndpointConfig:
{create_endpoint_config_response['EndpointConfigArn']}")
```

```
Created EndpointConfig: arn:aws:sagemaker:us-east-
1:040700907151:endpoint-config/promoteshadow-epconfig-2023-12-13-03-
42-10
```

```
update_endpoint_api_response = sm.update_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=promote_ep_config_name,
)

wait_for_endpoint_in_service(endpoint_name)

sm.describe_endpoint(EndpointName=endpoint_name)
```

```
Waiting for endpoint in service
.....
Done!

{'EndpointName': 'xgb-prod-shadow-2023-12-13-03-34-11',
 'EndpointArn':
'arn:aws:sagemaker:us-east-1:040700907151:endpoint/xgb-prod-shadow-
2023-12-13-03-34-11',
 'EndpointConfigName': 'PromoteShadow-EpConfig-2023-12-13-03-42-10',
 'ProductionVariants': [{'VariantName': 'shadow',
   'DeployedImages': [{'SpecifiedImage': '683313688378.dkr.ecr.us-
east-1.amazonaws.com/sagemaker-xgboost:0.90-1-cpu-py3',
     'ResolvedImage':
'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-
xgboost@sha256:4814427c3e0a6cf99e637704da3ada04219ac7cd5727ff622841537
61d36d7d3',
     'ResolutionTime': datetime.datetime(2023, 12, 13, 3, 42, 20,
984000, tzinfo=tzlocal())}],
   'CurrentWeight': 1.0,
   'DesiredWeight': 1.0,
   'CurrentInstanceCount': 2,
   'DesiredInstanceCount': 2}],
 'EndpointStatus': 'InService',
 'CreationTime': datetime.datetime(2023, 12, 13, 3, 34, 12, 28000,
tzinfo=tzlocal()),
 'LastModifiedTime': datetime.datetime(2023, 12, 13, 3, 44, 39,
578000, tzinfo=tzlocal()),
 'ResponseMetadata': {'RequestId': 'b56f6fd3-9e91-4508-a334-
bb7c1459a119',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'b56f6fd3-9e91-4508-a334-
bb7c1459a119',
```

```
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '762',
   'date': 'Wed, 13 Dec 2023 03:44:51 GMT'},
  'RetryAttempts': 0}}
```

If you do not want to create multiple endpoint configurations and want SageMaker to manage the end to end workflow of creating, managing, and acting on the results of the shadow tests, consider using the SageMaker Inference Experiement APIs/Console experience. As stated earlier, they enable you to setup shadow tests for a predefined duration of time, monitor the progress through a live dashboard, presents clean up options upon completion, and act on the results. To get started, please navigate to the 'Shadow Tests' section of the SageMaker Inference console.

Cleanup

If you do not plan to use this endpoint further, you should delete the endpoint to avoid incurring additional charges and clean up other resources created in this notebook.

```
sm.delete_endpoint(EndpointName=endpoint_name)
sm.delete_endpoint_config(EndpointConfigName=ep_config_name)
sm.delete_endpoint_config(EndpointConfigName=promote_ep_config_name)
sm.delete_model(ModelName=model_name)
sm.delete_model(ModelName=model_name2)
```

Notebook CI Test Results

This notebook was tested in multiple regions. The test results are as follows, except for us-west-2 which is shown at the top of the notebook.

This us-east-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This us-east-2 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This us-west-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This ca-central-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This sa-east-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This eu-west-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This eu-west-2 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This eu-west-3 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This eu-central-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This eu-north-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This ap-southeast-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This ap-southeast-2 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This ap-northeast-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This ap-northeast-2 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable

This ap-south-1 badge failed to load. Check your device's internet connectivity, otherwise the service is currently unavailable