A/B testing, traffic shifting and autoscaling

Introduction

In this lab you will create an endpoint with multiple variants, splitting the traffic between them. Then after testing and reviewing the endpoint performance metrics, you will shift the traffic to one variant and configure it to autoscale.

Table of Contents

- 1. Create an endpoint with multiple variants
  - 1.1. Construct Docker Image URI
    - Exercise 1
  - 1.2. Create Amazon SageMaker Models
    - Exercise 2
    - Exercise 3
  - 1.3. Set up Amazon SageMaker production variants
    - Exercise 4
    - Exercise 5
  - 1.4. Configure and create endpoint
    - Exercise 6
- 2. Test model
  - 2.1. Test the model on a few sample strings
    - Exercise 7
  - 2.2. Generate traffic and review the endpoint performance metrics
- 3. Shift the traffic to one variant and review the endpoint performance metrics
  - Exercise 8
- 4. Configure one variant to autoscale

Let's install and import the required modules.

In [2]:

```
# please ignore warning messages during the installation
!pip install --disable-pip-version-check -q sagemaker==2.35.0
!conda install -q -y pytorch==1.6.0 -c pytorch
!pip install --disable-pip-version-check -q transformers==3.5.1
```

```
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommen
ded to use a virtual environment instead: https://pip.pypa.io/warnings/ve
nv
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retr
ying with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: /opt/conda

  added / updated specs:
    - pytorch==1.6.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2022.07.19 |       h06a4308_0         124 KB
    conda-4.14.0               |   py37h06a4308_0         909 KB
    cudatoolkit-10.2.89        |       hfd86e86_1       365.1 MB
    ninja-1.10.2               |       h06a4308_5           8 KB
    ninja-base-1.10.2          |       hd09550d_5         109 KB
    pytorch-1.6.0              |py3.7_cuda10.2.89_cudnn7.6.5_0       537.
7 MB  pytorch
    ------------------------------------------------------------
                                           Total:       903.9 MB

The following NEW packages will be INSTALLED:

  cudatoolkit        pkgs/main/linux-64::cudatoolkit-10.2.89-hfd86e86_1
  ninja              pkgs/main/linux-64::ninja-1.10.2-h06a4308_5
  ninja-base         pkgs/main/linux-64::ninja-base-1.10.2-hd09550d_5
  pytorch            pytorch/linux-64::pytorch-1.6.0-py3.7_cuda10.2.89_cu
dnn7.6.5_0

The following packages will be UPDATED:

  ca-certificates    conda-forge::ca-certificates-2022.6.1~ --> pkgs/main
::ca-certificates-2022.07.19-h06a4308_0

The following packages will be SUPERSEDED by a higher-priority channel:

  conda              conda-forge::conda-4.14.0-py37h89c186~ --> pkgs/main
::conda-4.14.0-py37h06a4308_0


Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Retrieving notices: ...working... done
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommen
ded to use a virtual environment instead: https://pip.pypa.io/warnings/ve
nv
```

In [3]:

```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

In [4]:

```
import boto3
import sagemaker
import pandas as pd
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c3/w2')

# low-level service client of the boto3 session
sm = boto3.client(service_name='sagemaker',
                  config=config)

sm_runtime = boto3.client('sagemaker-runtime',
                          config=config)

sess = sagemaker.Session(sagemaker_client=sm,
                         sagemaker_runtime_client=sm_runtime)

bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = sess.boto_region_name

cw = boto3.client(service_name='cloudwatch',
                  config=config)

autoscale = boto3.client(service_name="application-autoscaling",
                         config=config)
```

1. Create an endpoint with multiple variants

Two models trained to analyze customer feedback and classify the messages into positive (1), neutral (0), and negative (-1) sentiments are saved in the following S3 bucket paths. These `tar.gz` files contain the model artifacts, which result from model training.

In [5]:

```
model_a_s3_uri = 's3://dlai-practical-data-science/models/ab/variant_a/mo
model_b_s3_uri = 's3://dlai-practical-data-science/models/ab/variant_b/mo
```

Let's deploy an endpoint splitting the traffic between these two models 50/50 to perform A/B Testing. Instead of creating a PyTorch Model object and calling `model.deploy()` function, you will create an `Endpoint configuration` with multiple model variants. Here is the workflow you will follow to create an endpoint:

1.1. Construct Docker Image URI

You will need to create the models in Amazon SageMaker, which retrieves the URI for the pre-built SageMaker Docker image stored in Amazon Elastic Container Re gistry (ECR). Let's construct the ECR URI which you will pass into the `create_model` function later.

Set the instance type. For the purposes of this lab, you will use a relatively small instance. Please refer to this link for additional instance types that may work for your use cases outside of this lab.

In [6]:
```
inference_instance_type = 'ml.m5.large'
```

Exercise 1

Create an ECR URI using the `'PyTorch'` framework. Review other parameters of the image.

In [10]:
```
inference_image_uri = sagemaker.image_uris.retrieve(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    framework='pytorch', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    version='1.6.0',
    instance_type=inference_instance_type,
    region=region,
    py_version='py3',
    image_scope='inference'
)
print(inference_image_uri)
```

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.6.0-cpu-
py3
```

1.2. Create Amazon SageMaker Models

Amazon SageMaker Model includes information such as the S3 location of the model, the container image that can be used for inference with that model, the execution role, and the model name.

Let's construct the model names.

In [11]:
```python
import time
from pprint import pprint

timestamp = int(time.time())

model_name_a = '{}-{}'.format('a', timestamp)
model_name_b = '{}-{}'.format('b', timestamp)
```

You will use the following function to check if the model already exists in Amazon SageMaker.

In [12]:
```python
def check_model_existence(model_name):
    for model in sm.list_models()['Models']:
        if model_name == model['ModelName']:
            return True
    return False
```

Exercise 2

Create an Amazon SageMaker Model based on the `model_a_s3_uri` data.

Instructions: Use `sm.create_model` function, which requires the model name, Amazon SageMaker execution role and a primary container description (`PrimaryContainer` dictionary). The `PrimaryContainer` includes the S3 bucket location of the model artifacts (`ModelDataUrl` key) and ECR URI (`Image` key).

In [13]:
```python
if not check_model_existence(model_name_a):
    model_a = sm.create_model(
        ModelName=model_name_a,
        ExecutionRoleArn=role,
        PrimaryContainer={
            ### BEGIN SOLUTION - DO NOT delete this comment for grading p
            'ModelDataUrl': model_a_s3_uri, # Replace None
            'Image': inference_image_uri # Replace None
            ### END SOLUTION - DO NOT delete this comment for grading pur
        }
    )
    pprint(model_a)
else:
    print("Model {} already exists".format(model_name_a))
```


```
{'ModelArn': 'arn:aws:sagemaker:us-east-1:643403534509:model/a-1662363360
',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '74',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:36:58
GMT',
                                      'x-amzn-requestid': 'ef12a4b2-eba0-
43f6-a0c5-017fdc4ee35e'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'ef12a4b2-eba0-43f6-a0c5-017fdc4ee35e'
,
                      'RetryAttempts': 0}}
```

Exercise 3

Create an Amazon SageMaker Model based on the `model_b_s3_uri` data.

Instructions: Use the example in the cell above.

In [14]:

```
if not check_model_existence(model_name_b):
    model_b = sm.create_model(
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        ModelName=model_name_b, # Replace all None
        ExecutionRoleArn=role, # Replace all None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
        PrimaryContainer={
            'ModelDataUrl': model_b_s3_uri,
            'Image': inference_image_uri
        }
    )
    pprint(model_b)
else:
    print("Model {} already exists".format(model_name_b))
```


```
{'ModelArn': 'arn:aws:sagemaker:us-east-1:643403534509:model/b-1662363360
',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '74',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:37:40
GMT',
                                      'x-amzn-requestid': 'a43d35ae-16b2-
4393-abbd-2be881672bdd'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'a43d35ae-16b2-4393-abbd-2be881672bdd'
,
                      'RetryAttempts': 0}}
```


1.3. Set up Amazon SageMaker production variants

A production variant is a packaged SageMaker Model combined with the configuration related to how that model will be hosted.

You have constructed the model in the section above. The hosting resources configuration includes information on how you want that model to be hosted: the number and type of instances, a pointer to the SageMaker package model, as well as a variant name and variant weight. A single SageMaker Endpoint can actually include multiple production variants.

Exercise 4

Create an Amazon SageMaker production variant for the SageMaker Model with the `model_name_a`.

Instructions: Use the `production_variant` function passing the `model_name_a` and instance type defined above.

```
variantA = production_variant(
     model_name=..., # SageMaker Model name
     instance_type=..., # instance type
     initial_weight=50, # traffic distribution weight
     initial_instance_count=1, # instance count
     variant_name='VariantA', # production variant name
)
```

In [15]:
```
from sagemaker.session import production_variant

variantA = production_variant(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    model_name=model_name_a, # Replace None
    instance_type=inference_instance_type, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    initial_weight=50,
    initial_instance_count=1,
    variant_name='VariantA',
)
print(variantA)
```

```
{'ModelName': 'a-1662363360', 'InstanceType': 'ml.m5.large', 'InitialInst
anceCount': 1, 'VariantName': 'VariantA', 'InitialVariantWeight': 50}
```

Exercise 5

Create an Amazon SageMaker production variant for the SageMaker Model with the `model_name_b`.

Instructions: See the required arguments in the cell above.

In [16]:
```
variantB = production_variant(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    model_name=model_name_b, # Replace all None
    instance_type=inference_instance_type, # Replace all None
    initial_weight=50, # Replace all None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    initial_instance_count=1,
    variant_name='VariantB'
)
print(variantB)
```

```
{'ModelName': 'b-1662363360', 'InstanceType': 'ml.m5.large', 'InitialInst
anceCount': 1, 'VariantName': 'VariantB', 'InitialVariantWeight': 50}
```

1.4. Configure and create the endpoint

You will use the following functions to check if the endpoint configuration and endpoint itself already exist in Amazon SageMaker.

In [17]:
```
def check_endpoint_config_existence(endpoint_config_name):
    for endpoint_config in sm.list_endpoint_configs()['EndpointConfigs']:
        if endpoint_config_name == endpoint_config['EndpointConfigName']:
            return True
    return False

def check_endpoint_existence(endpoint_name):
    for endpoint in sm.list_endpoints()['Endpoints']:
        if endpoint_name == endpoint['EndpointName']:
            return True
    return False
```

Create the endpoint configuration by specifying the name and pointing to the two production variants that you just configured that tell SageMaker how you want to host those models.

In [18]:
```
endpoint_config_name = '{}-{}'.format('ab', timestamp)

if not check_endpoint_config_existence(endpoint_config_name):
    endpoint_config = sm.create_endpoint_config(
        EndpointConfigName=endpoint_config_name,
        ProductionVariants=[variantA, variantB]
    )
    pprint(endpoint_config)
else:
    print("Endpoint configuration {} already exists".format(endpoint_conf
```

```
{'EndpointConfigArn': 'arn:aws:sagemaker:us-east-1:643403534509:endpoint-
config/ab-1662363360',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '94',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:39:22
GMT',
                                      'x-amzn-requestid': '6ae567cb-4aec-
48b0-95b5-5989cd6ad0f2'},
                      'HTTPStatusCode': 200,
                      'RequestId': '6ae567cb-4aec-48b0-95b5-5989cd6ad0f2'
,
                      'RetryAttempts': 0}}
```

Construct the endpoint name.

In [19]:
```
model_ab_endpoint_name = '{}-{}'.format('ab', timestamp)
print('Endpoint name: {}'.format(model_ab_endpoint_name))
```

```
Endpoint name: ab-1662363360
```

Exercise 6

Create an endpoint with the endpoint name and configuration defined above.

In [20]:
```
if not check_endpoint_existence(model_ab_endpoint_name):
    endpoint_response = sm.create_endpoint(
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        EndpointName=model_ab_endpoint_name, # Replace None
        EndpointConfigName=endpoint_config_name # Replace None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
    )
    print('Creating endpoint {}'.format(model_ab_endpoint_name))
    pprint(endpoint_response)
else:
    print("Endpoint {} already exists".format(model_ab_endpoint_name))
```

```
Creating endpoint ab-1662363360
{'EndpointArn': 'arn:aws:sagemaker:us-east-1:643403534509:endpoint/ab-166
2363360',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '81',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:40:09
GMT',
                                      'x-amzn-requestid': 'b011779d-2e23-
40af-b9b0-19cf50b501cf'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'b011779d-2e23-40af-b9b0-19cf50b501cf'
,
                      'RetryAttempts': 0}}
```

Review the created endpoint configuration in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoint configuration
- check the name of the endpoint configuration, its Amazon Resource Name (ARN) and production variants
- click on the production variants and check their container information: image and model data location

In [21]:
```
from IPython.core.display import display, HTML

display(
    HTML(
        '<b>Review <a target="blank" href="https://console.aws.amazon.com
            region, endpoint_config_name
        )
    )
)
```

Review **REST Endpoint configuration**

Review the created endpoint in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoints
- check the name of the endpoint, its ARN and status
- below you can review the monitoring metrics such as CPU, memory and disk utilization. Further down you can see the endpoint configuration settings with its production variants

In [22]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="https://console.aws.amazo
```

Review **SageMaker REST endpoint**

Wait for the endpoint to deploy.

This cell will take approximately 5-10 minutes to run.

In [23]:
```
%%time

waiter = sm.get_waiter('endpoint_in_service')
waiter.wait(EndpointName=model_ab_endpoint_name)
```

```
CPU times: user 221 ms, sys: 13.6 ms, total: 234 ms
Wall time: 8min 2s
```

Wait until the ^^ endpoint ^^ is deployed

2. Test model

2.1. Test the model on a few sample strings

Here, you will pass sample strings of text to the endpoint in order to see the sentiment. You are given one example of each, however, feel free to play around and change the strings yourself!

Exercise 7

Create an Amazon SageMaker Predictor based on the deployed endpoint.

Instructions: Use the `Predictor` object with the following parameters. Please pass JSON serializer and deserializer objects here, calling them with the functions `JSONLinesSerializer()` and `JSONLinesDeserializer()`, respectively. More information about the serializers can be found here.

```
predictor = Predictor(
    endpoint_name=..., # endpoint name
    serializer=..., # a serializer object, used to encode data
for an inference endpoint
    deserializer=..., # a deserializer object, used to decode
data from an inference endpoint
    sagemaker_session=sess
)
```

In [24]:

```
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONLinesSerializer
from sagemaker.deserializers import JSONLinesDeserializer

inputs = [
    {"features": ["I love this product!"]},
    {"features": ["OK, but not great."]},
    {"features": ["This is not the right product."]},
]

predictor = Predictor(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    endpoint_name=model_ab_endpoint_name, # Replace None
    serializer=JSONLinesSerializer(), # Replace None
    deserializer=JSONLinesDeserializer(), # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    sagemaker_session=sess
)

predicted_classes = predictor.predict(inputs)

for predicted_class in predicted_classes:
    print("Predicted class {} with probability {}".format(predicted_class
```

```
Predicted class 1 with probability 0.9605445861816406
Predicted class 0 with probability 0.5798221230506897
Predicted class -1 with probability 0.7667604684829712
```

2.2. Generate traffic and review the endpoint performance metrics

Now you will generate traffic. To analyze the endpoint performance you will review some of the metrics that Amazon SageMaker emits in CloudWatch: CPU Utilization, Latency and Invocations. Full list of namespaces and metrics can be found here. CloudWatch `get_metric_statistics` documentation can be found here.

But before that, let's create a function that will help to extract the results from CloudWatch and plot them.

In [25]:
```
def plot_endpoint_metrics_for_variants(endpoint_name,
                                       namespace_name,
                                       metric_name,
                                       variant_names,
                                       start_time,
                                       end_time):

    try:
        joint_variant_metrics = None

        for variant_name in variant_names:
            metrics = cw.get_metric_statistics( # extracts the results in
                Namespace=namespace_name, # the namespace of the metric,
                MetricName=metric_name, # the name of the metric, e.g. "C
                StartTime=start_time, # the time stamp that determines th
                EndTime=end_time, # the time stamp that determines the la
                Period=60, # the granularity, in seconds, of the returned
                Statistics=["Sum"], # the metric statistics
                Dimensions=[ # dimensions, as CloudWatch treats each uniq
                    {"Name": "EndpointName", "Value": endpoint_name},
                    {"Name": "VariantName", "Value": variant_name}
                ],
            )

            if metrics["Datapoints"]: # access the results from the disti
                df_metrics = pd.DataFrame(metrics["Datapoints"]) \
                    .sort_values("Timestamp") \
                    .set_index("Timestamp") \
                    .drop("Unit", axis=1) \
                    .rename(columns={"Sum": variant_name}) # rename the c

                if joint_variant_metrics is None:
                    joint_variant_metrics = df_metrics
                else:
                    joint_variant_metrics = joint_variant_metrics.join(df

        joint_variant_metrics.plot(title=metric_name)
    except:
        pass
```

Establish wide enough time bounds to show all the charts using the same timeframe:

In [26]:
```
from datetime import datetime, timedelta

start_time = datetime.now() - timedelta(minutes=30)
end_time = datetime.now() + timedelta(minutes=30)

print('Start Time: {}'.format(start_time))
print('End Time: {}'.format(end_time))
```

```
Start Time: 2022-09-05 07:22:22.029450
End Time: 2022-09-05 08:22:22.029644
```

Set the list of the the variant names to analyze.

In [27]:
```
variant_names = [variantA["VariantName"], variantB["VariantName"]]

print(variant_names)
```

```
['VariantA', 'VariantB']
```

Run some predictions and view the metrics for each variant.

This cell will take approximately 1-2 minutes to run.

In [28]:
```
%%time

for i in range(0, 100):
    predicted_classes = predictor.predict(inputs)
```

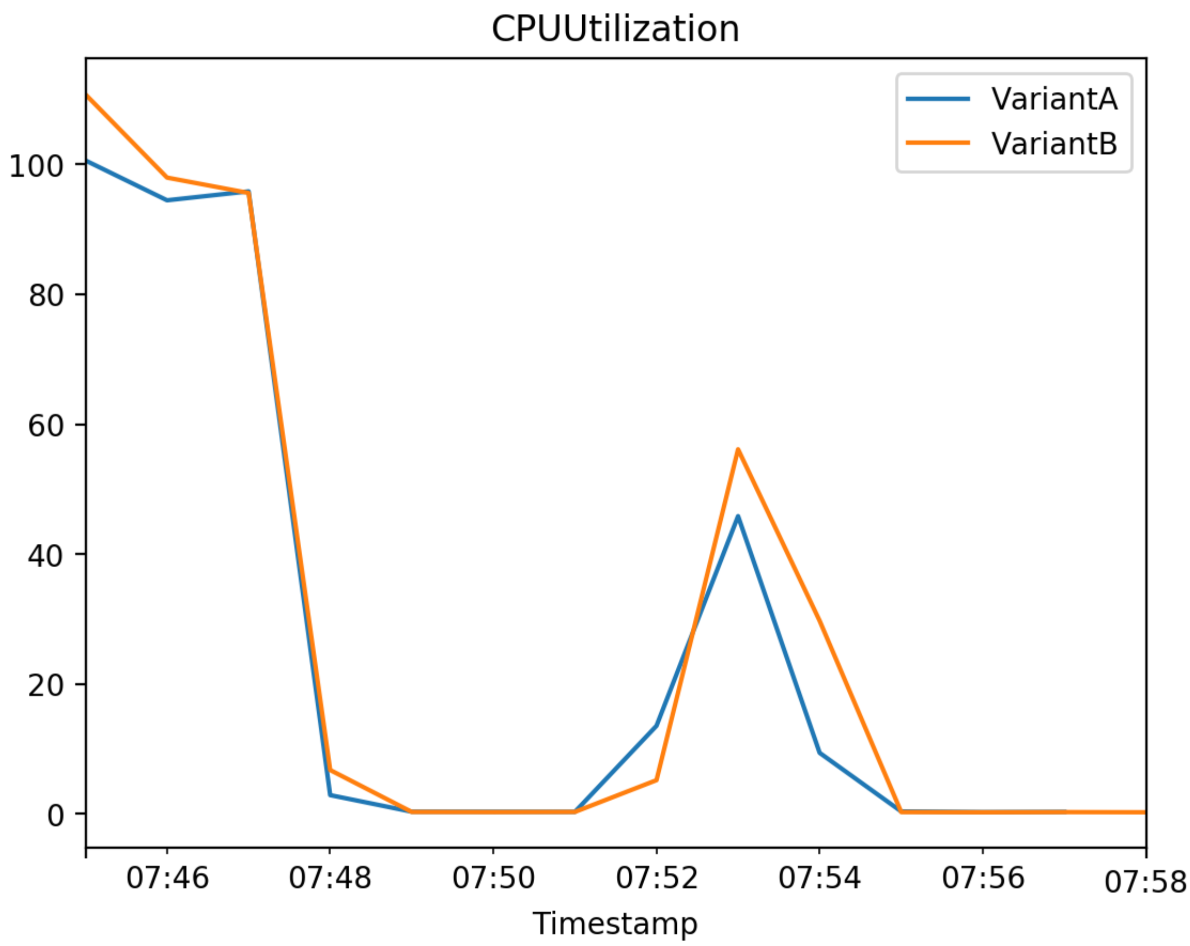
```
CPU times: user 218 ms, sys: 14.9 ms, total: 233 ms
Wall time: 1min 34s
```

Make sure the predictions ^^ above ^^ ran successfully

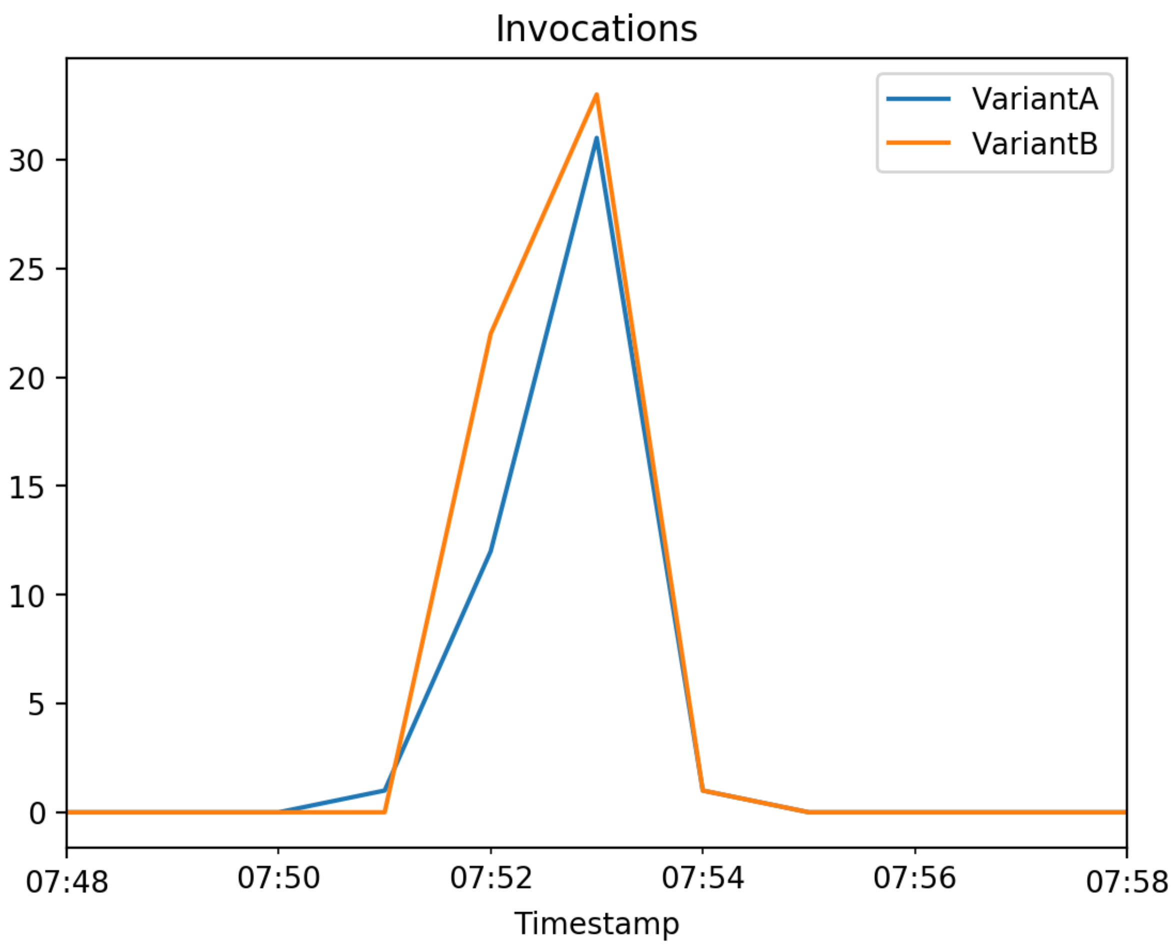
Let's query CloudWatch to get a few metrics that are split across variants. If you see `Metrics not yet available`, please be patient as metrics may take a few mins to appear in CloudWatch.

In [29]:
```
time.sleep(30) # Sleep to accomodate a slight delay in metrics gathering
```

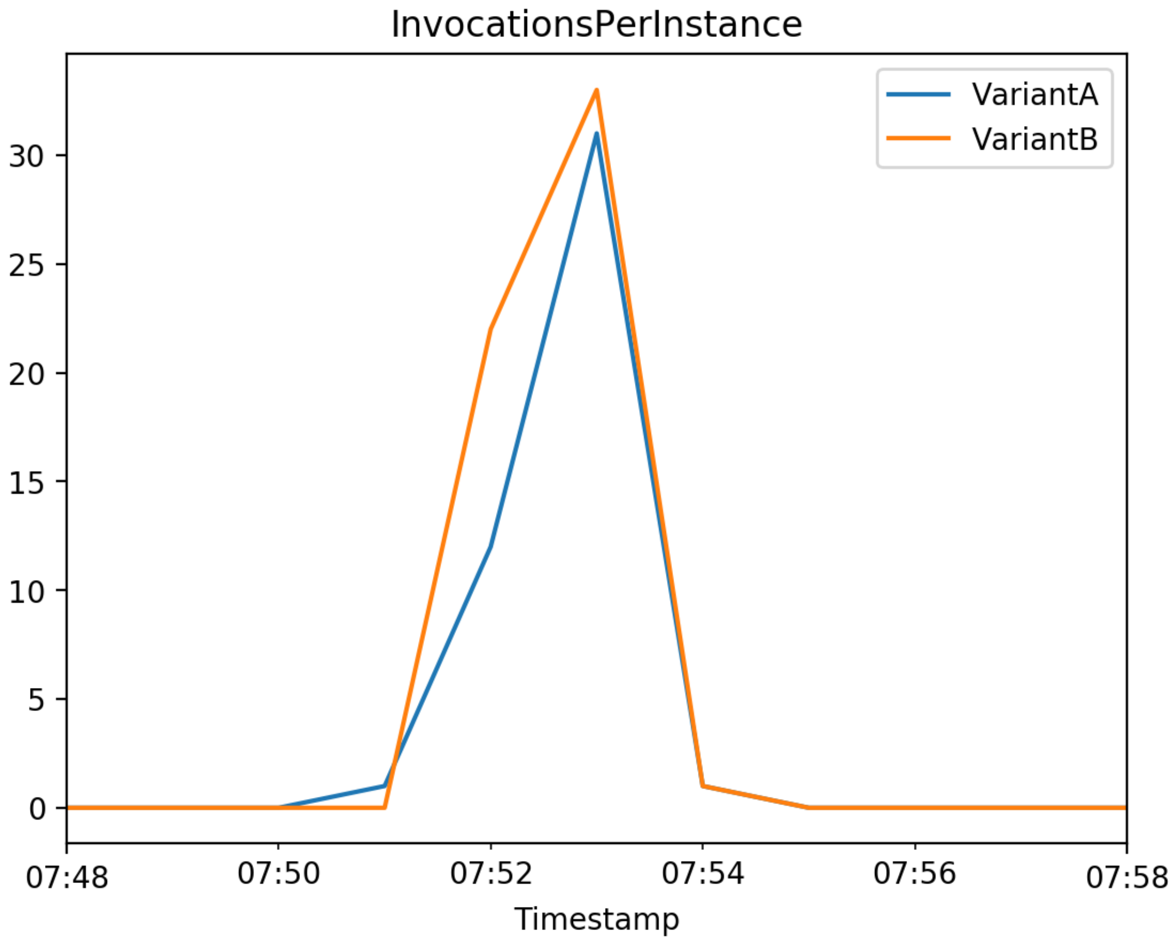
In [30]:
```
# CPUUtilization
# The sum of each individual CPU core's utilization.
# The CPU utilization of each core can range between 0 and 100. For examp
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="/aws/sagemaker/Endpoints",
    metric_name="CPUUtilization",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [31]:
```
# Invocations
# The number of requests sent to a model endpoint.
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="Invocations",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```

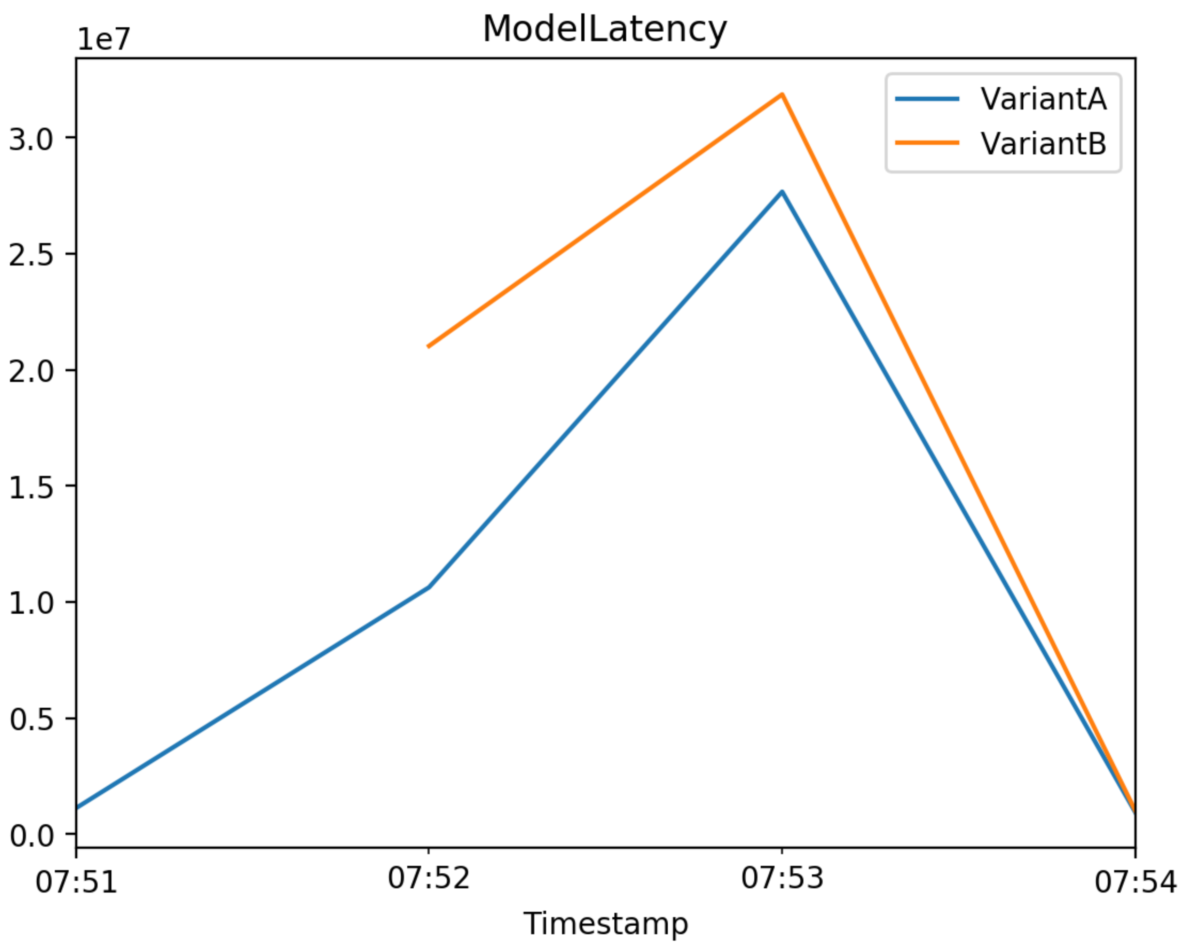


In [32]:
```
# InvocationsPerInstance
# The number of invocations sent to a model, normalized by InstanceCount
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="InvocationsPerInstance",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [33]:

```
# ModelLatency
# The interval of time taken by a model to respond as viewed from SageMak
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="ModelLatency",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



3. Shift the traffic to one variant and review the endpoint performance metrics

Generally, the winning model would need to be chosen. The decision would be made based on the endpoint performance metrics and some other business related evaluations. Here you can assume that the winning model is in the Variant B and shift all traffic to it.

Construct a list with the updated endpoint weights.

***No downtime** occurs during this traffic-shift activity.*

This may take a few minutes. Please be patient.

In [34]:
```
updated_endpoint_config = [
    {
        "VariantName": variantA["VariantName"],
        "DesiredWeight": 0,
    },
    {
        "VariantName": variantB["VariantName"],
        "DesiredWeight": 100,
    },
]
```

Exercise 8

Update variant weights in the configuration of the existing endpoint.

Instructions: Use the `sm.update_endpoint_weights_and_capacities` function, passing the endpoint name and list of updated weights for each of the variants that you defined above.

In [35]:
```
sm.update_endpoint_weights_and_capacities(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    EndpointName=model_ab_endpoint_name, # Replace None
    DesiredWeightsAndCapacities=updated_endpoint_config # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)
```

Out[35]:

```
{'EndpointArn': 'arn:aws:sagemaker:us-east-1:643403534509:endpoint/ab-166
2363360',
 'ResponseMetadata': {'RequestId': 'f87b7bf1-4b75-4433-8048-617932f1b48f'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'f87b7bf1-4b75-4433-8048-617932f1b4
8f',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '81',
   'date': 'Mon, 05 Sep 2022 07:59:21 GMT'},
  'RetryAttempts': 0}}
```


Wait for the ^^ endpoint update ^^ to complete above

This may take a few minutes. Please be patient.

*There is **no downtime** while the update is applying.*

While waiting for the update (or afterwards) you can review the endpoint in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoints
- check the name of the endpoint, its ARN and status (`Updating` or `InService`)
- below you can see the endpoint runtime settings with the updated weights

In [36]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="https://console.aws.amazo
```

Review **SageMaker REST endpoint**

In [37]:
```
waiter = sm.get_waiter("endpoint_in_service")
waiter.wait(EndpointName=model_ab_endpoint_name)
```

Run some more predictions and view the metrics for each variant.

This cell will take approximately 1-2 minutes to run.

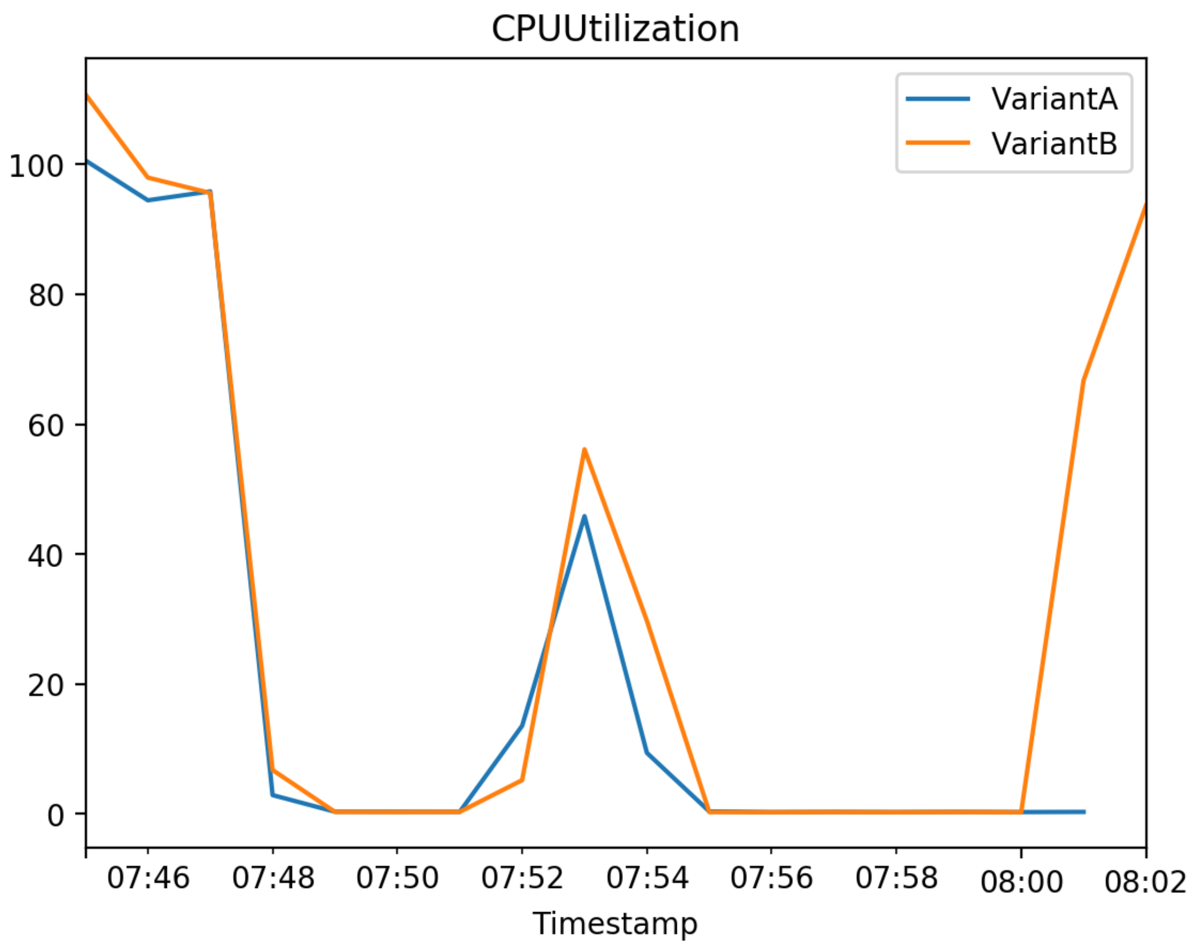
In [38]:
```
%%time

for i in range(0, 100):
    predicted_classes = predictor.predict(inputs)
```

```
CPU times: user 237 ms, sys: 14 ms, total: 252 ms
Wall time: 1min 36s
```

Make sure the predictions ^^ above ^^ ran successfully

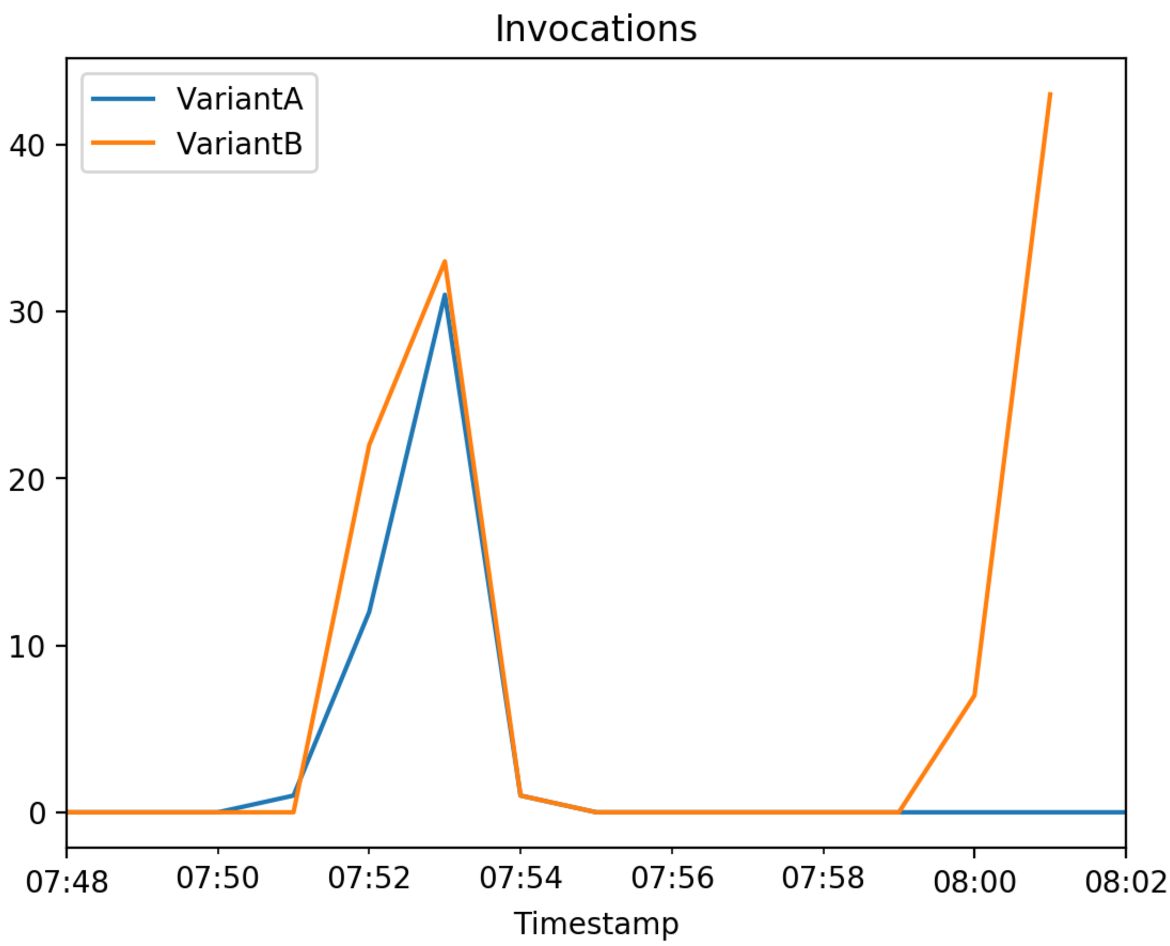
If you see `Metrics not yet available`, please be patient as metrics may take a few minutes to appear in CloudWatch. Compare the results with the plots above.

In [39]:
```
# CPUUtilization
# The sum of each individual CPU core's utilization.
# The CPU utilization of each core can range between 0 and 100. For examp
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="/aws/sagemaker/Endpoints",
    metric_name="CPUUtilization",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



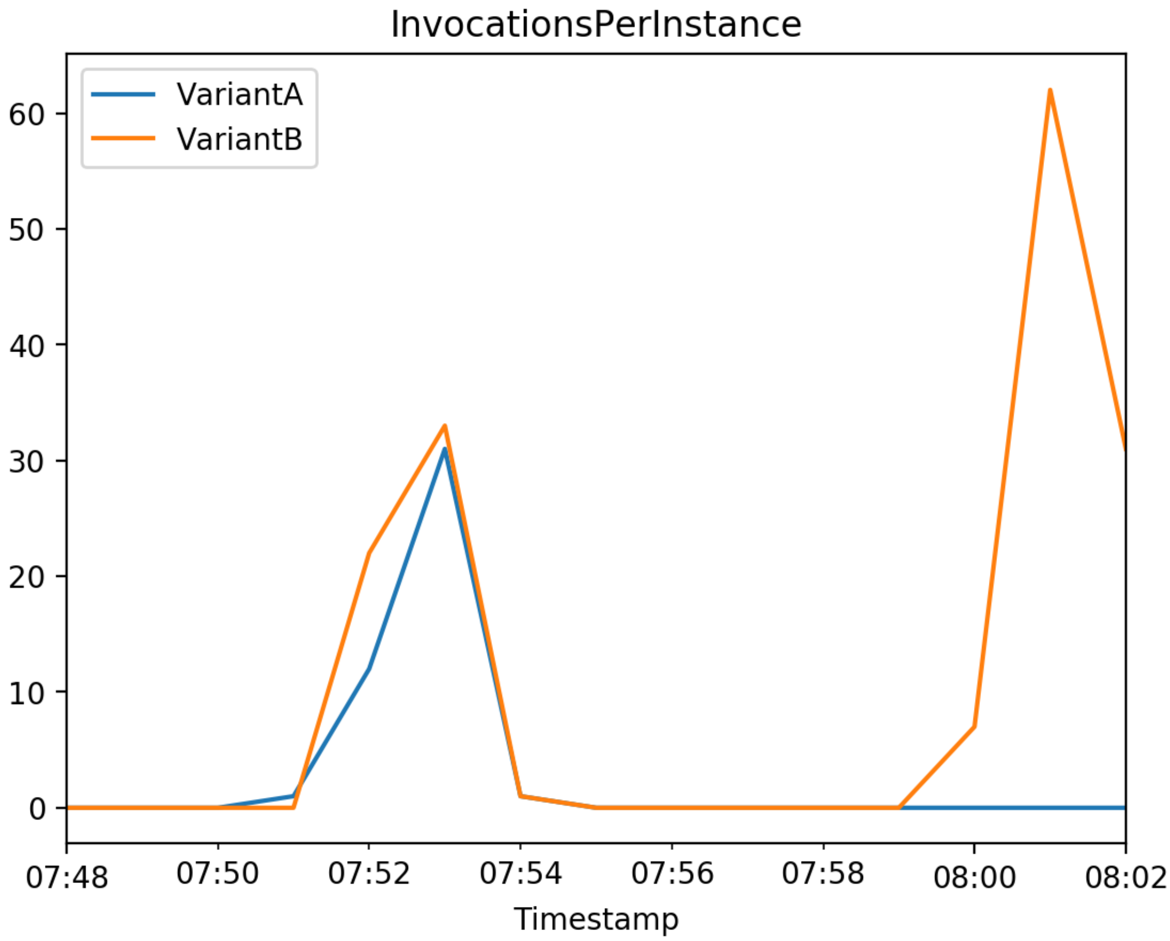
In [40]:

```
# Invocations
# The number of requests sent to a model endpoint.
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="Invocations",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



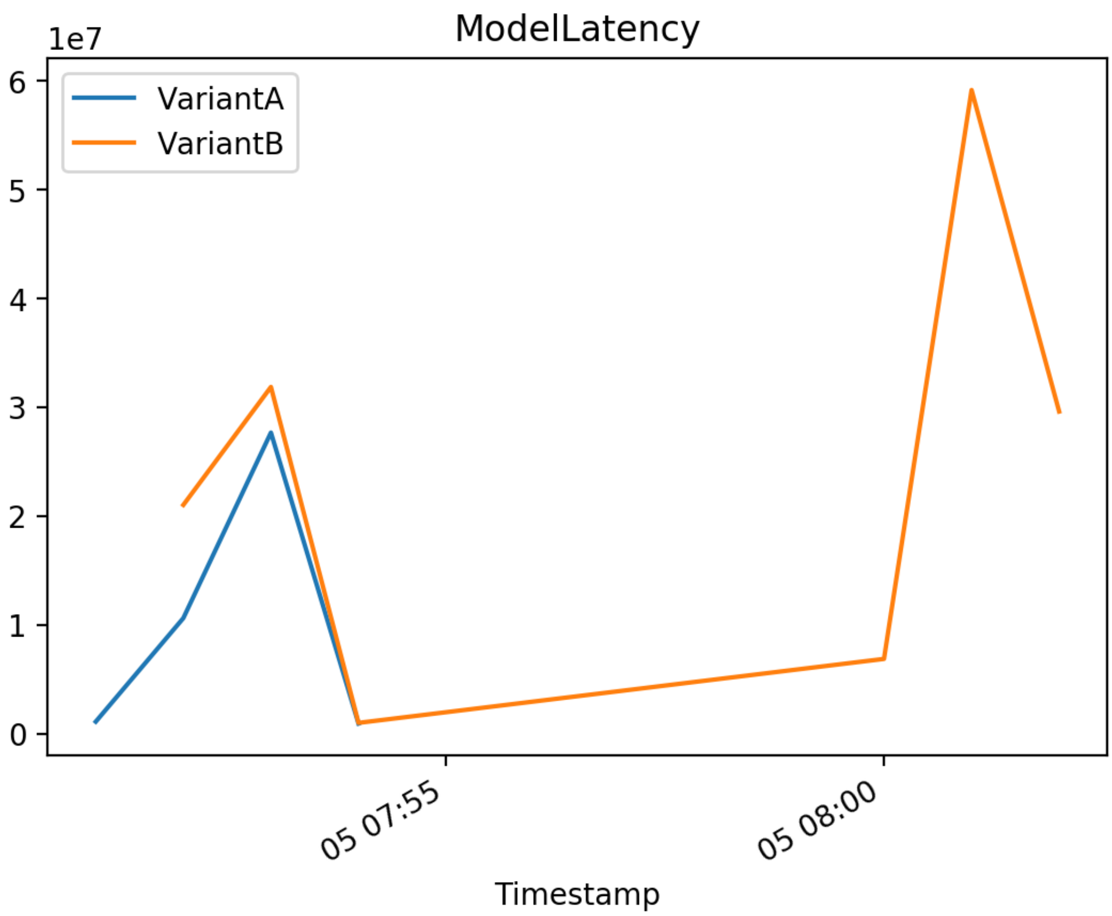
In [41]:

```
# InvocationsPerInstance
# The number of invocations sent to a model, normalized by InstanceCount
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="InvocationsPerInstance",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [42]:

```
# ModelLatency
# The interval of time taken by a model to respond as viewed from SageMak
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="ModelLatency",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



4. Configure one variant to autoscale

Let's configure Variant B to autoscale. You would not autoscale Variant A since no traffic is being passed to it at this time.

First, you need to define a scalable target. It is an AWS resource and in this case you want to scale a `sagemaker` resource as indicated in the `ServiceNameSpace` parameter. Then the `ResourceId` is a SageMaker Endpoint. Because autoscaling is used by other AWS resources, you'll see a few parameters that will remain static for scaling SageMaker Endpoints. Thus the `ScalableDimension` is a set value for SageMaker Endpoint scaling.

You also need to specify a few key parameters that control the min and max behavior for your Machine Learning instances. The `MinCapacity` indicates the minimum number of instances you plan to scale in to. The `MaxCapacity` is the maximum number of instances you want to scale out to. So in this case you always want to have at least 1 instance running and a maximum of 2 during peak periods.

In [43]:
```
autoscale.register_scalable_target(
    ServiceNamespace="sagemaker",
    ResourceId="endpoint/" + model_ab_endpoint_name + "/variant/VariantB"
    ScalableDimension="sagemaker:variant:DesiredInstanceCount",
    MinCapacity=1,
    MaxCapacity=2,
    RoleARN=role,
    SuspendedState={
        "DynamicScalingInSuspended": False,
        "DynamicScalingOutSuspended": False,
        "ScheduledScalingSuspended": False,
    },
)
```

Out[43]:
```
{'ResponseMetadata': {'RequestId': 'ce2fa889-5861-43f5-8704-95a5e112dc5e'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'ce2fa889-5861-43f5-8704-95a5e112dc
5e',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '2',
   'date': 'Mon, 05 Sep 2022 08:02:57 GMT'},
  'RetryAttempts': 0}}
```

In [44]:
```
waiter = sm.get_waiter("endpoint_in_service")
waiter.wait(EndpointName=model_ab_endpoint_name)
```

Check that the parameters from the function above are in the description of the scalable target:

In [45]:
```
autoscale.describe_scalable_targets(
    ServiceNamespace="sagemaker",
    MaxResults=100,
)
```

Out[45]:
```
{'ScalableTargets': [{'ServiceNamespace': 'sagemaker',
   'ResourceId': 'endpoint/ab-1662363360/variant/VariantB',
   'ScalableDimension': 'sagemaker:variant:DesiredInstanceCount',
   'MinCapacity': 1,
   'MaxCapacity': 2,
   'RoleARN': 'arn:aws:iam::643403534509:role/aws-service-role/sagemaker.
application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoSca
ling_SageMakerEndpoint',
   'CreationTime': datetime.datetime(2022, 9, 5, 8, 2, 57, 340000, tzinfo
=tzlocal()),
   'SuspendedState': {'DynamicScalingInSuspended': False,
    'DynamicScalingOutSuspended': False,
    'ScheduledScalingSuspended': False}}],
 'ResponseMetadata': {'RequestId': '18931257-8e70-408d-8b37-2f0a37af4dd9'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '18931257-8e70-408d-8b37-2f0a37af4d
d9',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '521',
   'date': 'Mon, 05 Sep 2022 08:03:04 GMT'},
  'RetryAttempts': 0}}
```

Define and apply scaling policy using the `put_scaling_policy` function. The scaling policy provides additional information about the scaling behavior for your instance. `TargetTrackingScaling` refers to a specific autoscaling type supported by SageMaker, that uses a scaling metric and a target value as the indicator to scale.

In the scaling policy configuration, you have the predefined metric `PredefinedMetricSpecification` which is the number of invocations on your instance and the `TargetValue` which indicates the number of invocations per ML instance you want to allow before triggering your scaling policy. A scale out cooldown of 60 seconds means that after autoscaling successfully scales out it starts to calculate the cooldown time. The scaling policy won't increase the desired capacity again until the cooldown period ends.

The scale in cooldown setting of 300 seconds means that SageMaker will not attempt to start another cooldown policy within 300 seconds of when the last one completed.

In [46]:
```
autoscale.put_scaling_policy(
    PolicyName="bert-reviews-autoscale-policy",
    ServiceNamespace="sagemaker",
    ResourceId="endpoint/" + model_ab_endpoint_name + "/variant/VariantB"
    ScalableDimension="sagemaker:variant:DesiredInstanceCount",
    PolicyType="TargetTrackingScaling",
    TargetTrackingScalingPolicyConfiguration={
        "TargetValue": 2.0, # the number of invocations per ML instance y
        "PredefinedMetricSpecification": {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstan
        },
        "ScaleOutCooldown": 60, # wait time, in seconds, before beginning
        "ScaleInCooldown": 300, # wait time, in seconds, before beginning
    },
)
```

Out[46]:

```
{'PolicyARN': 'arn:aws:autoscaling:us-east-1:643403534509:scalingPolicy:f
1674af7-74ec-45a8-8a77-e5c3b60e7dfe:resource/sagemaker/endpoint/ab-166236
3360/variant/VariantB:policyName/bert-reviews-autoscale-policy',
 'Alarms': [{'AlarmName': 'TargetTracking-endpoint/ab-1662363360/variant/
VariantB-AlarmHigh-35bf164b-d15a-4d65-b6af-7e66aadaa965',
   'AlarmARN': 'arn:aws:cloudwatch:us-east-1:643403534509:alarm:TargetTra
cking-endpoint/ab-1662363360/variant/VariantB-AlarmHigh-35bf164b-d15a-4d6
5-b6af-7e66aadaa965'},
  {'AlarmName': 'TargetTracking-endpoint/ab-1662363360/variant/VariantB-A
larmLow-9b440097-6241-4337-a5f2-95cf0fb9e5d8',
   'AlarmARN': 'arn:aws:cloudwatch:us-east-1:643403534509:alarm:TargetTra
cking-endpoint/ab-1662363360/variant/VariantB-AlarmLow-9b440097-6241-4337
-a5f2-95cf0fb9e5d8'}],
 'ResponseMetadata': {'RequestId': '4039eedd-3e05-4f20-a98e-d4352ecb190b'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '4039eedd-3e05-4f20-a98e-d4352ecb19
0b',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '780',
   'date': 'Mon, 05 Sep 2022 08:03:10 GMT'},
  'RetryAttempts': 0}}
```


In [47]:
```
waiter = sm.get_waiter("endpoint_in_service")
waiter.wait(EndpointName=model_ab_endpoint_name)
```

Generate traffic again and review the endpoint in the AWS console.

This cell will take approximately 1-2 minutes to run.

In [48]:
```
%%time

for i in range(0, 100):
    predicted_classes = predictor.predict(inputs)
```

```
CPU times: user 238 ms, sys: 8.39 ms, total: 246 ms
Wall time: 1min 36s
```

Review the autoscaling:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoints
- below you can see the endpoint runtime settings with the instance counts. You can run the predictions multiple times to observe the increase of the instance count to 2

In [ ]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="https://console.aws.amazo
```

Upload the notebook into S3 bucket for grading purposes.

Note: you may need to click on "Save" button before the upload.

In [ ]:
```
!aws s3 cp ./C3_W2_Assignment.ipynb s3://$bucket/C3_W2_Assignment_Learner
```

```
In [ ]:
```