**Deploying an ML Model is easy with MLFlow and AWS Sagemaker. Step-by-step Tutorial**

[[](https://medium.com/@GenerousDataAnalyst?source=post_page-----d2bde5701d04--------------------------------)](https://medium.com/@GenerousDataAnalyst?source=post_page-----d2bde5701d04--------------------------------)

[[DevOps.dev](https://blog.devops.dev/?source=post_page-----d2bde5701d04--------------------------------)](https://blog.devops.dev/?source=post_page-----d2bde5701d04--------------------------------)

[Oleg Kazanskyi](https://medium.com/@GenerousDataAnalyst?source=post_page-----d2bde5701d04--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fe09fef5bf698&operation=register&redirect=https%3A%2F%2Fblog.devops.dev%2Fdeploying-an-ml-model-is-easy-with-mlflow-and-aws-sagemaker-step-by-step-tutorial-d2bde5701d04&user=Oleg+Kazanskyi&userId=e09fef5bf698&source=post_page-e09fef5bf698----d2bde5701d04---------------------post_header-----------)

Published in

[DevOps.dev](https://blog.devops.dev/?source=post_page-----d2bde5701d04--------------------------------)

·

9 min read

·

Sep 1

103

1

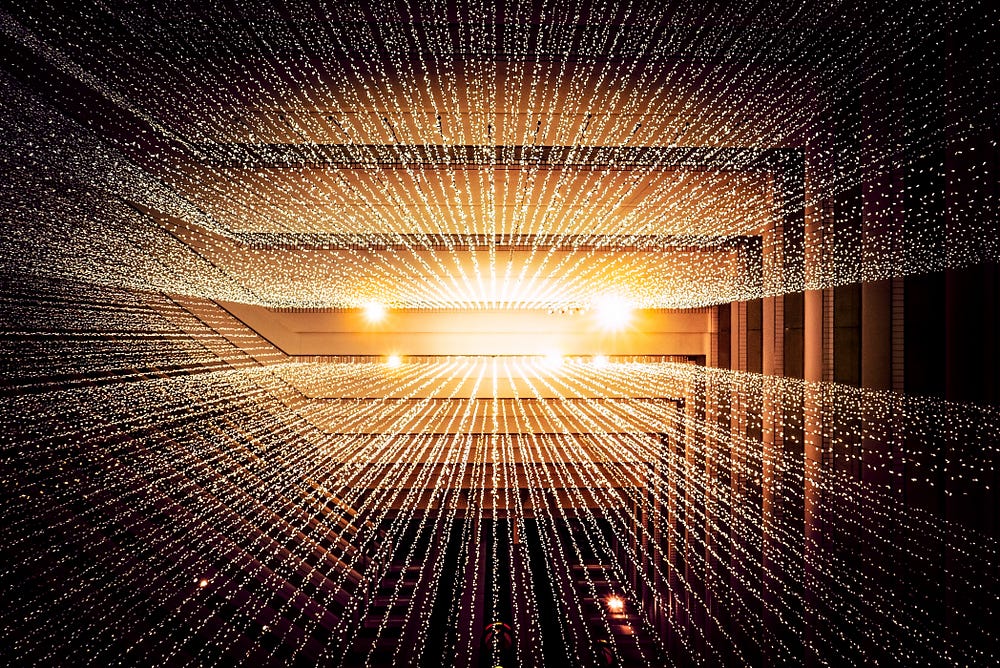


Photo by [Joshua Sortino](https://unsplash.com/@sortino?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

Knowing how to properly fine-tune an ML model is great, but the question arises of how to let other users, inside or outside the company, to use it. That’s when the rubber hits the road because only sharing your model and running it on real data creates actual value for the business.

At the same time, deploying requires a different set of knowledge that has become an important domain in data science. The questions are: “Where to start your introduction with the cloud solutions?”, “What service provider to use?”, “How to deploy and keep the model live?”.

The data analytics world develops rapidly, and working day-on-day on your skills is essential to keep up with the market. The good news is that the new software developments that make data scientist’s life easier are showing up every day.

With the right set of tools pushing the ML model live is easy. Today I’ll show one of the approaches that might work for you. Let’s start learning some ML engineering from here.

We will use:

1. MLFlow
2. Docker Desktop
3. Amazon Web Services / Sagemaker

**WHAT IS MLFLOW**



MLFlow is an open-source software created by Databricks, and originally it was designed to help keep track of machine learning model runs along with the parameters used and quality metrics. It simplifies data analytics’ routine a simple set up and convenient UI. Everyone can learn basics about how to use it in less than a day.

As a product of logging a model, MLflow creates metadata files with information about the utilized libraries and the model itself. This important detail makes further deployment much easier as we can duplicate the environment and the model on a cloud.

You can install MLFlow from pip

pip install mlflow

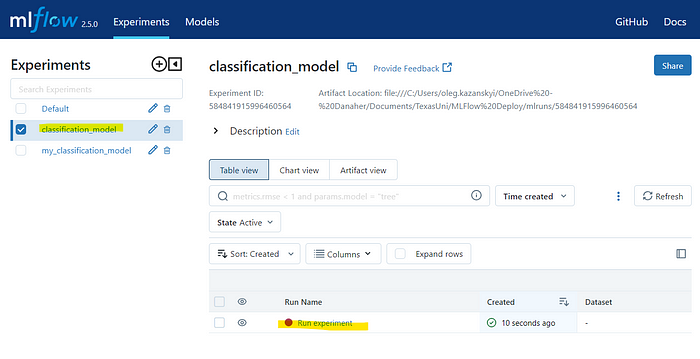
I’m creating a dummy classification model that I will use as a sample in this tutorial. I am using MLFlow functions like log\_param(), log\_model() and log\_metric() to record model details that we are going to check later.

import mlflow  
import mlflow.sklearn  
  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import train\_test\_split  
  
from sklearn.datasets import make\_classification  
from sklearn.linear\_model import LogisticRegression  
# define dataset  
X, y = make\_classification(n\_samples=50000, n\_features=3, n\_informative=3, n\_redundant=0, n\_classes=2, random\_state=1)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
mlflow.set\_experiment('classification\_model')  
  
with mlflow.start\_run(run\_name='My model experiment') as run:  
   
 # add parameters for tuning  
 c = 0.1  
 solver = 'liblinear'  
   
 # Log parameters to MLFlow  
 mlflow.log\_param('c', c)  
 mlflow.log\_param('solver', solver)  
  
 # train the model  
 lr = LogisticRegression(C = c, solver = solver)  
 lr.fit(X\_train, y\_train)  
 predictions = lr.predict(X\_test)  
  
 # Log the model to MLFlow  
 mlflow.sklearn.log\_model(lr, 'logistic-regression-model')  
   
 # log model performance to MLFlow  
 mse = mean\_squared\_error(y\_test, predictions)  
 mlflow.log\_metric('mse', mse)  
 print('mse: %f' % mse)

We need to move to the location of the the python file with the training (please replace with your location) and run MLFlow UI in this folder to check the model.

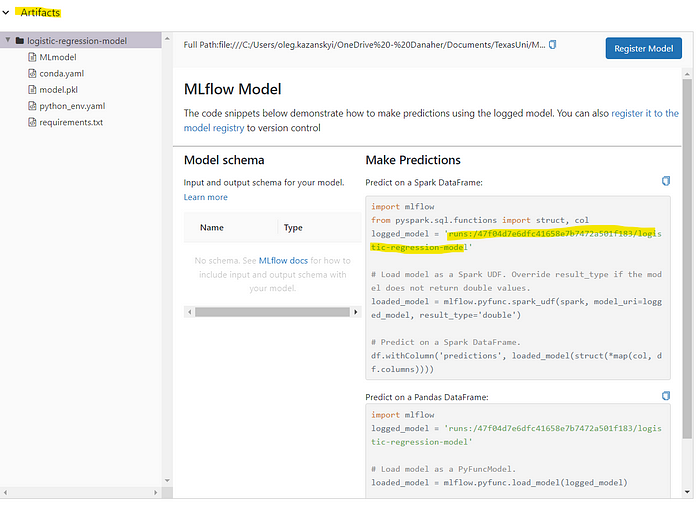
cd "C:\Users\ ......"  
  
mlflow ui

After the UI is running in your terminal we can navigate to [http://127.0.0.1:5000](http://127.0.0.1:5000/) in the browser, MLFlow show us the visual UI there.



Here we find our logged experiment (left pane) and runs (on the right bottom side).

Once you click on your run name, in my case it’s “Run experiment”, a new navigation window opens with more details. That’s where you can find all the metrics, parameters and the model itself stored. We need to pay attention to the **logged\_model** string as we would require this later.



**LET’S PREPARE AMAZON WEB SERVICES (AWS)**



Sagemaker is a cloud solution from Amazon created for data scientists to train, test and deploy models with serverless experience. An ability of Sagemaker to store model and access it through the endpoint makes it a great fit for our case.

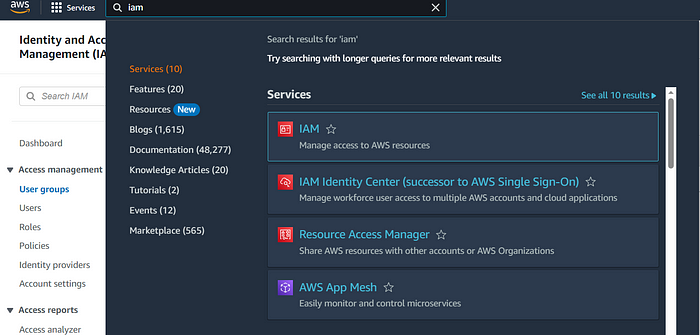
Other solutions are Azure ML Studio if you prefer Microsoft Solutions, or native for MLFlow — Seldon’s MLServer.

Let’s start our implementation with installing python libraries required for communication with Amazon Sagemaker.

pip install awscli  
pip install boto3

Now we need to navigate to AWS in our browser. Please create an account if you don’t have one.

In AWS we go to “Identity and Access Management tool (IAM)”, where we provide the required permissions to deploy and use models to users.

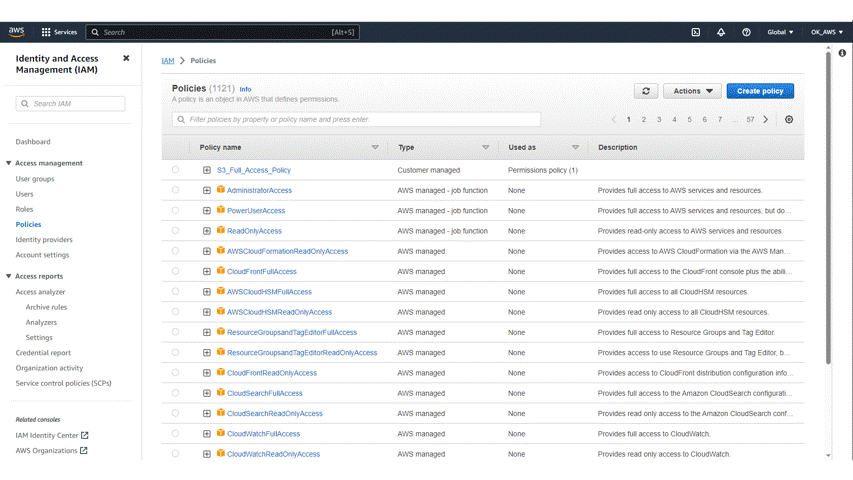


We would need to create a user with specific set of permissions.

Please remember that the good practice is to create user groups with the specific permissions and assign users to them. But for the simplicity I will work with a single user.

At first, I will create a new security AWS policy that we would need later outside of some out of the box policies in the AWS list.

**Begin with creation of a new Policy for AWS S3**



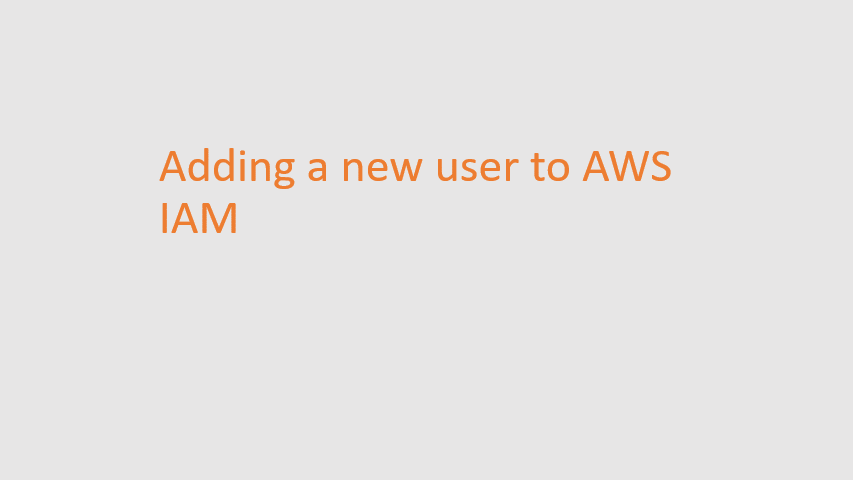
S3 is a cloud storage for AWS. We would need it to save our model. In order to use it we need to assign the proper rights to the user. And to do this we create a policy with the S3 full permission that I will assign to the user later.

The process is next:

In the “IAM” window on the left pane select “Policies” and press “create policy”. In the new window you need to find “S3” and select all “actions” and all “resources” for it *(We’re adding roles without any restrictions for the simplicity of the task, your organization may require some additional limitations for the security reasons)*.

Once the policy has been created you should find it in the list of AWS’s policies.

**We are ready to create a User.**



I recorded a short GIF with all the steps above and here is a short description following.

Go to Users and press “New User”. We would need to add several policies to the user, so select “Attach policies Directly” on the next menu. Two of the required policies are in the list and one we’ve just created.

Add another 3 policies to this user:

* AmazonEC2ContainerRegistryFullAccess
* AmazonSageMakerFullAccess
* S3\_Full\_Access\_Policy (this is manually created)

After the new user is added to the system, we need to get access keys for it. These are required to set up communication with AWS from our machine.

To do this select the new user in the list of users and in the new window go to “Create Access Key”. We need it for the “Command Line Interface (CLI)” so select this option after the menu shows up. Once the keys are created please **store them in a secure place**. We would need them later.

Let’s use the keys right away to set up AWS connection from our machine.  
to do so, we need another terminal window with the proper Python environment where all the packages are installed. Please type

aws configure

You should get a request to paste AWS Access Key ID (it’s our Access Key), AWS Secret Access Key (that is our Secret Access Key), default region name (go to <https://console.aws.amazon.com/> and check the highlighted region on the right top corner) and the output format where we type “json”. The output should look like this:

AWS Access Key ID [**\*\*\*\****\*]: AKI****\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*****\**  
AWS Secret Access Key [**\*\*\*\****\*]: PYb****\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*****\**  
Default region name [**\*\*\*\****\*]: us-east-2  
Default output format [****\*\*\*\*****\**]: json

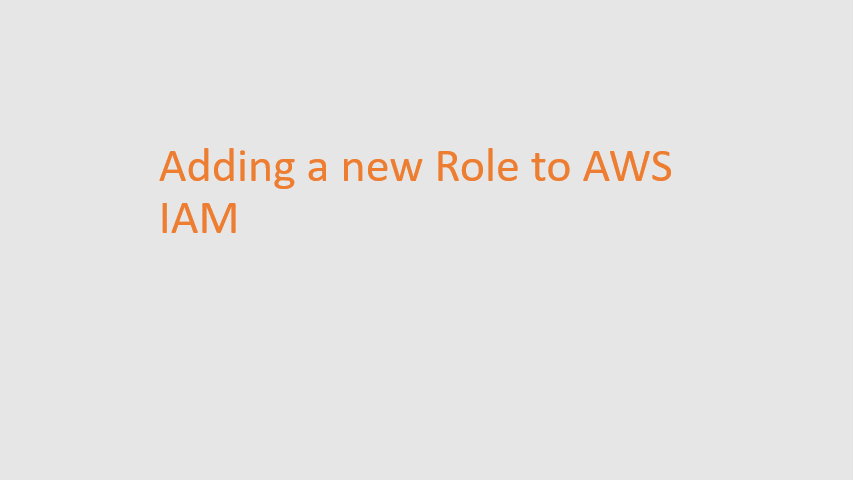
If you did not get any errors, it means the connection is properly set up.

There is another one tiny thing we need to create in AWS that is required by MLFlow, that is a role with permission to use Sagemaker.

**Create a role**

Go to “Roles” on the left pane of the IAM and press “Create Role”. In the new window we need to select “AWS Service” and in the list of services find “Sagemaker”. Give a name to the role and press “Create”.

After the new role has been created we would need it’s ARN (Amazon Resource Name)



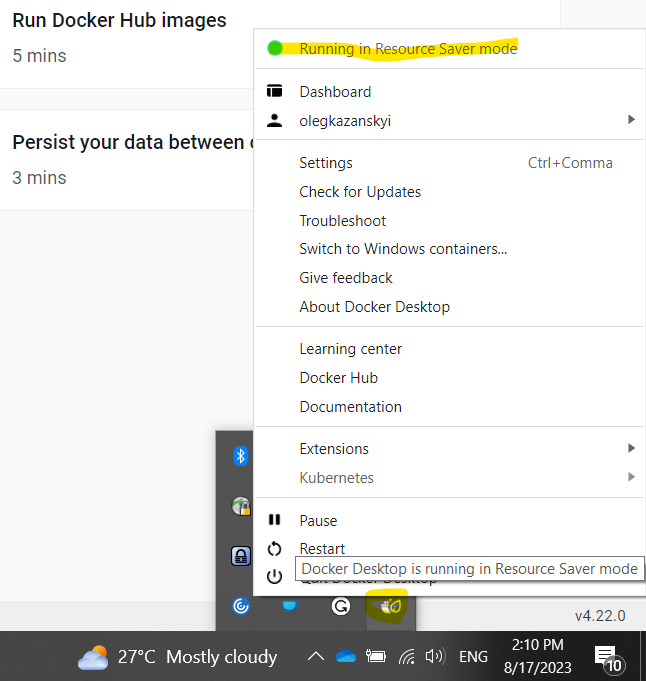


**YES, ONE MORE TOOL! :-)**

In short, we would need to pack our model into container to simplify deployment. Docker manages to make it a simple process.

First of all, we need a “Docker desktop” running on our machine. It’s free and can be loaded from [here](https://www.docker.com/products/docker-desktop/).

Be sure that your docker is running properly otherwise you may get an error.



After you make it successfully run, we need to start working on the final stage — model deployment.

**MODEL DEPLOYMENT**

At first we need to create a container with our model that we will further deploy. With MLFlow it’s only one command with a little trick.

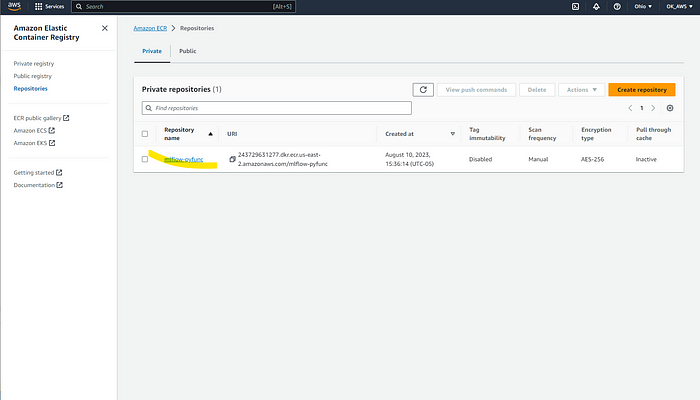
First, we need to navigate to the MLflow folder where all the artifacts are stored. This folder “mlruns” is automatically created in the location of the script with ML model training. There we would need to find a subfolder that hides under the run\_id. In my case the path looks like this:

cd "C:\Users\\*\*\*\MLFlow Deploy\mlruns\884328759121468859\a8536f7973ae49358dabb97d1a0e9c3c\artifacts\logistic-regression-model"

In the “logistic-regression-model” folder there are files containing all information about used libraries and environments with extensions \*.yaml, \*.pkl, \*.txt. Those files are used to create a container that fully replicates the environment used to train the model. To prepare the container I type in terminal window:

aws sagemaker build-and-push-container

It takes a while but after it’s over you should find a container on your desktop Docker and also in AWS ECR (Elastic Container Registry)



As we have the container ready, we can deploy a model to the Sagemaker.

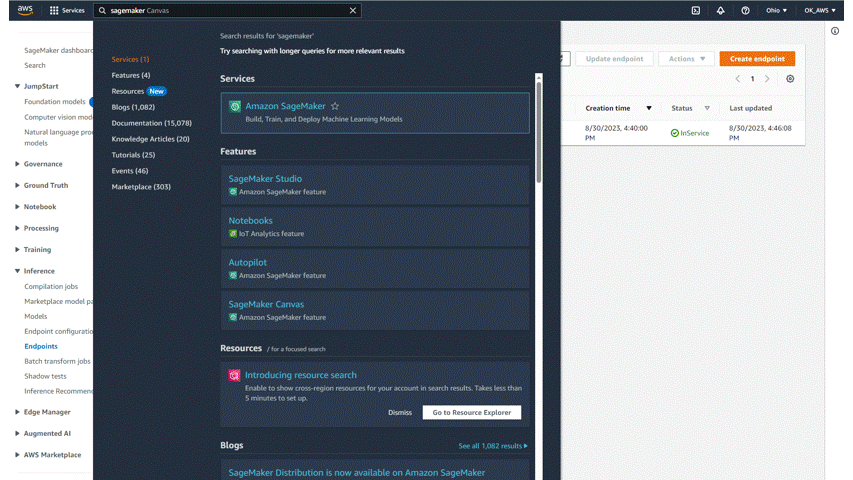
To keep all the deployment details in one place I‘m’ creating a new file “deployment.py” in the folder with the model training file.

It’s almost fully copy-pasted from the [MLFlow documentation](https://mlflow.org/docs/latest/python_api/mlflow.sagemaker.html), and you can refer to it.

from mlflow.deployments import get\_deploy\_client  
  
region = 'us-east-2'  
# You can run "aws sts get-caller-identity" to get your AWS ID  
aws\_id = '243729631277'  
# Use ARN from the role we created in AWS with the full permission to Sagemaker  
arn = 'arn:aws:iam::\*\*\*\*\*\*\*(\*\*\*\*\*\*\*:user/sagemaker\_deploy\_user'  
#Create your name of the future application  
app\_name = 'OK\_model\_application'  
# you may find model uri in "mlflow ui" recorded as "logged\_model"  
model\_uri = f'runs:/47f04d7e6dfc41658e7b7472a501f183/classification\_model'  
# tag\_id is equal to the mlflow version. It is used in Docker url  
tag\_id = '2.5.0'  
  
  
image\_url = aws\_id + '.dkr.ecr.' + region + '.amazonaws.com/mlflow-pyfunc:' + tag\_id  
  
  
config = dict(  
 execution\_role\_arn=arn,  
 bucket\_name="New-s3-bucket",  
 image\_url=image\_url,  
 region\_name="us-east-2",  
 archive=False,  
 instance\_type="ml.t2.medium",  
 instance\_count=1,  
 synchronous=True,  
 timeout\_seconds=3600,  
 variant\_name="prod-variant-1",  
 tags={"training\_timestamp": "2023-08-31"},  
)  
  
client = get\_deploy\_client("sagemaker")  
  
client.create\_deployment(  
 #you can use any name you want to see in Sagemaker"  
 "my-deployment-attemp",  
 model\_uri=model\_uri,  
 flavor="python\_function",  
 config=config,  
)

If everything went well, you should see models and endpoints created in AWS Sagemaker.

Go to the Inference menu to find these.



**FINAL STEP. TESTING THE ENDPOINT**

Let’s create another \*.py file to keep everything in one place

import pandas as pd  
import numpy as np  
import json  
import boto3  
  
global app\_name  
global region  
app\_name = 'my-deployment-logisticRegression'  
region = 'us-east-2'  
  
def check\_status(app\_name):  
 sage\_client = boto3.client('sagemaker', region\_name=region)  
 endpoint\_description = sage\_client.describe\_endpoint(EndpointName=app\_name)  
 endpoint\_status = endpoint\_description['EndpointStatus']  
 return endpoint\_status  
  
def query\_endpoint(app\_name, input\_json):  
 client = boto3.session.Session().client('sagemaker-runtime', region)  
  
 response = client.invoke\_endpoint(  
 EndpointName = app\_name,  
 Body = input\_json,  
 ContentType = 'application/json'#'; format=pandas-split',  
 )  
  
 preds = response['Body'].read().decode('ascii')  
 preds = json.loads(preds)  
 print('Received response: {}'.format(preds))  
 return preds  
  
# Check endpoint status  
print('Application status is {}'.format(check\_status(app\_name)))  
  
#Let's create a test array that we'll use to test our model  
arr\_predict = np.random.randn(2,3)  
  
# Create test data and make inference from endpoint  
query\_input = pd.DataFrame(arr\_predict).to\_dict(orient='split')  
print(query\_input)  
  
data = {"dataframe\_split": query\_input}  
  
byte\_data = json.dumps(data).encode('utf-8')  
  
predictions = query\_endpoint(app\_name=app\_name, input\_json=byte\_data)  
print(predictions)

As an output we got

Input Query: {'index': [0, 1], 'columns': [0, 1, 2], 'data': [[2.144226772387518, 0.2629843635807662, -0.6612623851504181], [-1.8784664271968212, -2.060043013874982, 3.232492188432877]]}  
  
Received response: {'predictions': [1, 0]}

We got an output from our model predicting one of 2 inputs as True. It means that our model is live and working and now we can share it with the world!

**FINAL WORD**

I expected this tutorial be shorter, but I wrapped it as much as I could trying not to miss anything important.  
I hope some of you will find it helpful. If you have any questions you can reach me on <https://www.linkedin.com/in/oleg-kazanskyi/>