**Guide to Building AWS Lambda Functions from ECR Images to Manage SageMaker Inference Endpoints**

**We breakdown the process of building a lambda function for machine-learning API endpoints**

Lambda is a powerful serverless managed service on the AWS cloud. At its introduction in 2014, Lambda offered a unique event-driven abstraction that took the hassle of managing compute resources out of the equation. In many ways, it was the first true serverless cloud service.

Today, they play a crucial role in stitching together enterprise machine-learning applications due to their nimble ability to perform critical machine-learning pipeline tasks such as batch data processing, small to moderate model training, workflow triggers, model deployments, and more.

Lambdas can be considered small empty compute sandboxes and, therefore, require that we provide the operating system, code, and dependencies required to execute its tasks. This tutorial will build a lambda function from a docker image. The goal of our lambda will be to download resources from S3, receive JSON payloads, perform feature engineering, and feed them to a sagemaker endpoint for inference.

*This tutorial is part of a series about building hardware-optimized SageMaker endpoints with the Intel AI Analytics Toolkit. You can find all of the code for this tutorial* [***here***](https://github.com/eduand-alvarez/ai-kit-sagemaker-templates/tree/main/xgboost-daal4py/1_lambda-container)*.*

**Preparing our Container Environment**

Usually, we would be able to package our code and files into a .zip file and leverage an AWS public image to run our workload on lambda. However, lambda has strict size requirements — the components of our work can’t exceed 50MB zipped or 250MB unzipped, and sadly most machine-learning python packages exceed this. This is where we turn to docker, arguably a cleaner and more intuitive way to build our lambda images.

1. Navigate to the Cloud9 IDE in the AWS console. You are welcome to build the container image locally, but we will use Cloud9 since it comes with all of the AWS permissions and resources we require.   
   - Select m5.large instance (or larger if you intend on creating a larger image)  
   - Select Ubuntu as your Platform

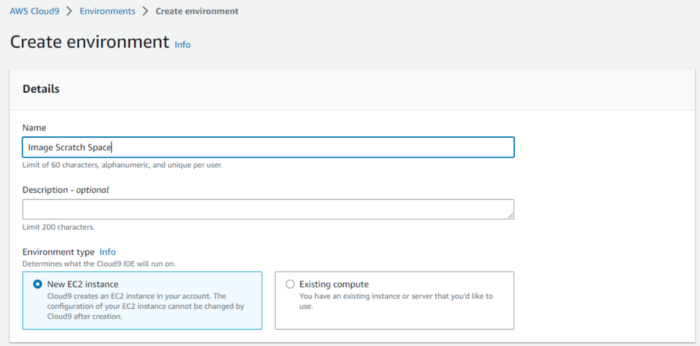


Figure 1. Interface for creating Cloud9 Environments — Image by Author

2. Use touch to create a Dockerfile, requirements.txt, and handler.py files. After creation, you should see all files in the directory tree on the left — double-click on each file to open and edit them.

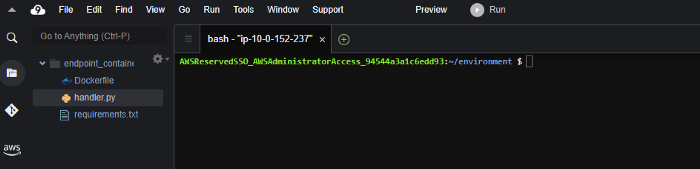


Figure 2. Cloud9 IDE Interface — Image by Author

3. Below, you’ll find the code for the lambda handler script, which will receive lambda events and return predictions from our model. In the example below, we invoke an endpoint as part of a SageMaker pipeline. Let’s review the different functions in the script:

* **process\_data** downloads the transformation.sav file, which includes the label binarizer, standard scaler, and one-hot encoding transforms and applies them to our payload along with some basic data processing steps.
* **sagemaker\_endpoint** invokes an active SageMaker endpoint, sends our processed payload, and returns prediction results.

import os  
import json  
import boto3  
import pickle  
import sklearn  
import warnings  
import tarfile  
  
warnings.simplefilter("ignore")  
# grab environment variables  
ENDPOINT\_NAME = os.environ['ENDPOINT\_NAME']  
runtime= boto3.client('runtime.sagemaker')  
trans\_bucket = "your transformation bucket name"  
s3\_trans\_key = "path to transformation.sav in your bucket"  
s3 = boto3.resource('s3')  
  
def process\_data(event):  
 trans = pickle.loads(s3.Object(trans\_bucket, s3\_trans\_key).get()['Body'].read())  
 event.pop('Phone')  
 event['Area Code'] = int(event['Area Code'])  
 obj\_data = [[value for key,value in event.items() if key in trans['obj\_cols']]]  
 num\_data = [[value for key,value in event.items() if key in trans['num\_cols']]]  
   
 obj\_data = trans['One\_Hot'].transform(obj\_data).toarray()  
 num\_data = trans['scaler'].transform(num\_data)  
   
 obj\_data = [str(i) for i in obj\_data[0]]  
 num\_data = [str(i) for i in num\_data[0]]  
   
 data = obj\_data + num\_data  
   
 return ",".join(data)  
   
def sagemaker\_endpoint(event, context):  
   
 payload = process\_data(event)  
   
 response = runtime.invoke\_endpoint(EndpointName=ENDPOINT\_NAME,  
 ContentType='text/csv',  
 Body=payload)  
   
 # decode and extract prediction   
 response\_preds = json.loads(response['Body'].read().decode())  
 result = response\_preds['predictions'][0]['score']  
 predicted\_label = 'True' if result > 0.39 else 'False'   
   
 return predicted\_label

4. Let’s build our requirements.txt file. We will use this to install the necessary dependencies into our container image.

boto3  
numpy==1.21.4  
pandas==1.3.5  
sagemaker==2.93.0  
scikit-learn==0.24.2

5. Our Dockerfile is responsible for configuring our image. We start with a publicly available Linux ubuntu image from AWS’ public container registry. This image comes preloaded with python 3.8. The rest of the commands in the Dockerfile will take care of copying files, installing dependencies, and executing the functions in our handler.py script.

# download base image  
FROM public.ecr.aws/lambda/python:3.8  
  
# copy our lambda handler script  
COPY app.py ${LAMBDA\_TASK\_ROOT}  
  
  
# install our dependencies  
COPY requirements.txt .  
RUN pip3 --no-cache-dir install -r requirements.txt --target "${LAMBDA\_TASK\_ROOT}"  
  
# execute function for sagemaker endpoint prediction  
RUN echo Utilizing SageMaker Endpoint  
CMD [ "app.sagemaker\_endpoint" ]

**Building Image and Registering to ECR**

We will need to make our image available to our lambda function. There are other image registries, but we will use AWS Elastic Container Registry (ECR).

*If you need help building your image and pushing it to ECR, follow this tutorial:*[***Creating an ECR Registry and Pushing a Docker Image***](https://medium.com/@eduand-alvarez/creating-an-ecr-registry-and-pushing-a-docker-image-93e372e74ff7?source=your_stories_page-------------------------------------)

**Building Lambda Function from Image on ECR**

1. To build your lambda function, navigate to the lambda service, click on **create function,** and select **Container Image.** Provide a *function name* and the *container image URI*, and click on **Create Function**.

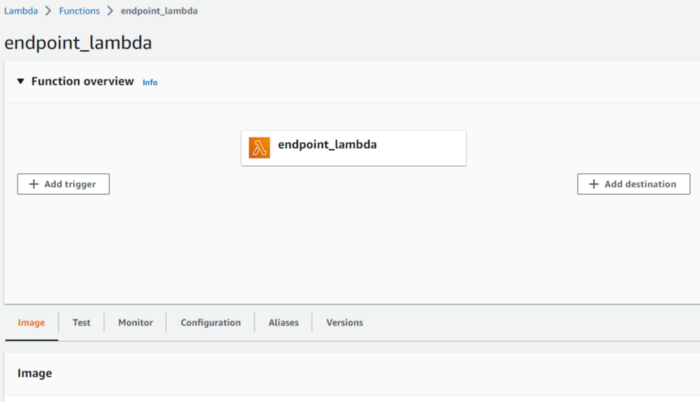


Figure 3. Lambda management interface — Image by Author

2. If we tried to test our function right now, we would likely get errors because our IAM role does not have permission to access SageMaker or S3 resources. To address this, we will go to **Configuration> Execution Role Name> Add Permissions> Attach Policy** and attach **“AmazonS3FullAccess”** and **“AmazonSageMakerFullAccess.”** In a production scenario, you may want to limit the access you give specific services, but that is beyond the scope of our tutorial.

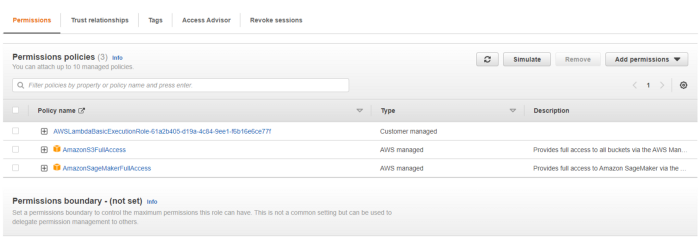


Figure 4. AWS Policies that needed to be added to our Lambda function — Image by Author

3. Under **Configurations > Edit environment variables**, add an environment variable for your SageMaker endpoint name.The name of your endpoint can be found under **Inference > Endpoints** in the AWS SageMaker console.

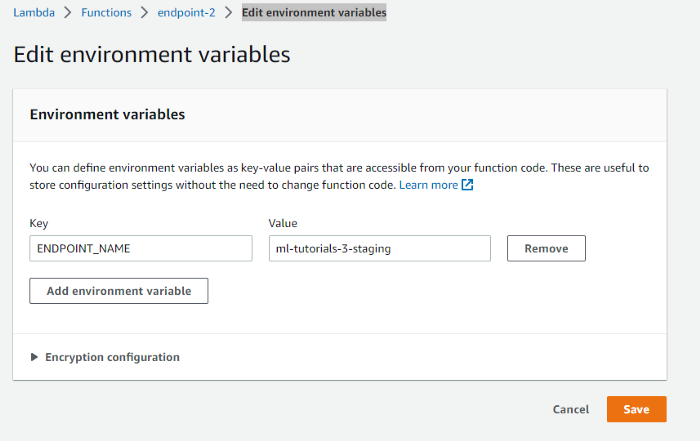


Figure 5. Adding our SageMaker endpoint name to environment variables for easy access by our function — Image by Author

4. Once we set permissions, we are ready to test our lambda function. Select the Test tab and paste the payload below into the **Event JSON** space.

{  
 "State": "PA",  
 "Account Length": "163",  
 "Area Code": "806",  
 "Phone": "403-2562",  
 "Int'l Plan": "no",  
 "VMail Plan": "yes",  
 "VMail Message": "300",  
 "Day Mins": "8.1622040217391",  
 "Day Calls": "3",  
 "Day Charge": "7.579173703343681",  
 "Eve Mins": "3.9330349941938625",  
 "Eve Calls": "4",  
 "Eve Charge": "6.508638877091394",  
 "Night Mins": "4.065759457683862",  
 "Night Calls": "100",  
 "Night Charge": "5.1116239145545554",  
 "Intl Mins": "4.9281602056057885",  
 "Intl Calls": "6",  
 "Intl Charge": "5.673203040696216",  
 "CustServ Calls": "3"  
 }

Click on **Test**. Your test might fail due to the server timing out on the first request made to your endpoint. If so, attempt the test again, and you should see a response with the inference response, “True.”

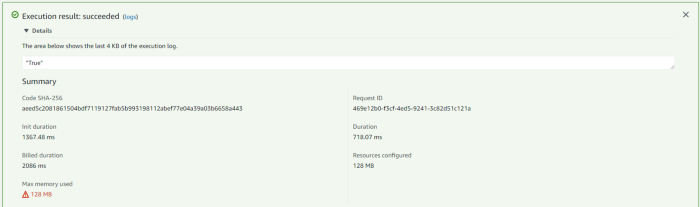


Figure 7. Response from Lambda Test — Image by Author

Congratulations, you have successfully built a lambda function to manage your SageMaker endpoint resource.

**Conclusion and Discussion**

AWS lambda provides a serverless option for managing small components of your application. Due to Lambda’s space constraints, most machine learning and data science packages require dedicated container images.

With the information in this tutorial, you should be able to build compelling serverless microservice architectures to support your own machine-learning applications.