**MLOps:**

Typically covers these areas:

* Deployment and automation
* Reproducibility of models and predictions
* Diagnostics
* Governance and regulatory compliance
* Scalability
* Collaboration
* Business use

**Amazon SageMaker:**

Sagemaker is an AWS based machine learning platform that enables developers to build machine learning models, train data, deploy an inference point on the public cloud. Sagemaker consists of various services such as Ground Truth for build and manage training data sets. SageMaker Notebooks are one-click provisioned notebooks with EC2(Elastic Compute), and SageMaker Studio that is an integrated development environment(IDE) for machine learning.

SageMaker leverages EC2 computing resources for training a machine learning model and running a deployed inference. Along with SageMaker NoteBooks, there are bindings for a number of languages, including Ruby, Python, Java, Node.js.

**ML Concepts:**

A close up of a device

Description automatically generated

<https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-mlconcepts.html>

**Terraform:**

Terraform is a tool to deliver infrastructure as code, typically written in HCL.

HCL (Hashicorp Configuration Language) is somewhat similar to JSON (JavaScript Object Notation). It has some distinctive characteristics as an infrastructure as code software tool

* Automate infrastructure provision
* Write declarative configuration file
* Consistent and repeatable workflows
* Reproducible and reusable infrastructure
* Versioning infrastructure with shared code

**Terraform, Sagemaker & Lambda functions**:

Once a model training job is finished, the trained model will be written into an S3 bucket. S3 PUT event can be associated with a lambda function to trigger model deployment. Additionally, we can set CloudWatch to call the Lambda function to trigger the retrain function. Therefore, we can invoke periodic re-trainings via CloudWatch events to generate new models. The new model put in S3 location triggers the update of the endpoint. The components required for the solution are:

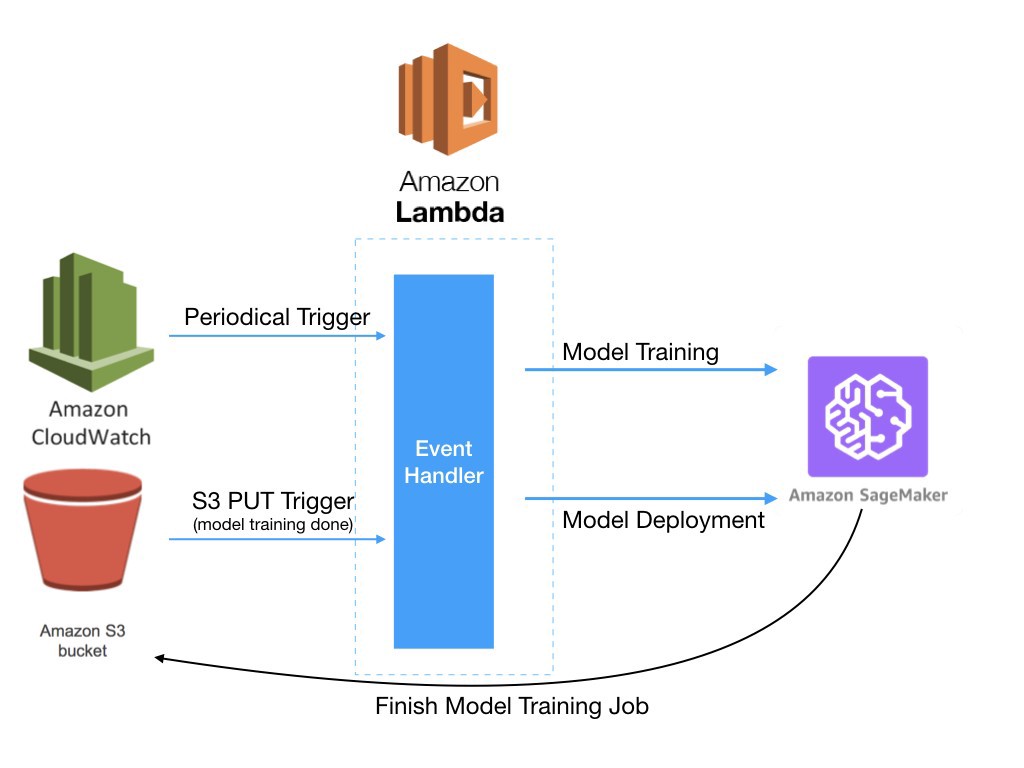
* CloudWatch event for periodic trigger of re-trainings (assume a batch)
* Lambda function to be called by S3 PUT event or by CloudWatch event

**AWS Sagemaker Architecture with Terraform**

A screenshot of a cell phone

Description automatically generated

**Architecture with Lambda, S3 and CloudWatch Events:**



We will look at the architecture more in depth in the subsequent sections.

**Overview**

The following diagram illustrates the overall AWS architecture with terraform automation. To achieve **Reproducibility of models**with terraform we would need specific terraform configurations as seen below.

A screenshot of a cell phone

Description automatically generated

AWS architecture with terraform

Here are the resources that are required for this integration.

* IAM(Identity and Management) role and policy for Lambda function

A lambda function must have a few CloudWatch logs operations and SageMaker endpoint invocation privileges in the policy. I identified the policy under the module iam in the same place of the existing IAM configurations.

* Lambda function code with permissions from CloudWatch and S3

In terraform we could use either a filename argument for local file or s3\_bucket argument for specific S3 location to load the lambda deployment package. We archived the local python code with the template provider and uploaded it to S3 path. This bucket must reside in the same AWS region where we will create the Lambda function.

* CloudWatch event to trigger periodic retraining

We would want to have periodic model redeployment besides S3 PUT event triggers the model redeployment and update the existing endpoint. This could be used as a batch job to update the model. We can add any functions in the lambda function for this CloudWatch event.

We took the rudimentary dataset and a simple scikit-learn estimator for the example notebook. The goal of the given notebook was to predict a housing price with the built model from boston housing price dataset. We used the gradient boosting regressor algorithm (provided in ECR) for training data. The prepared Jupyter notebook was located at source/notebooks/sagemaker-terraform-boston.ipynb in the repository and loaded in aws\_s3\_bucket resource block to upload it to S3 bucket properly.

**Terraform Modules**

Terraform [modules](https://www.terraform.io/docs/configuration/modules.html) provide an easy way to abstract common blocks of configuration into reusable infrastructure elements. Let’s think each element would be vpc, EC2, lambda function, each AWS component level that is relevant to each terraform resource level. To write a module, we can apply the same concepts that we would for any configuration. Modules are collections of .tf files containing resources, input [variables](https://www.terraform.io/docs/configuration/variables.html), and [outputs](https://www.terraform.io/docs/configuration/outputs.html), which exist outside the root folder of your configuration.

Here’s the directory structure for the Terraform building blocks. We have the additional directories for new components such as cloudwatch and lambda .

$ tree --charset=o -I "\*.template"

A screenshot of a cell phone

Description automatically generated

Terraform building blocks

I used the terraform v0.12.6 and the providers as follows. provider.archive was added.

$ cd main  
$ terraform version  
**Terraform v0.12.6**+ provider.archive v1.3.0  
+ provider.aws v2.23.0  
+ provider.local v1.4.0  
+ provider.template v2.1.2

We can provide the deployment package of the Lambda function either directly as a filename or indirectly via S3 location by using s3\_bucket and s3\_key arguments.

s3\_bucket = var.bucket\_name  
s3\_key = "lambda\_function.zip"

I used S3 bucket and the path because I had to give some variables in the lambda function python code before compression. Terraform data template\_file could take into account in this place to render the local python code with some exported outputs variables from terraform modules. Terraform data archive\_file compresses the rendered file of the prepared python code and the required libraries under the directory lambda\_function to output\_path as the zipped file. Right after the zipped file created we can call the resource aws\_s3\_bucket\_object to upload the file to S3 location as usual. There is no pain in embedding the variables that were defined in the terraform.tfvars file in the lambda function straight, which avoids modifications in the python code of the lambda function.

Install the required python libraries with pip commands locally for the lambda deployment package before calling terraform apply command.

mkdir modules/s3/lambda\_function  
cd modules/s3/lambda\_function  
pip install boto3 -t .  
pip install requests -t .  
pip install datetime -t .

Archive lambda function and required libraries

One drawback of using data archive\_file is that generated hash values vary every time in python runtime. So, it would be better to keep the python code consistent and immutable, not change the code often. Using template\_file and giving variables to the file with vars arguments will merit in this case as it turns out.

Let’s take a look at the Lambda module now. It looks straightforward at a glance with s3\_bucket and s3\_key arguments to specify where the compressed lambda function is located. depends\_on takes the lambda object which is the uploaded zip file prior to this lambda function creation. Especially when we adopt terraform modularized approach, we should be careful. In this resource snippet the zipped lambda code must be uploaded before a lambda function is created obviously.

I used depends\_on twice in this example. The first case: the template file needs to happen prior to archiving the file and the python libraries.

data "archive\_file" "default" {  
 type = "zip"  
 output\_path = "${path.module}/lambda\_function.zip"  
 source\_dir = "${path.module}/lambda\_function" depends\_on = [local\_file.to\_dir]  
}

The second case: Uploading the archived zip file (the Lambda deployment package) must be processed prior to new lambda function generation.

resource "aws\_lambda\_function" "default" {  
 handler = "lambda\_function.lambda\_handler"  
 function\_name = "lambda\_function"...  
 runtime = "python3.7"  
 depends\_on = [var.lambda\_object]  
}

**Deployment**

Terraform main code is located under the main directory and here is the place we can issue terraform plan and terraform apply commands. Before running terraform plan and terraform apply, there are a few set-ups we need to perform:

*Complete code:*[*https://github.com/keshava/terraform-sagemaker-redux*](https://github.com/keshava/terraform-sagemaker-redux)

A screenshot of a social media post

Description automatically generated

**i**. Copy terraform\_backend.tf.template to terraform\_backend.tf and modify values accordingly. You need to manually create an S3 bucket or use an existing one to store the Terraform state file. Please configure bucket key region in the file.

terraform {  
 required\_version = "0.12.6"  
 backend "s3" {  
 bucket = "<bucket-name>"  
 key = "sagemaker-sample/terraform.tfstate"  
 region = "<region>"  
 }  
}

**ii**. Copy terraform.tfvars.template to terraform.tfvars and modify values accordingly. You don’t need to create any buckets specified in here; they’re to be created by terraform apply.

aws\_region = "<aws\_region>"  
aws\_profile = "<aws\_profile>"  
iam\_name = "<iam\_name>"  
iam\_name\_lambda = "<iam\_name\_lambda>"  
identifier = "sagemaker.amazonaws.com"  
identifier\_lambda = "lambda.amazonaws.com"  
notebook\_bucket\_name = "<notebook\_bucket\_name>"  
sagemaker\_bucket\_name = "<sagemaker\_bucket\_name>"  
sagemaker\_notebook\_name = "<sagemaker\_notebook\_name>"  
endpoint\_name = "<endpoint\_name>"  
event\_name = "<event\_name>"

**iii.** Modify lambda\_function.py in the directory modules/s3/template and replace <your bucket>with the appropriate bucket name of yours.

'DataSource': {  
 'S3DataSource': {  
 'S3DataType': 'S3Prefix',  
 'S3Uri': 's3://**<your bucket>**/sagemaker/sample/boston\_housing.csv',  
 'S3DataDistributionType': 'FullyReplicated',  
 }  
}  
...OutputDataConfig={  
 'S3OutputPath': 's3://**<your bucket>**/sagemaker/sample/output'  
}  
...PrimaryContainer={  
 'ContainerHostname': 'model-Container',  
 'Image': '354813040037.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-scikit-learn:0.20.0-cpu-py3',  
 'ModelDataUrl': f's3://**<your bucket>**/sagemaker/sample/output/{training\_job\_name}/output/model.tar.gz',  
 'Environment': {  
 'SAGEMAKER\_PROGRAM': 'scikit\_learn\_gradient.py',  
 'SAGEMAKER\_REGION':'ap-northeast-1',  
 'SAGEMAKER\_SUBMIT\_DIRECTORY': src\_path  
 },  
}

**iv**. Once the above files are created, simply run through the following terraform commands. Remember to ensure all commands return ok and to **review the terraform plan before applying**.

$ terraform init  
$ terraform validate  
Success! The configuration is valid.  
$ terraform plan -var-file=terraform.tfvars  
Plan: 20 to add, 0 to change, 0 to destroy.  
$ terraform apply -var-file=terraform.tfvars # yes

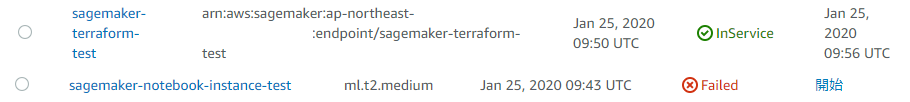
Bugs:

*Error: error waiting for sagemaker notebook instance (sagemaker-notebook-instance-test) to create: unexpected state ‘Failed’, wanted target ‘InService’. last error: %!s(<nil>)*

You may face the error under the sagemaker module after terraform apply command as seen above. This is one defect in this code. We configured *“Lifecycle Configuration”* policy such that the provided shell scripts can’t run for longer than 5 minutes by nohup operation in the shell script.

But the condition seems the same for the Sagemaker notebook also. The notebook initialization can’t take more than 5 minutes and it seems waiting for the configured lifecycle configuration to be completed at the end. The behavior results to incomplete notebook initialization while the endpoint deployment succeeds.

If you encounter this error, you will see this error in AWS console also. All resources have been created and the inference endpoint enabled finally except the Sagemaker notebook resource in case if you saw the error. I configured the endpoint name sagemaker-terraform-test in this experiment.



**Summary:**

we’ve discussed the use of terraform to code AWS components such as IAM(Identity and Access Management), SageMaker Notebook, S3 buckets, Lambda and CloudWatch event in this article. We brought a CloudWatch event to update the endpoint and retraining newer dataset and redeploy the inference endpoint by Lambda for **Reproducibility of models**with terraform.