

# Operationalizing an AWS ML Project

The goal of this project will be to use several important tools and features of AWS to adjust improve, configure, and prepare a machine-learning model for production-grade deployment.

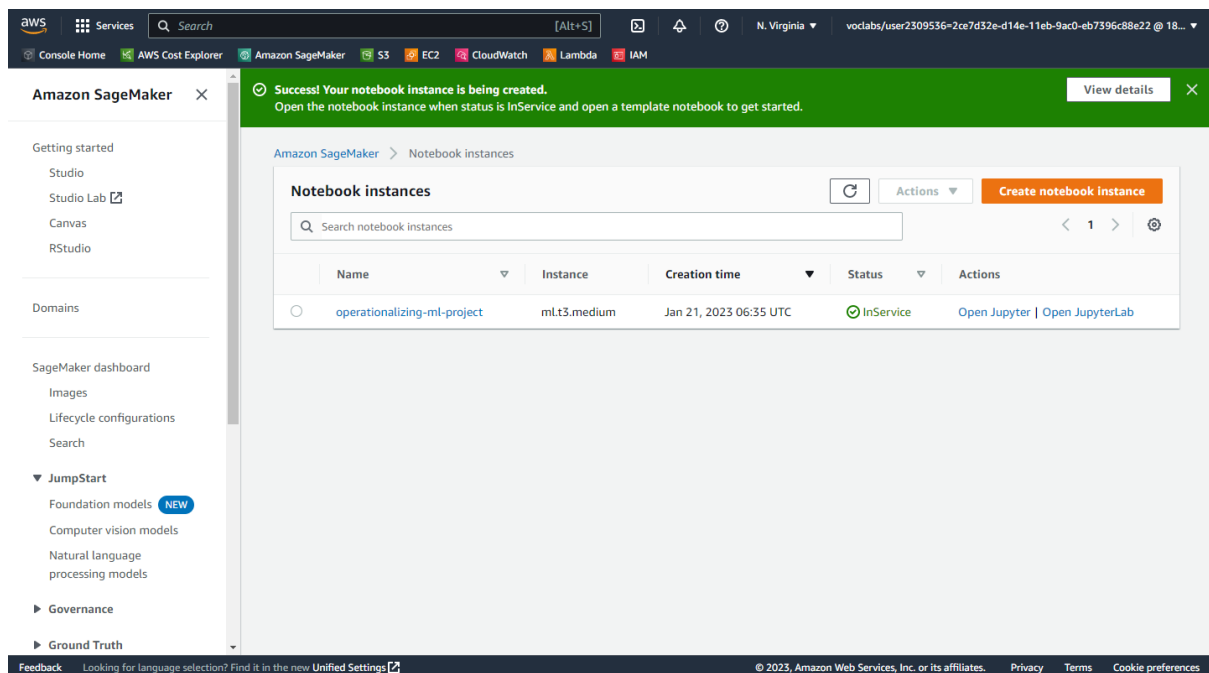
## Training and Deployment on Sagemaker

### Initial Setup

1. Create and open a sagemaker notebook instance
2. Install unzip command

```
sudo yum install unzip -y
```

I choose `ml.t3.medium` as the cheapest compute instance just for running the jupyter notebook.



### Download and Upload Data to an S3 Bucket

I create an s3 bucket with the name `operationalizing-ml`.

```
wget -nc https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
unzip -q dogImages.zip
aws s3 cp ./dogImages s3://operationalizing-ml/dataset
```

The screenshot shows the Amazon S3 console interface. On the left, there's a navigation menu with options like Buckets, Access Points, and Storage Lens. The main area displays an 'Account snapshot' and a list of buckets. The bucket 'operationalizing-ml-project' is highlighted with a green box. Below it, other buckets like 'sagemaker-us-east-1-188932755969' and 'udacitysolutions' are listed.

Name	AWS Region	Access	Creation date
operationalizing-ml-project	US East (N. Virginia) us-east-1	Bucket and objects not public	January 21, 2023, 14:00:07 (UTC+07:00)
sagemaker-us-east-1-188932755969	US East (N. Virginia) us-east-1	Objects can be public	January 19, 2023, 12:40:25 (UTC+07:00)
udacitysolutions	US East (N. Virginia) us-east-1	Bucket and objects not public	January 20, 2023, 14:20:33 (UTC+07:00)

## Training and Deployment

### Single Instance

In my opinion because this training uses a small epoch, I don't need to spend more for a GPU instance, and a CPU-optimised instance like `m1.c5.2xlarge` is enough. It charges **\$0.408/hour** and took **11 minutes and 3 seconds** to finish training.

### Multi Instance

For the multi-instance training, I used spot instances. And the cheapest spot instance available in this account is `m1.c5.2xlarge`. It provides an affordable price of **\$0.174/hour**. I used 2 instances and took **22 minutes and 33 seconds** to finish training in total.

### Endpoint

I created an endpoint from multi-instance estimator

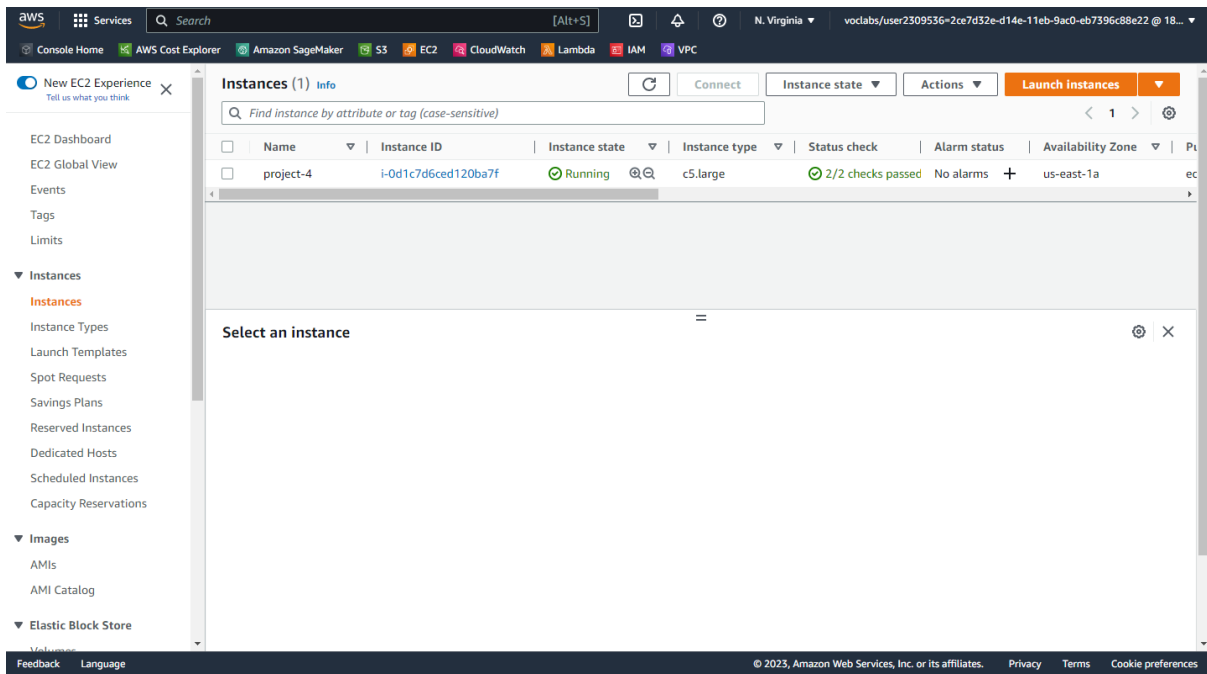
The screenshot shows the Amazon SageMaker console interface. On the left, there's a navigation menu with options like Ground Truth, Notebook, Processing, Training, and Inference. The main area displays the 'Endpoints' page with a table listing the endpoints.

Name	ARN	Creation time	Status	Last updated
pytorch-inference-2023-01-24-07-44-31-770	arn:aws:sagemaker:us-east-1:188932755969:endpoint/pytorch-inference-2023-01-24-07-44-31-770	Jan 24, 2023 07:44 UTC	InService	Jan 24, 2023 07:46 UTC

# EC2 Training

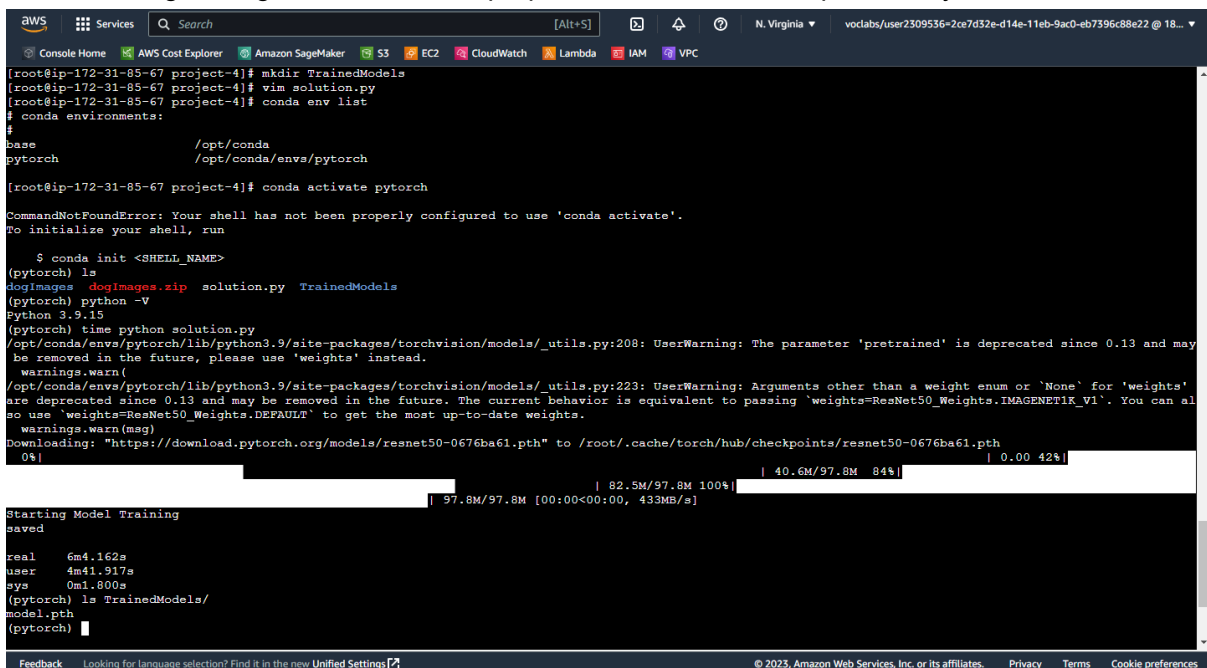
## EC2 Setup

From previous training on Sagemaker, I used `ml.c5.2xlarge`. When I take a look at CloudWatch data. It shows me that the training doesn't use that much storage. So I choose `ml.c5.large` with **Amazon Deep Learning AMI PyTorch 1.13.1 (Amazon Linux 2)**.



## EC2 Model training

Unlike training on sagemaker, the data preparation has to be set up manually.



# Sagemaker Training vs EC2 Training

sagemaker:

- invoke a training job
- input data from s3
- save model data to s3

ec2:

- training on local instance
- load data from local instance directory
- save model data to local directory

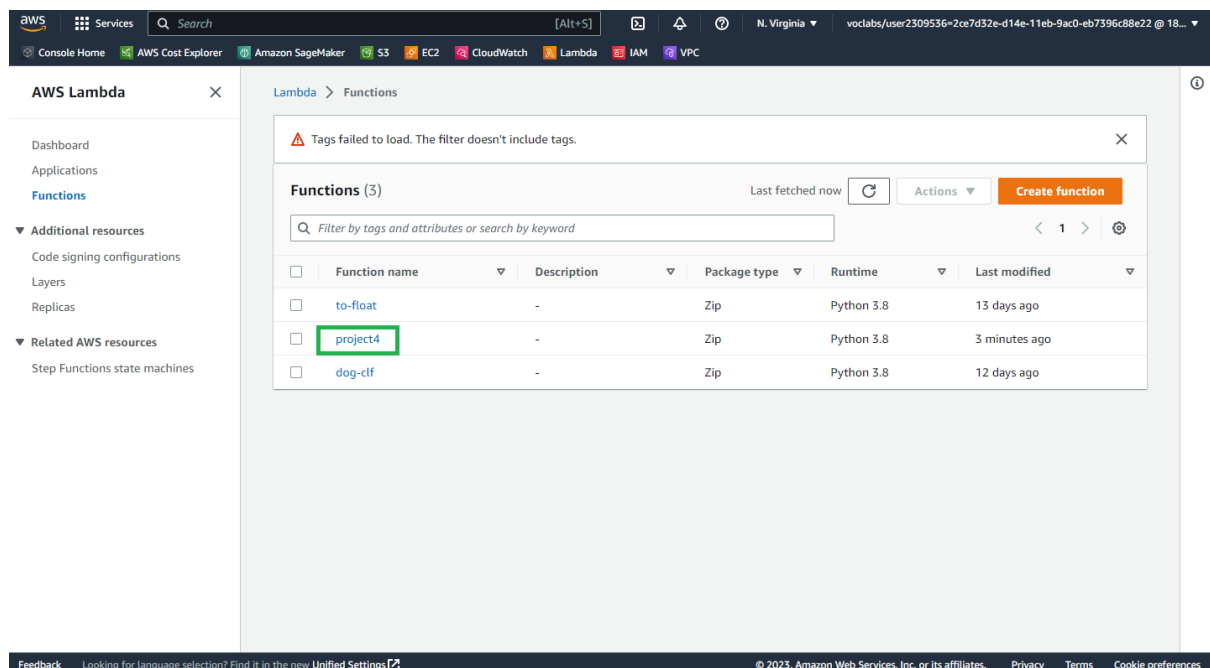
## Lambda Function

This lambda function will invoke an endpoint

pytorch-inference-2023-01-24-07-44-31-770. It only accepts a request with content type application/json and returns data with the following format:

```
{
  'statusCode': 200,
  'headers': { 'Content-Type': 'text/plain', 'Access-Control-Allow-Origin':
  '*' },
  'type-result':str(type(result)),
  'Content-Type-In':str(context),
  'body': json.dumps(sss)
}
```

The detail about this lambda function can be found at `src/code/lamdafunction.py`.



The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with the AWS logo, a search bar, and various service icons. Below this, the 'AWS Lambda' console is open, displaying a list of functions. A sidebar on the left contains navigation links like 'Dashboard', 'Applications', 'Functions', and 'Additional resources'. The main content area shows a table of functions with columns for 'Function name', 'Description', 'Package type', 'Runtime', and 'Last modified'. Three functions are listed: 'to-float', 'project4', and 'dog-clf'. The 'project4' function is highlighted with a green box. Above the table, there's a message indicating that tags failed to load. The footer of the console contains feedback links, a language selection prompt, and copyright information for Amazon Web Services, Inc.

Function name	Description	Package type	Runtime	Last modified
to-float	-	Zip	Python 3.8	13 days ago
project4	-	Zip	Python 3.8	3 minutes ago
dog-clf	-	Zip	Python 3.8	12 days ago

## Security and Testing

By default for security purposes, AWS Lambda does not have access to AWS Sagemaker. Certain authentication methods are required. One of the methods is by using the `IAM` role with the corresponding policies. In this case `AmazonSageMakerFullAccess` policy. It grants the lambda function broader permissions to access SageMaker services. This scenario will produce vulnerability because the lambda function only needs permissions to invoke an endpoint from SageMaker. So using the principle of least privilege is recommended. The corresponding policy needs to be adjusted to limit the lambda function to invoke an endpoint only. Also, the old or unused roles need to be deleted to grant more security.

The image displays two screenshots from the AWS Management Console. The top screenshot shows the IAM console interface for a role named `project4-role-ktvzixj1`. A green notification banner at the top states "Policy was successfully attached to role." The left sidebar shows the "Identity and Access Management (IAM)" menu. The main content area shows the role's summary, including its creation date (February 01, 2023) and ARN. Below the summary, the "Permissions" tab is selected, showing two attached policies: `AWSLambdaBasicExecutionRole-f2dddfcb-7d0b-4e15-bb5c-921...` (Customer managed) and `AmazonSageMakerFullAccess` (AWS managed). The bottom screenshot shows the AWS Lambda console for the function `project4`. A green notification banner states "Successfully updated the function project4." The interface includes a file editor for `lambda_function.py` and a "Test" button. The "Execution results" tab is active, showing a successful test event. The response is a JSON object with status code 200 and headers. The function logs show the execution details, including the request ID, duration, and memory usage.

**Top Screenshot: IAM Role Configuration**

**Role:** project4-role-ktvzixj1

**Summary:**

- Creation date: February 01, 2023, 10:23 (UTC+07:00)
- ARN: `arn:aws:iam:188932755969:role/service-role/project4-role-ktvzixj1`
- Last activity: None
- Maximum session duration: 1 hour

**Permissions policies (2):**

Policy name	Type	Description
<code>AWSLambdaBasicExecutionRole-f2dddfcb-7d0b-4e15-bb5c-921...</code>	Customer managed	
<code>AmazonSageMakerFullAccess</code>	AWS managed	Provides full access to Amazon SageMaker via the AW...

**Bottom Screenshot: Lambda Function Test Results**

**Function:** project4

**Test Event Name:** project4\_test

**Response:**

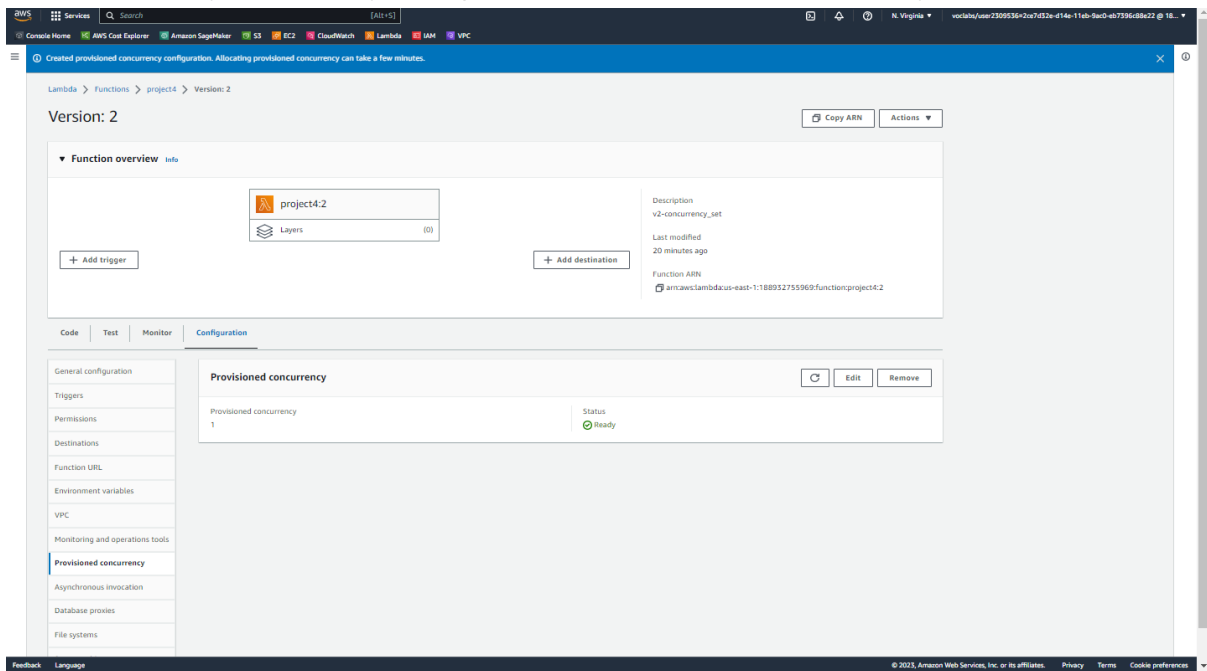
```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "Content-Type-In": "<_main_.LambdaContext object at 0x7f622028eca0>",
  "body": "[[-4.036006927490234, -2.0773777961730957, -1.5017143487930298, -0.05347501486539841, -3.226405143737793, -3.263850212097168, -0.599189698696136]]"
```

**Function Logs:**

```
Loading Lambda function
START RequestId: 8417169e-7013-4e2d-8395-84a8df6874cd Version: $LATEST
Context:: <_main_.LambdaContext object at 0x7f622028eca0>
Event type: <class 'dict'>
END RequestId: 8417169e-7013-4e2d-8395-84a8df6874cd
REPORT RequestId: 8417169e-7013-4e2d-8395-84a8df6874cd Duration: 1119.77 ms Billed Duration: 1120 ms Memory Size: 128 MB Max Memory Used: 66 MB
```

## Concurrency and Auto-scaling

The main purpose of concurrency and auto-scaling are to reduce latency during high-traffic scenarios. In this project, I just worked with low-traffic situations. So a small amount of concurrency is acceptable. I configured concurrency after creating a version of the lambda function. The only concurrency setting available is provisioned concurrency.



For the same reason as concurrency, a small amount of instances is acceptable for auto-scaling. I choose **3** for the maximum instances. It will be triggered if **10** requests come simultaneously.

### Configure variant automatic scaling

Deregister auto scaling

#### Variant automatic scaling [Learn more](#)

Variant name AllTraffic	Instance type ml.m5.large  Elastic Inference -	Current instance count 1  Current weight 1
----------------------------	--	--

Minimum instance count

1

-

Maximum instance count

3

IAM role

Amazon SageMaker uses the following service-linked role for automatic scaling. [Learn more](#)

AWSServiceRoleForApplicationAutoScaling\_SageMakerEndpoint

#### Built-in scaling policy [Learn more](#)

Policy name  
SageMakerEndpointInvocationScalingPolicy

Target metric  
[SageMakerVariantInvocationsPerInstance](#)

Target value  
10

Scale in cool down (seconds) - optional  
30

Scale out cool down (seconds) - optional  
30

☐ Disable scale in  
Select if you don't want automatic scaling to delete instances when traffic decreases. [Learn more](#)

#### Custom scaling policy [Learn more](#)

There are no custom scaling policies for this variant.

Cancel

Save