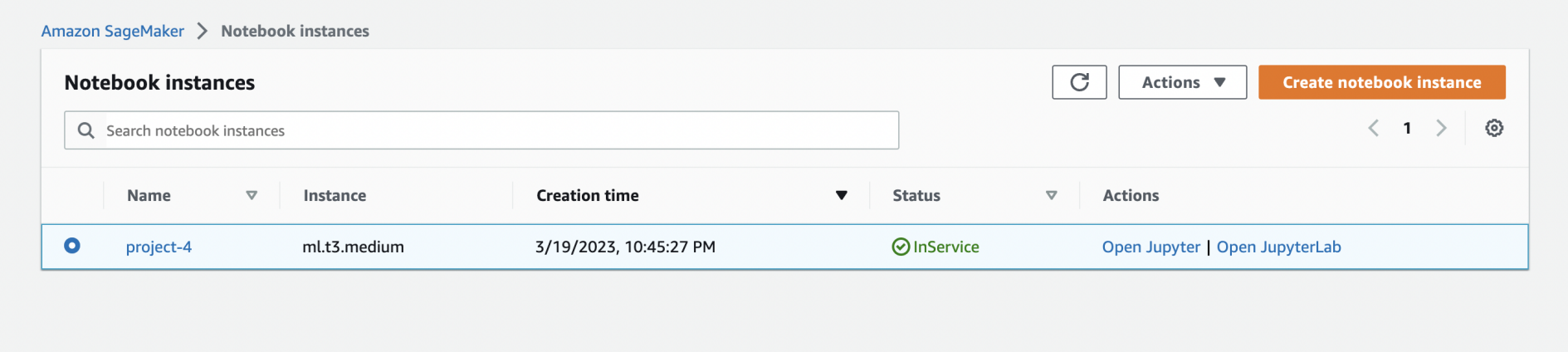
README.md

Udacity_project4

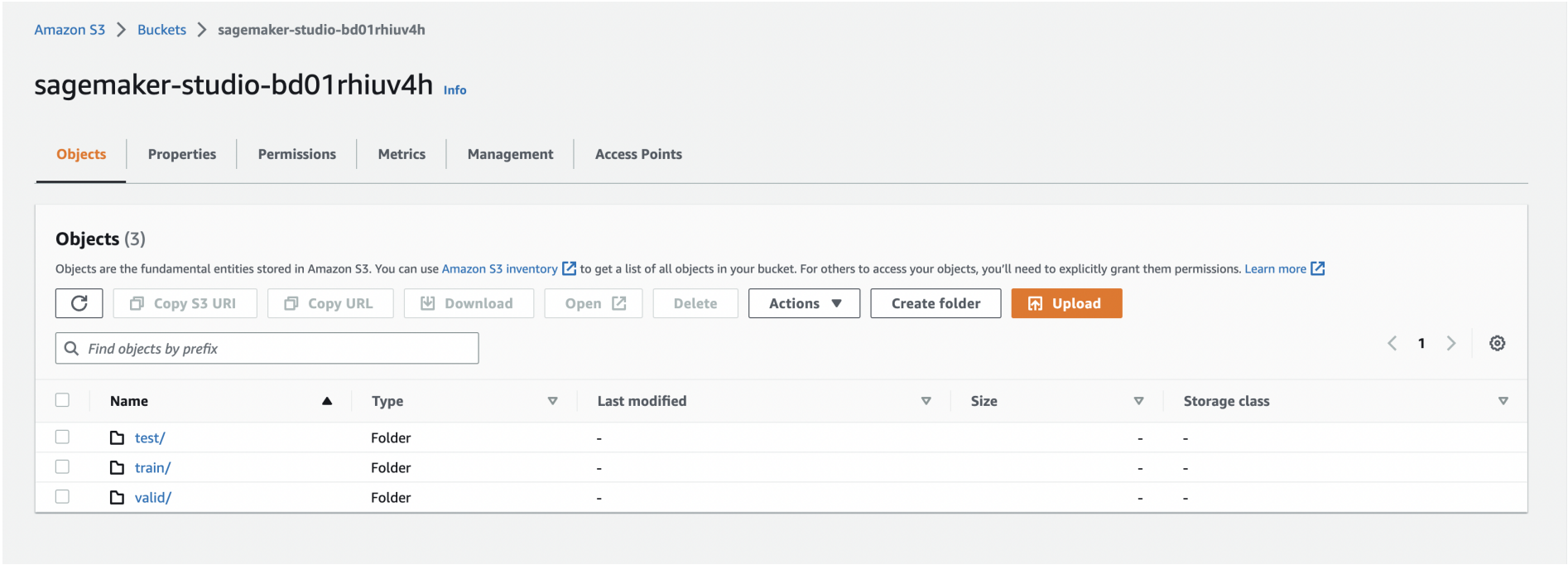
Step 1:

Used an ml.t3.medium notebook as from my previous experiences with similar projects this instance seemed sufficient with a low cost

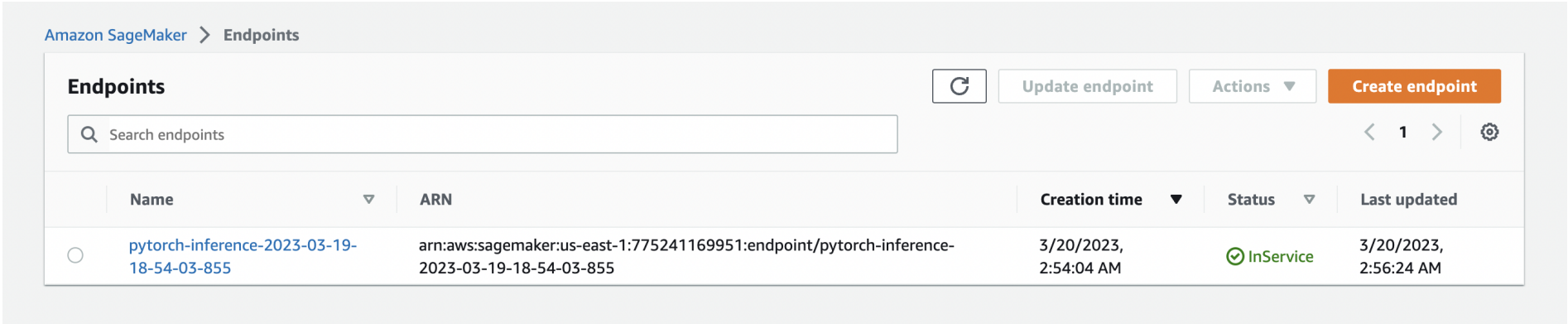
Notebook set up:



S3 Bucket used:



Deployed endpoint:



Step 2:

I chose the g5.2xlarge as it was on the list of approved instances that work with the Deep Learning AMI, I needed the pytorch env to be able to run the `solution.py` script. It was one of the cheaper options providing enough cpu and memory (32 GiB) to train the model. I played it safe and chose a model with more specs than needed since it only cost $1.212/hr and i would only be using it for 10-20 mins.

Recommended GPU Instances

PDF | RSS

We recommend a GPU instance for most deep learning purposes. Training new models is faster on a GPU instance than a CPU instance. You can scale sub-linearly when you have multi-GPU instances or if you use distributed training across many instances with GPUs. To set up distributed training, see Distributed Training.

The following instance types support the DLAMI. For information about GPU instance type options and their uses, see EC2 Instance Types and select **Accelerated Computing**.

> **Note**
> The size of your model should be a factor in selecting an instance. If your model exceeds an instance's available RAM, select a different instance type with enough memory for your application.

- Amazon EC2 P3 Instances have up to 8 NVIDIA Tesla V100 GPUs.
- Amazon EC2 P4 Instances have up to 8 NVIDIA Tesla A100 GPUs.
- Amazon EC2 G3 Instances have up to 4 NVIDIA Tesla M60 GPUs.
- Amazon EC2 G4 Instances have up to 4 NVIDIA T4 GPUs.
- Amazon EC2 G5 Instances have up to 8 NVIDIA A10G GPUs.
- Amazon EC2 G5g Instances have Arm-based AWS Graviton2 processors.

DLAMI instances provide tooling to monitor and optimize your GPU processes. For more information about monitoring your GPU processes, see GPU Monitoring and Optimization.

EC2 instance with `model.pth` saved to `TrainedModels` directory

```
BUILD_FROM_SOURCE_PACKAGES_LICENCES  PYTHON_PACKAGES_LICENSES      dogImages               solution.py
LINUX_PACKAGES_LICENSES              THIRD_PARTY_SOURCE_CODE_URLS  dogImages.zip
LINUX_PACKAGES_LIST                  TrainedModels                 nvidia-acknowledgements
(pytorch) python solution.py
/opt/conda/envs/pytorch/lib/python3.9/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretr
ained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than
 a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior
 is equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50_Weights.DEFAULT`
 to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /home/ec2-user/.cache/torch/hub/checkpoints
/resnet50-0676ba61.pth
100%|                                                                                | 97.8M/97.8M [00:00<00:00, 384MB/s]
Starting Model Training
saved
(pytorch) ls
BUILD_FROM_SOURCE_PACKAGES_LICENCES  PYTHON_PACKAGES_LICENSES      dogImages               solution.py
LINUX_PACKAGES_LICENSES              THIRD_PARTY_SOURCE_CODE_URLS  dogImages.zip
LINUX_PACKAGES_LIST                  TrainedModels                 nvidia-acknowledgements
(pytorch) cd TrainedModels/
(pytorch) ls
model.pth
(pytorch)
```

EC2 Code vs Sagemaker Code:

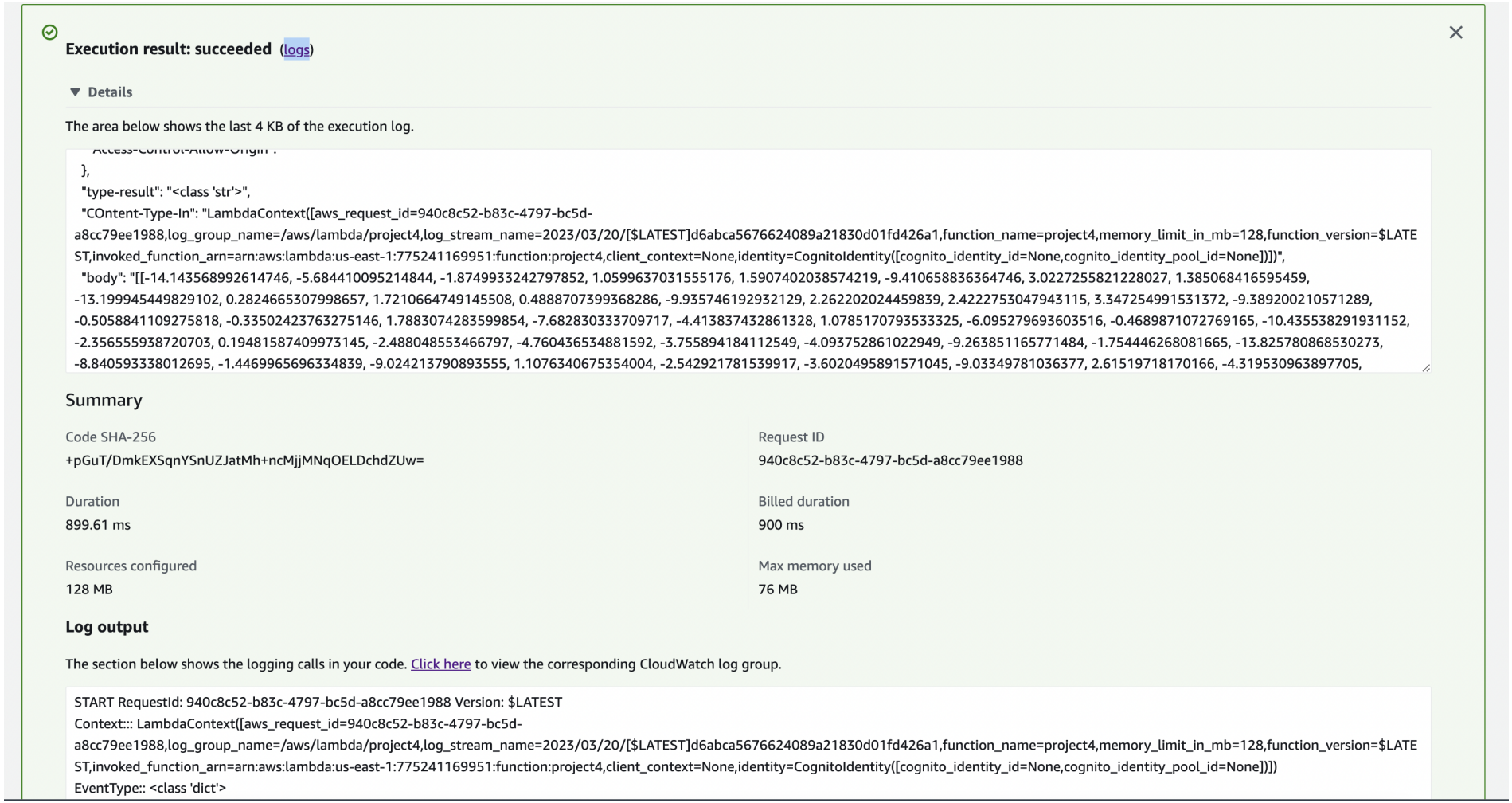
- Not a lot of hyperparameter tuning in the EC2 code compared to Sagemaker code
- Deployment method is different, one generates a model.pth file and one uses an endpoint that can be accessed publicly

Step 3:

I created a lambda function using the `lambdafunction.py` script where I replaced the `endpoint_Name` variable with my previously created endpoint `pytorch-inference-2023-03-19-18-54-03-855` . The lambda function simply calls this endpoint and returns the response. I created a test event to see if my function works correctly:

```
{ "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-s
```

Result of lambda function test:




```
Test Event Name
test

Response
{
  "statusCode": 200,
  "headers": {
```

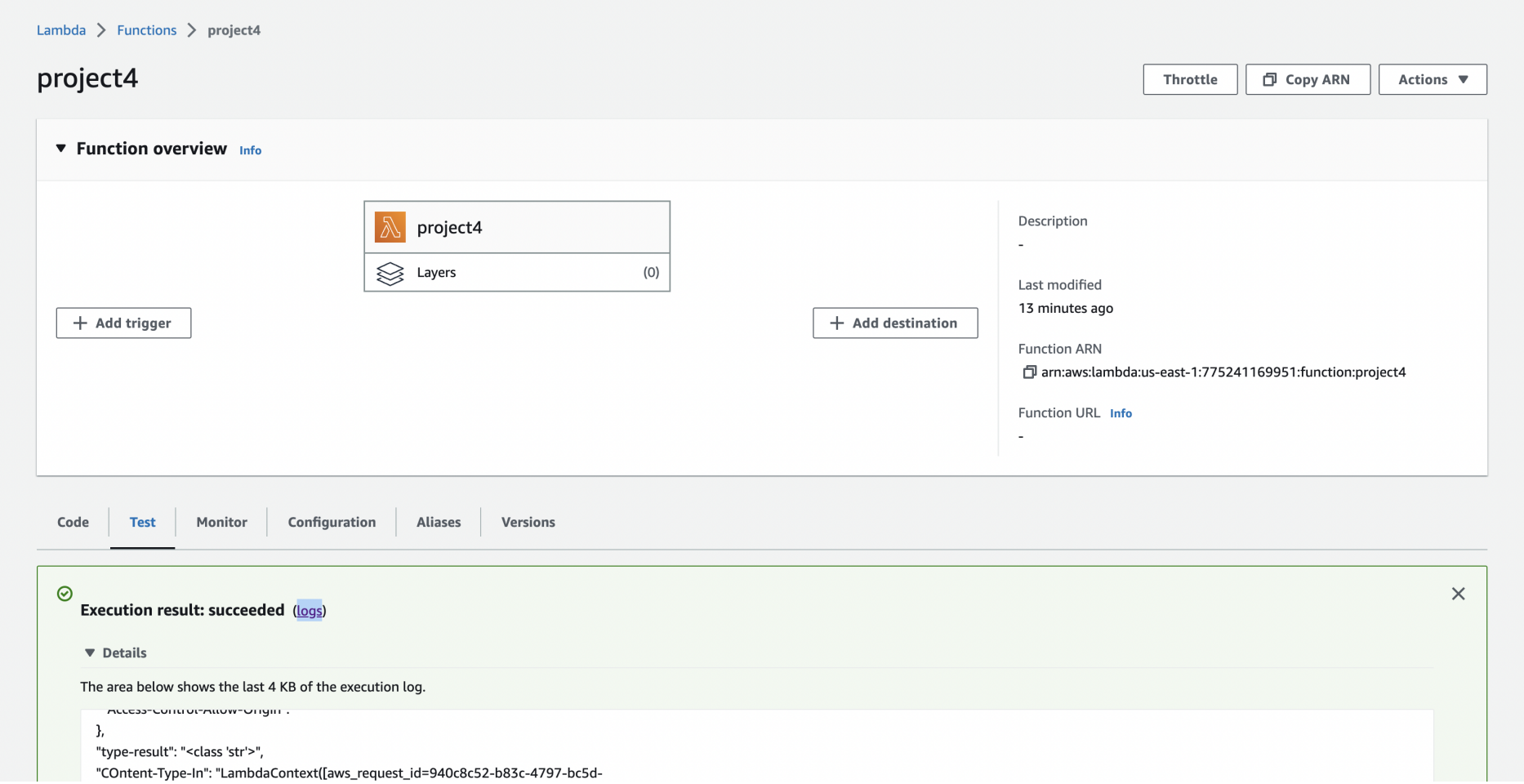
```
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=09360260-4de3-4ea2-af2e-96d067e143fa,log_group_name=/aws
  "body": "[[-14.143568992614746, -5.684410095214844, -1.8749933242797852, 1.0599637031555176, 1.59074020385
}

Function Logs
START RequestId: 09360260-4de3-4ea2-af2e-96d067e143fa Version: $LATEST
Context::: LambdaContext([aws_request_id=09360260-4de3-4ea2-af2e-96d067e143fa,log_group_name=/aws/lambda/pro
EventType:: <class 'dict'>
END RequestId: 09360260-4de3-4ea2-af2e-96d067e143fa
REPORT RequestId: 09360260-4de3-4ea2-af2e-96d067e143fa  Duration: 936.98 ms     Billed Duration: 937 ms Memo

Request ID
09360260-4de3-4ea2-af2e-96d067e143fa
```

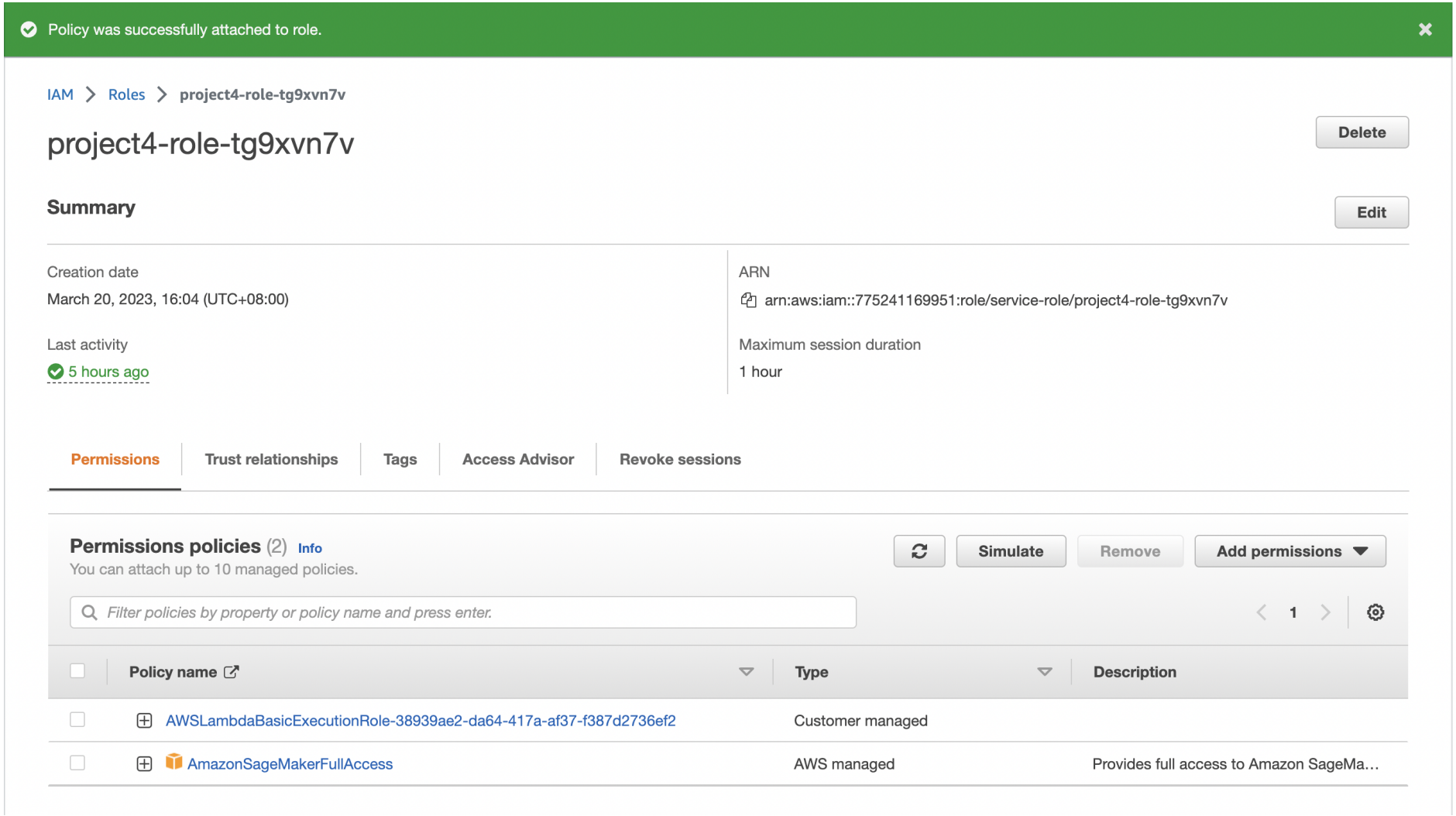
Step 4:

Lambda function setup:



Security policy:

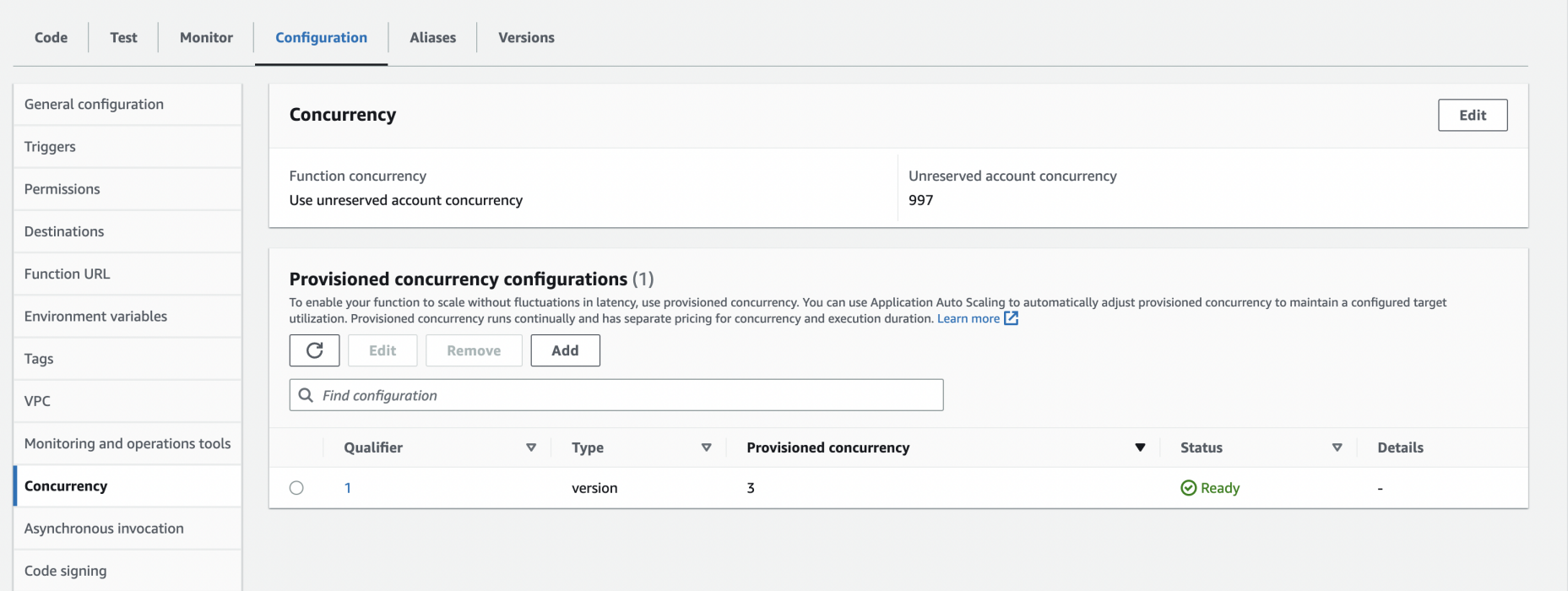
My policy has full SageMaker access, which is a lot more permissive than it needs to be. Also roles that are old or inactive can be dangerous as people who no longer work on the project can have access to the resources and can jeopardize the project with malicious intent. Likewise, roles with policies for functions that the project is no longer using can be dangerous because users can use resources to modify other unrelated resources that might affect the success and security of the project. I can make it secure by reducing the scope to my specific endpoint by naming the resource and specifying only read actions on the endpoint. I believe my AWS Workspace is relatively secure but constant monitoring of security policies across EC2, SageMaker, IAM roles, Lambda functions is required to make it as restrictive as possible.



Step 5:

Concurrency:

I set up a provisioned concurrency of 3 so that my Lambda function can handle that many more calls concurrently.



Auto-scaling

I set min instance to 1 and max instance to 5 so that it can scale accordingly. I also set the scale-in/out cool down to 60 seconds so that instances can scale in/out faster and it doesnt affect the availability of the endpoint.

Variant automatic scaling Learn more

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

Minimum instance count: 1 - Maximum instance count: 5

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: 20

Scale in cool down (seconds) - *optional*

Scale out cool down (seconds) - *optional*

Disable scale in