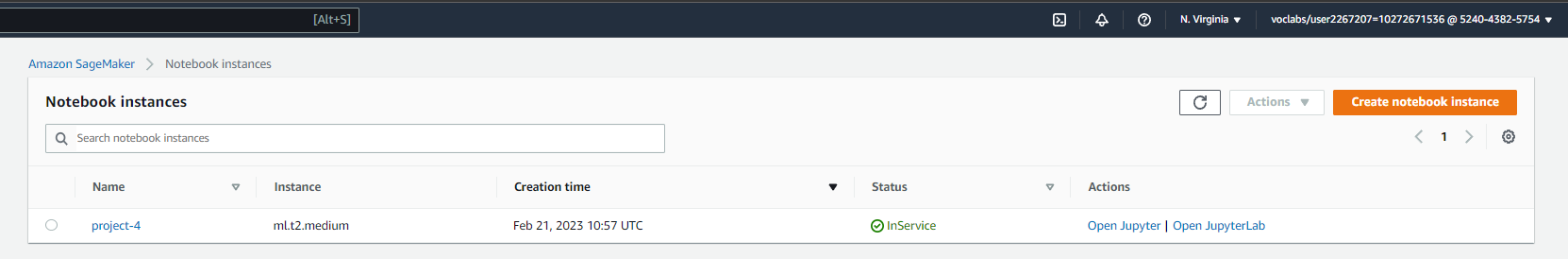
**Operationalizing an AWS ML Project**

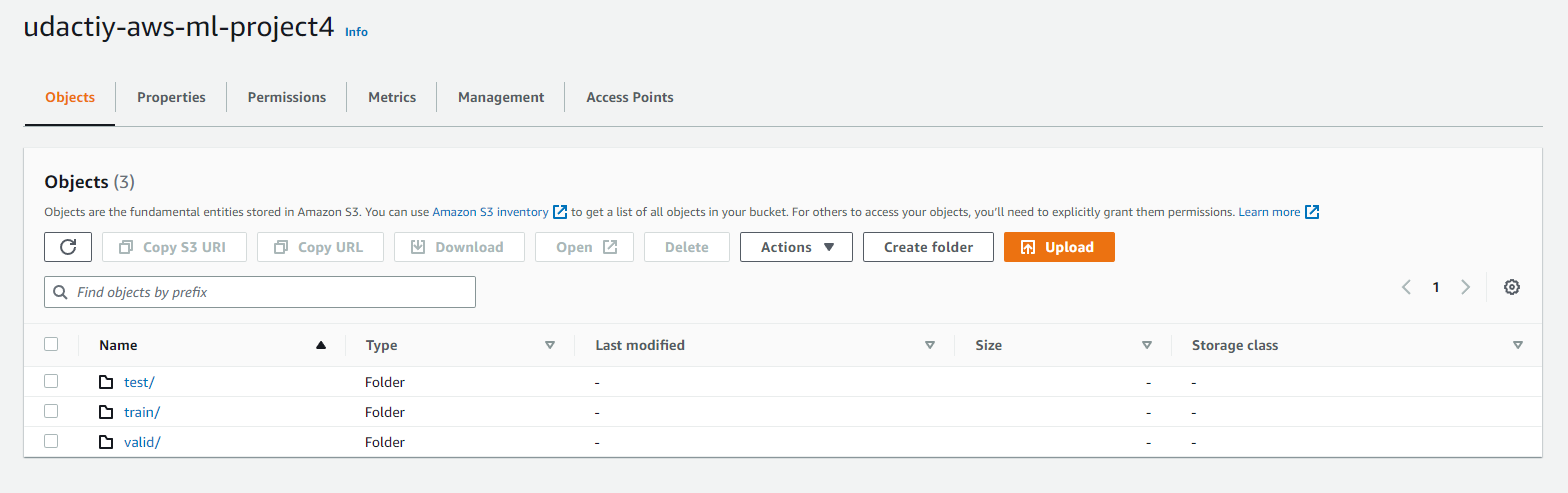
* **Initial setup**

For the Notebook instance, I've selected the "ml.t2.medium" instance type I chose this instance type for my notebook for a variety of factors.

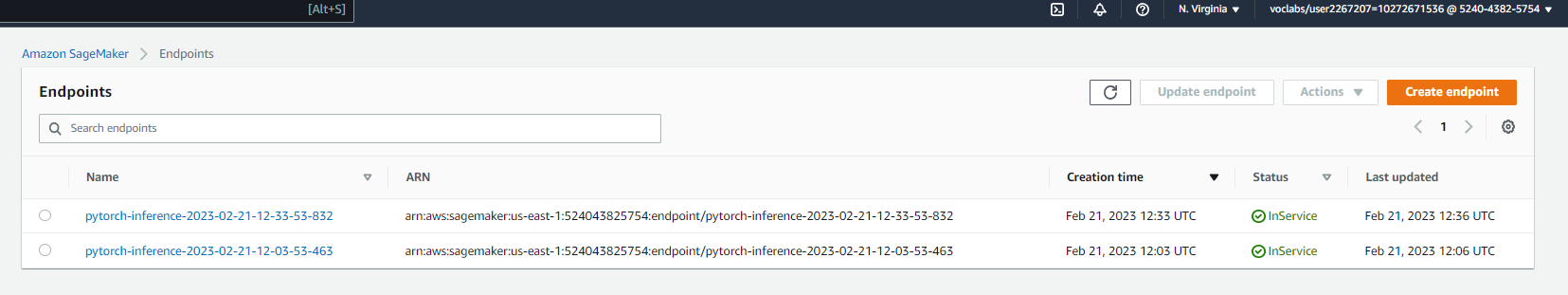
* First off, a powerful CPU and lots of RAM are not necessary for finishing the Jupyter notebooks for this project.
* When working on the project, we will need to keep this notebook instance in active status for a considerable amount of time.
* We should choose a notebook with a cheap cost per hour and decent CPU and RAM in order to prevent high charges.
* We can use the "ml.t2.medium" because it gives the same computational power and is less expensive per hour, given that we do have a crucial need for a quick startup time.



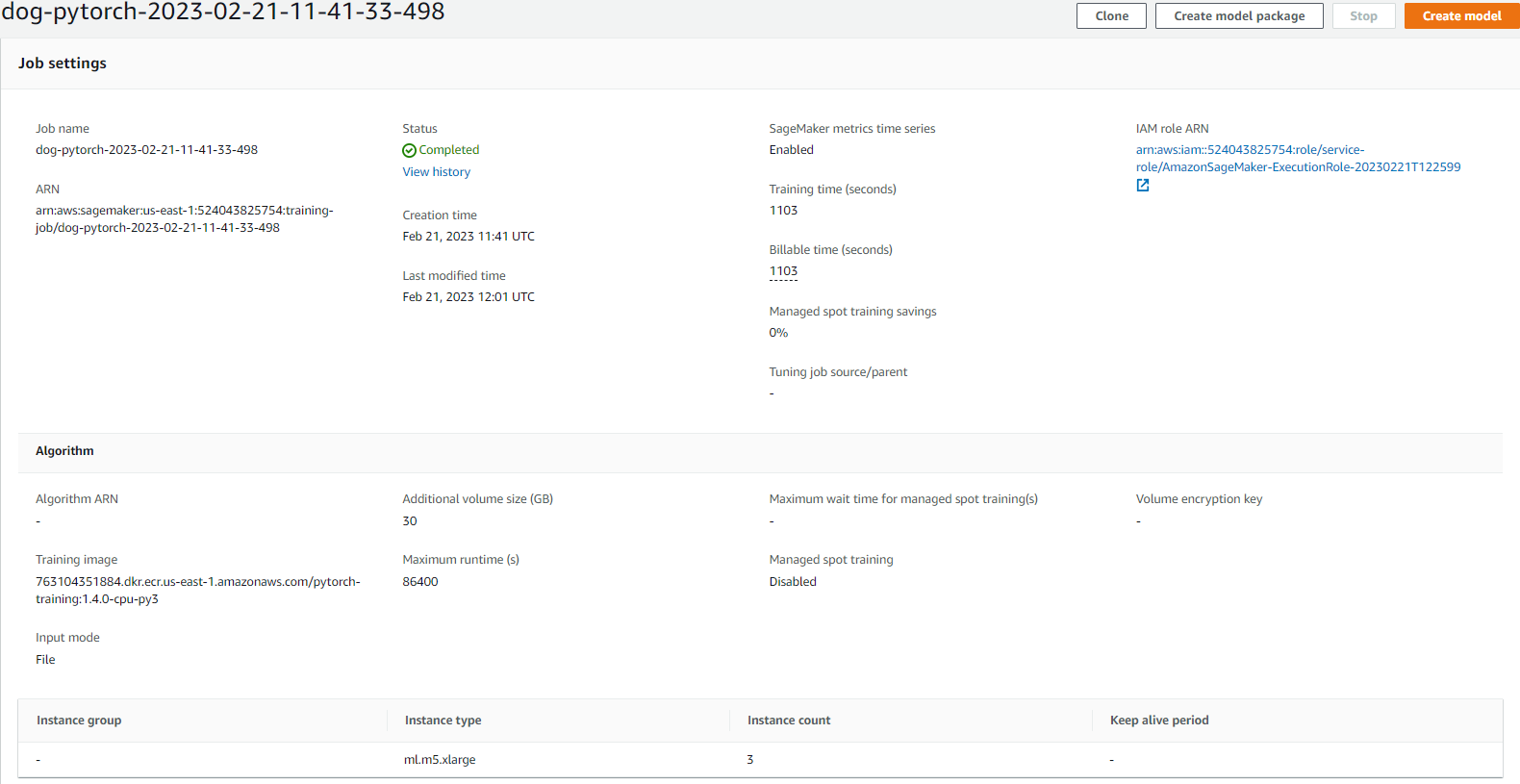
* **S3 bucket**



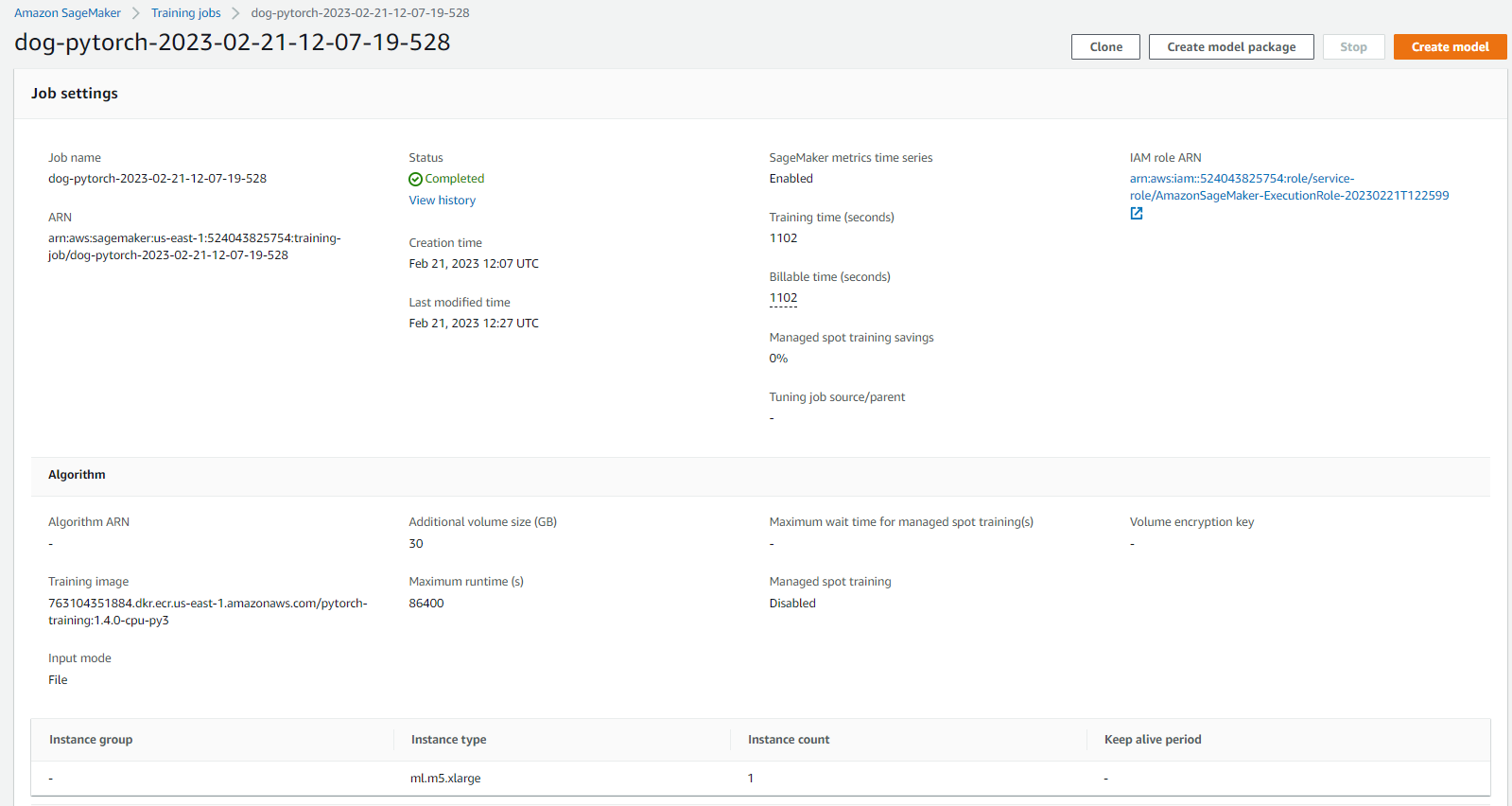
* **Endpoints**
* **Single instance “**[pytorch-inference-2023-02-21-12-33-53-832](https://us-east-1.console.aws.amazon.com/sagemaker/home?region=us-east-1#/endpoints/pytorch-inference-2023-02-21-12-33-53-832)”
* **Multi instance “**[pytorch-inference-2023-02-21-12-03-53-463](https://us-east-1.console.aws.amazon.com/sagemaker/home?region=us-east-1#/endpoints/pytorch-inference-2023-02-21-12-03-53-463)”



* **Multi Instance Training Job**



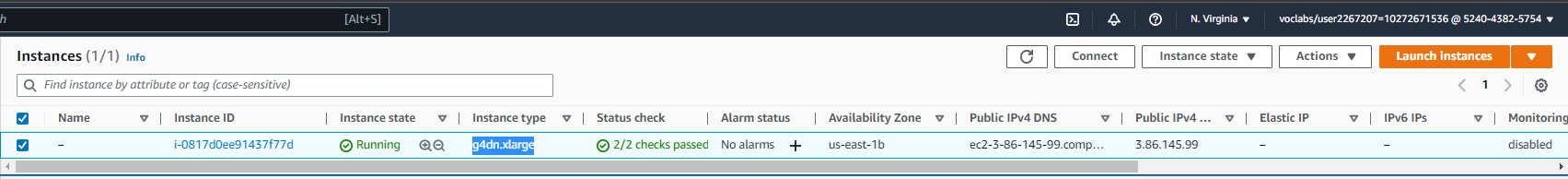
* **Single Instance Training Job**

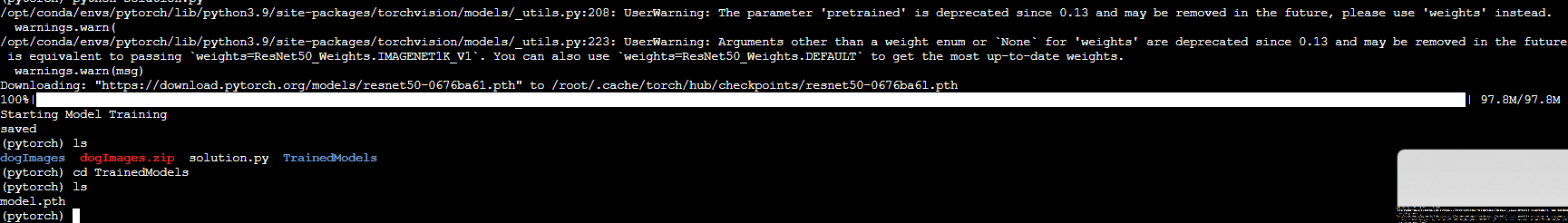


* **EC2 Setup**

We used the Deep Learning AMI and the g4dn.xlarge instance. This seems like a reasonable compromise between budget and performance.

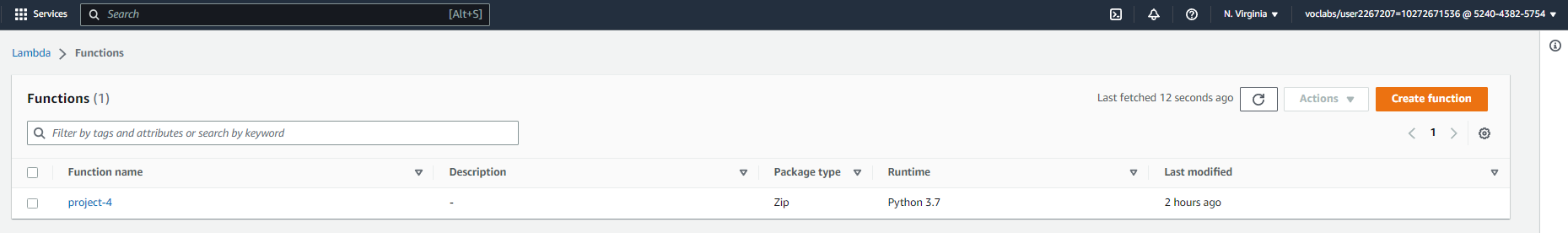
* First off, G4DN.Xlarge instances are the most cost-effective for this project, as in general, g4dn is the cheapest instance that supports Deep Learning AMI.
* G4DN.Xlarge instances can maintain high CPU performance for as long as a task requires it, according to the documentation.
* g4dn instances will deliver sufficient performance for the majority of general-purpose applications without any additional fees.

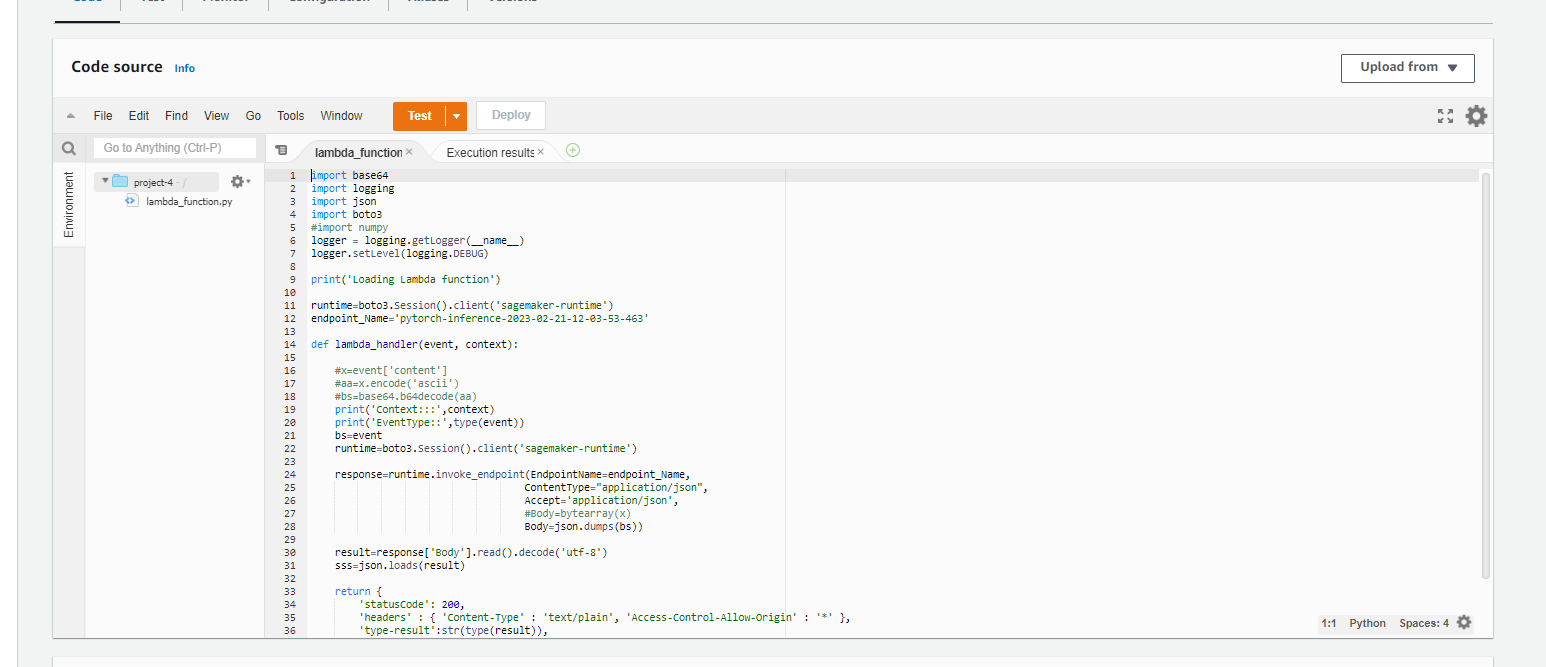




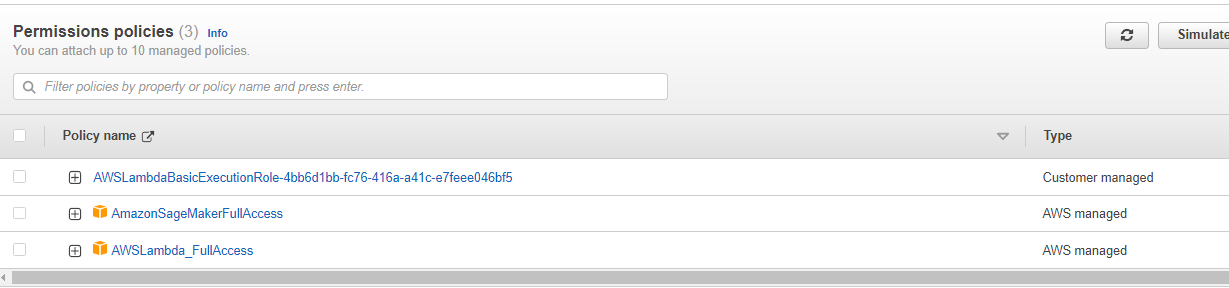
Sagemaker's code is fairly similar to that for EC2 instances, but the code for EC2 instances should be a self-contained Python script that trains the model on any machine using a local file system to access the data, as opposed to Sagemaker scripts that require methods to access S3 datasets. Using an EC2 instance is much more comparable to using your own computer.

* **Lambda Function**





* **IAM**



Since I just have one endpoint and one lambda, adding the full access role to the lambda is acceptable for the personal application. Nevertheless, in general, permissions should be managed and likely have a due date.

* The "Full Access" type permission policies that are present worry me.

For this lambda function, for instance, we have given it complete access to SageMaker resources, but this appears to violate the principle of least privilege access.

Ideally, just the endpoints that these lambda functions are authorised to query should be made available to them.

In order to determine whether there is anything we can do about it, we will need to conduct further study.

* **Lambda output**

"body": "[[-0.04420359060168266, 0.12962570786476135, -0.1439894586801529, 0.2965732216835022, 0.24435144662857056, 0.6687325835227966, -0.04147570580244064, -0.08199003338813782, -0.24063050746917725, 0.021341335028409958, 0.15183500945568085, 0.2998291254043579, -0.5228815674781799, 0.26981258392333984, 0.37708455324172974, 0.15939001739025116, 0.19085007905960083, 0.2331576645374298, -0.1019330620765686, 0.3789478838443756, 0.14189301431179047, -0.4271489679813385, 0.6616247892379761, -0.17211994528770447, 0.17868740856647491, -0.27528780698776245, 0.4714634120464325, -0.3745245933532715, 0.7525752186775208, 0.10917049646377563, 0.26970112323760986, 0.6853075623512268, 0.05942630022764206, 0.2536903917789459, 0.6250325441360474, 0.19215542078018188, 0.5037903785705566, 0.47488102316856384, 0.11332790553569794, -0.03736889734864235, -0.20172210037708282, 0.0015439391136169434, 0.3047942519187927, 0.2173231840133667, 0.03885681554675102, 0.29188448190689087, -0.20934823155403137, 0.13540184497833252, 0.037915512919425964, 0.01764487288892269, 0.25828254222869873, 0.6624497175216675, -0.2939926087856293, 0.09863366186618805, 0.2360335886478424, 0.3429732024669647, 0.16470471024513245, -0.10818889737129211, -0.18856334686279297, -0.19594715535640717, 0.1277601718902588, 0.17702533304691315, -0.0808212086558342, 0.11317931115627289, -0.12674687802791595, -0.7392841577529907, -0.23281840980052948, 0.2800247073173523, 0.3573417663574219, 0.26963672041893005, 0.11836199462413788, 0.16226693987846375, 0.04022480547428131, -0.14020581543445587, 0.12572070956230164, 0.12625442445278168, -0.13040825724601746, -0.027632055804133415, 0.26364296674728394, 0.034050021320581436, 0.48541292548179626, -0.1459079533815384, -0.06378944963216782, 0.08936718106269836, -0.020962441340088844, 0.35300004482269287, 0.24700608849525452, 0.060690417885780334, 0.4935251772403717, -0.16043977439403534, -0.08462701737880707, -0.016669314354658127, -0.3167859613895416, -0.15092653036117554, 0.4003157317638397, -0.10594890266656876, 0.21920444071292877, 0.07167436182498932, -0.4334968328475952, -0.2215501219034195, 0.1597968190908432, -0.03970000892877579, 0.32542523741722107, -0.10683713853359222, -0.5525974035263062, -0.04562503471970558, 0.039156224578619, -0.6656187176704407, -0.31180039048194885, -0.31561046838760376, -0.29647356271743774, -0.049371298402547836, -0.5093329548835754, -0.14106673002243042, -0.21339520812034607, -0.32145652174949646, -0.024381021037697792, -0.28888702392578125, -0.3814198672771454, -0.30282872915267944, -0.6610556244850159, -0.6459407806396484, 0.3232356607913971, 0.23819047212600708, -0.04107255861163139, -0.19141234457492828, -0.5061095952987671, -0.6881840229034424, 0.19804683327674866, -0.24417461454868317, -0.9520236849784851, -0.6306439638137817, -0.5195256471633911]]"

}

* **AutoScaling**

Because the expected endpoint load for this project is zero (I will delete it after the submission), I set a minimum accepted value of two concurrent lambda and endpoint runs.