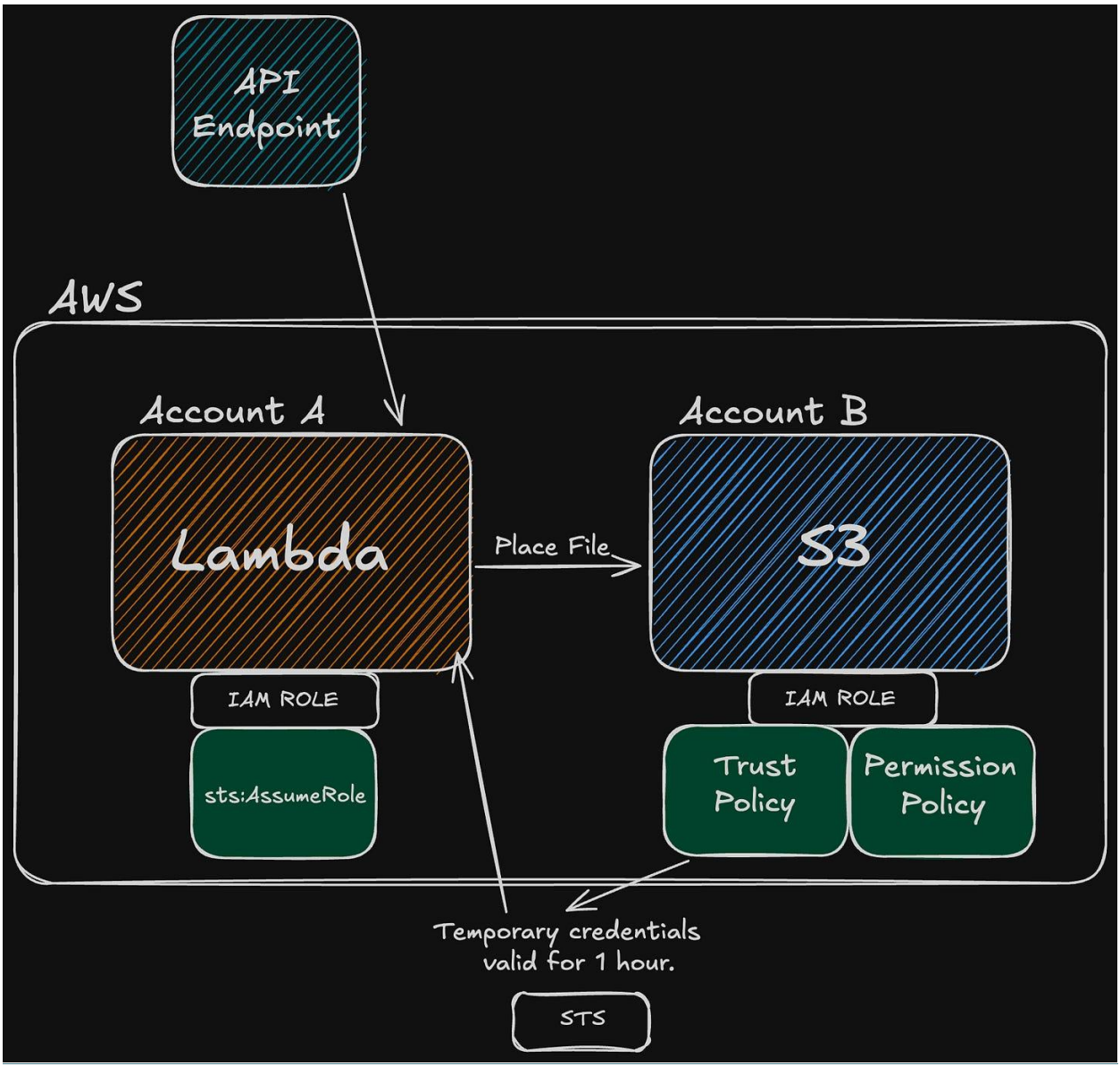
AWS STS

What is it and why is it so important? Okay, basically at a high level, AWS STS lets you request temporary credentials so a service in one account can safely take action another account.

Cross account access gets messy fast if you don't structure it properly, so here's the secure version of how it should work and what we are going to build today.



Account A
Runs something like:

- Lambda
- ECS task

It has a single IAM policy that says: *I am allowed to call sts:AssumeRole on a role in*

Account B
Contains the actual IAM role. This role has two things:

A trust policy that says: *I trust Account A to assume me.*

A permission policy that says: *Here's what I can do in Account B (example: write to S3, deploy to ECS, read Secrets Manager etc).*

Done.

Account B exposes a role.
Account A gets permission to assume it.

This is cool because, it means no permanent credentials and no giant list of access keys floating around.

Project Time

Okay, now you understand what's going on here. Let's build it out in action!

As usual, I reserve the Projects for community members... come join the fun!

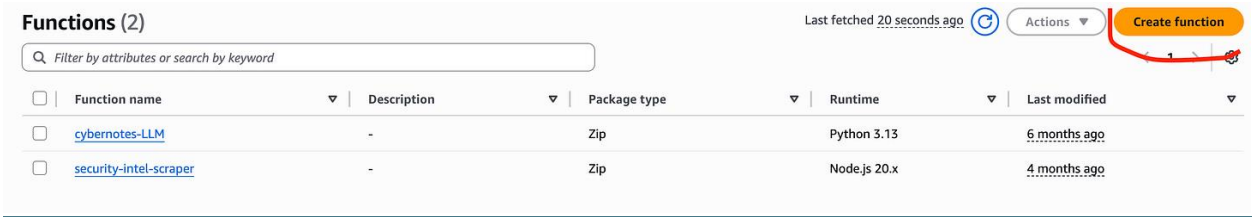
If you've made it this far, thank you. Your support genuinely means a lot. And if you're not completely satisfied with this issue, please hit reply and tell me what I can improve. I always want to make this newsletter more useful for you.

Heads up: This project won't fit in your inbox. Click through to view it.

We are going to deploy a Lambda, using Python to call an API Endpoint that tracks the location of the ISS, then dump the JSON results in S3 in another AWS Account.

Step One

Sign into your AWS Account and create a new Lambda. You could deploy this via Terraform. I'm going to use the console since it's best for visual walkthroughs.



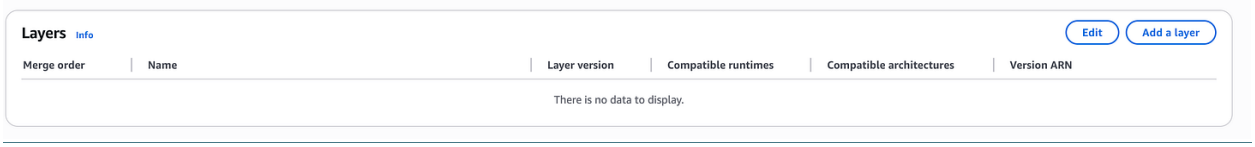
I'm using Python here, but use whatever you're comfortable with. When prompted, create a new role here:

Step Two

For this script to work we need the requests library.

We could add the requests library here, so let's quickly do that. In case you didn't know, libraries and external packages need to be separately uploaded to your Lambda as a "layer".

We are actually going to use the built in boto3 URL fetch method. However, I thought it would be cool to quickly show you how to upload a Lambda layer as it's a good skill to have.



On your local device, open a terminal and follow these steps:

mkdir layers

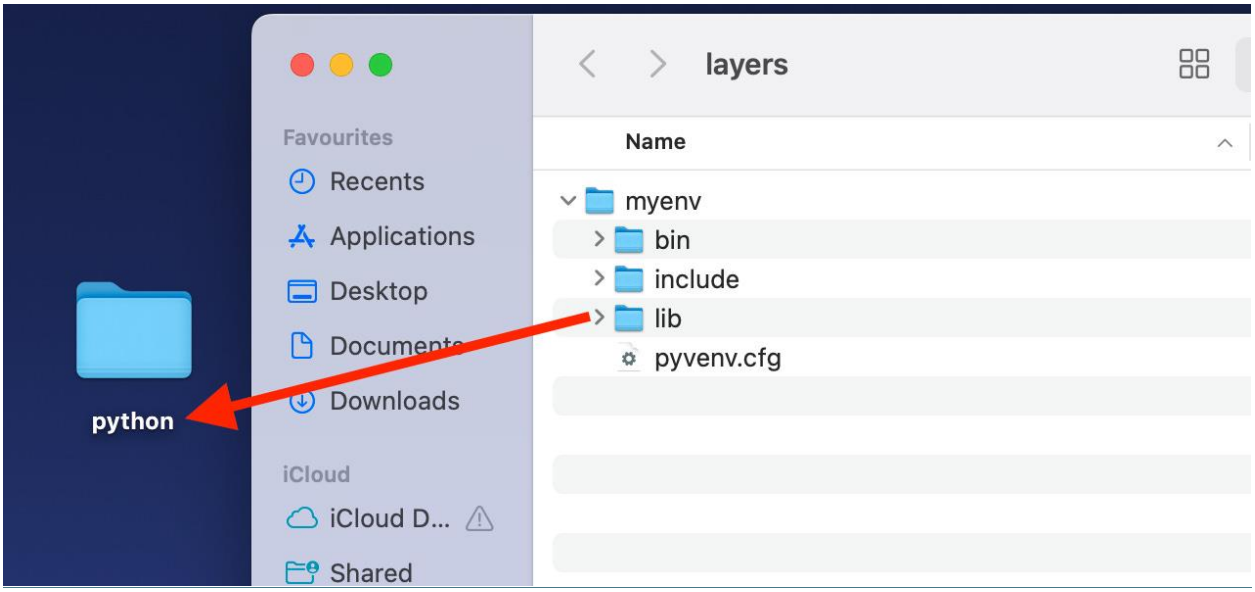
cd layers

python3 -m venv myenv

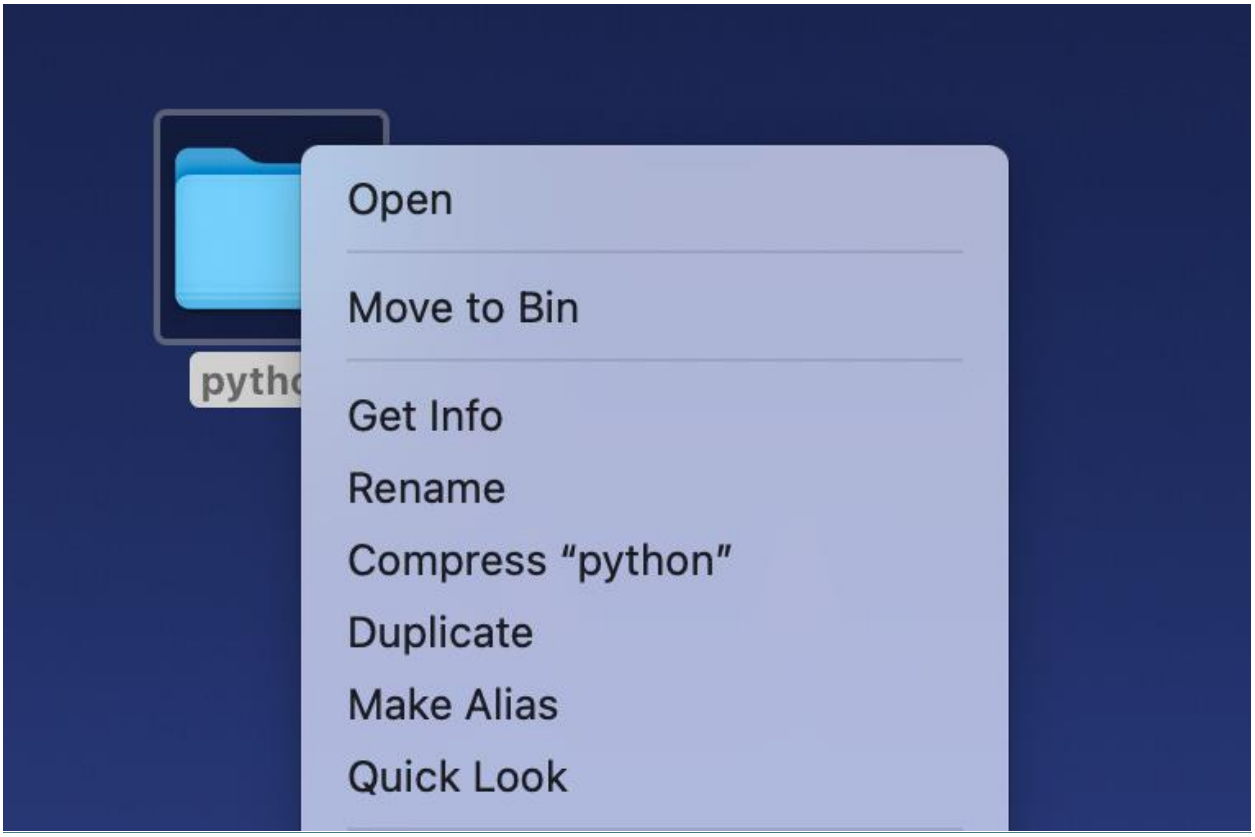
source myenv/bin/activate

pip3 install requests

Open this in your finder, create a new folder called python and drag the lib folder over

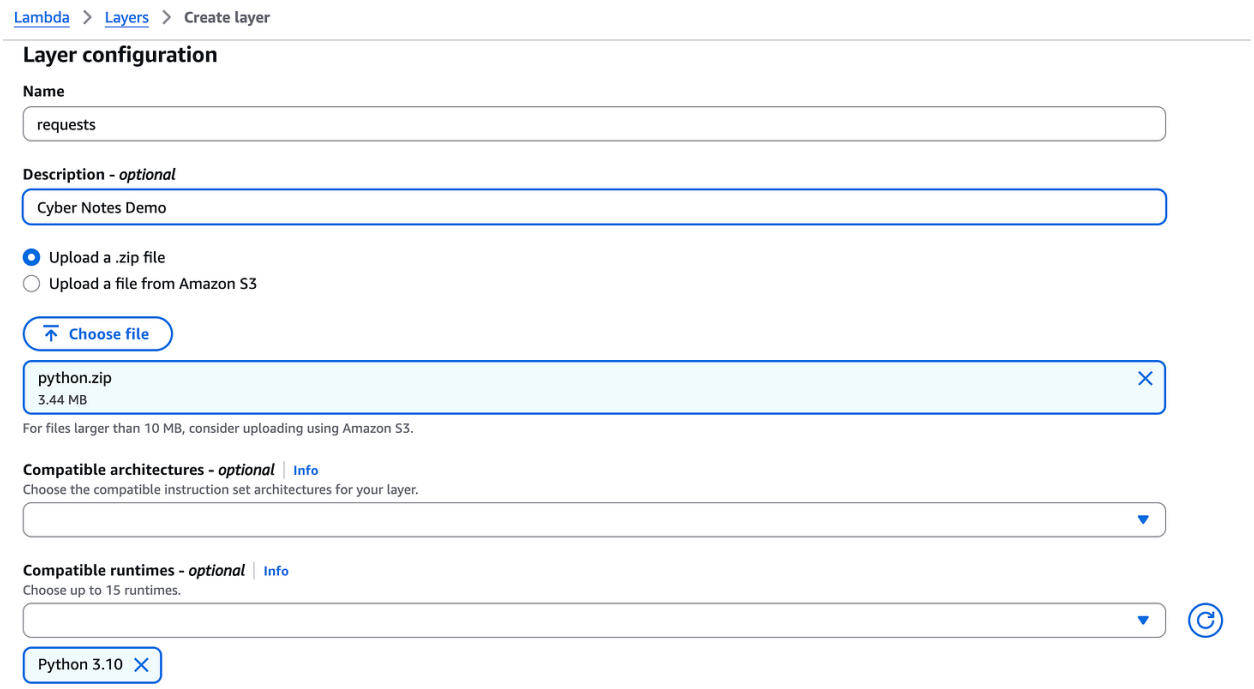


Compress this file

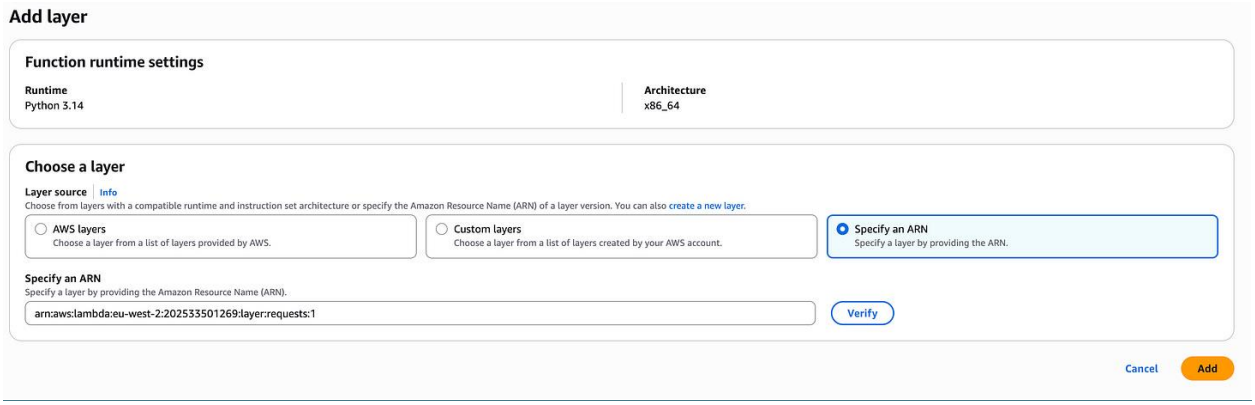


We can now upload it to our lambda.

Back in the main Lambda page, click into layers and select create layer.



Give it a suitable name and upload your compressed Python file, copy the ARN here, you'll need it next.

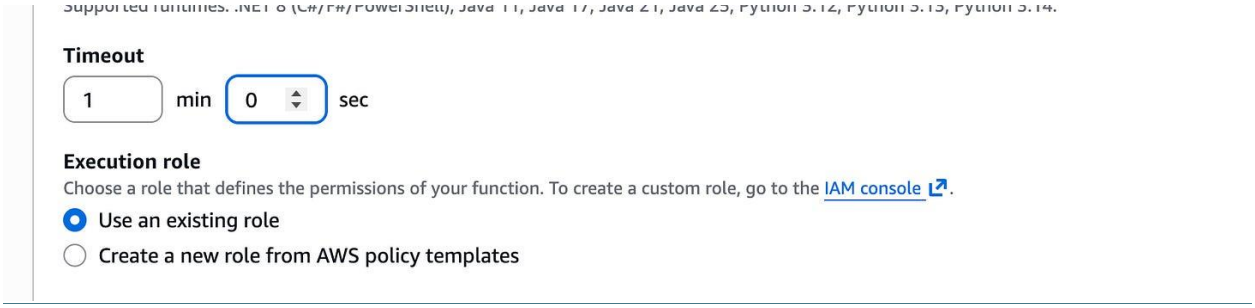


Go back to your Lambda, add layer and paste your ARN into the box.

Your script would now work using...

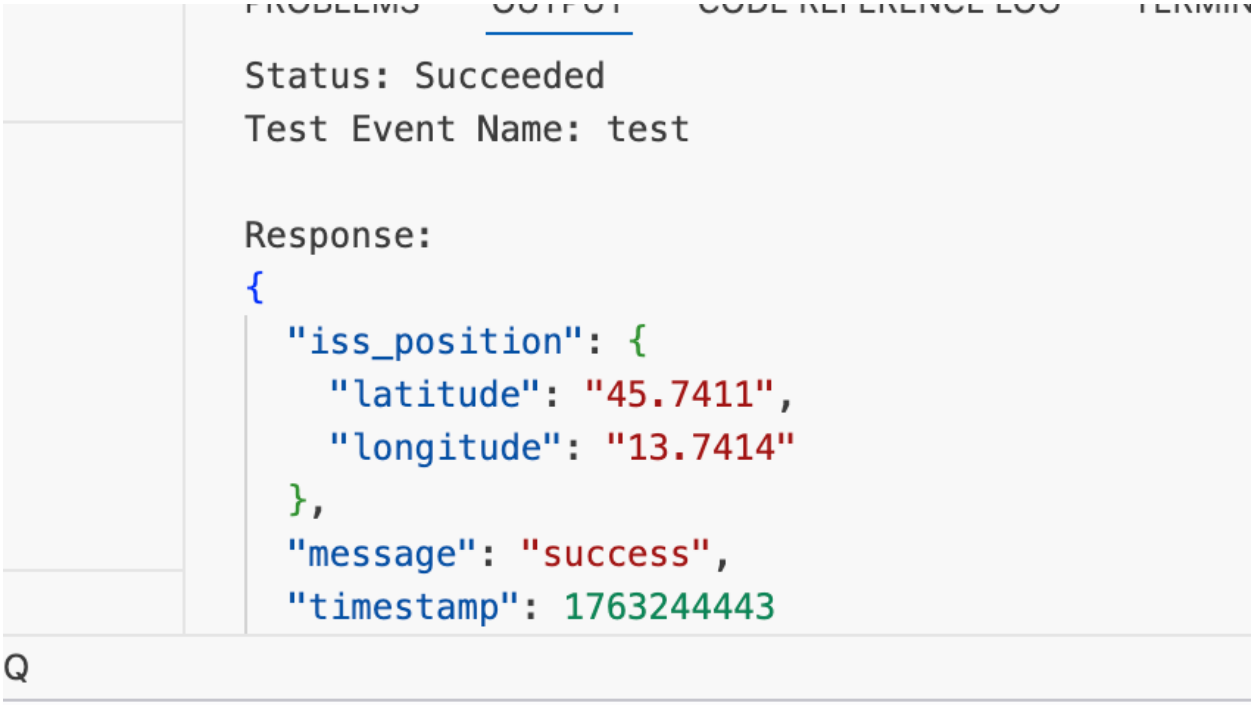
import requests

Layers side track out of the way, change the timeout of your Lambda to 1 min and paste this very small script in.




```
import urllib.request
import json

def lambda_handler(event, context):
    with urllib.request.urlopen("http://api.open-notify.org/iss-now.json") as r:
        data = json.load(r)
        return data
```

Deploy and run:

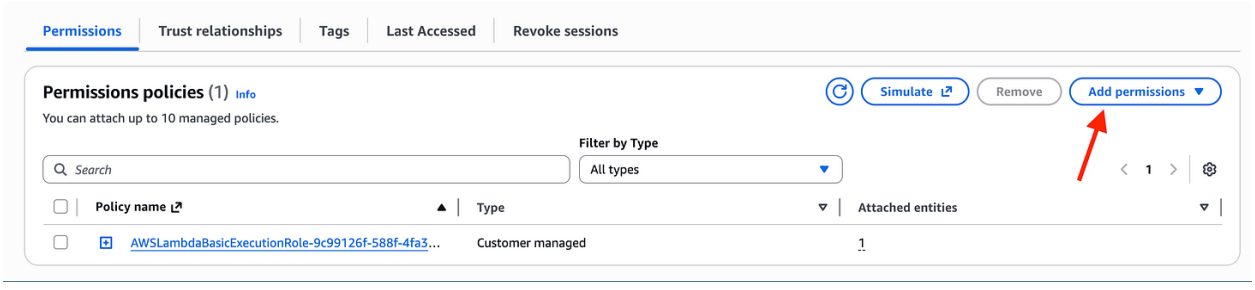


Nice! That's working but we want this Lambda to take the results here and dump them into another AWS account S3 bucket. To do that it needs the permissions we spoke about earlier.

Step Three

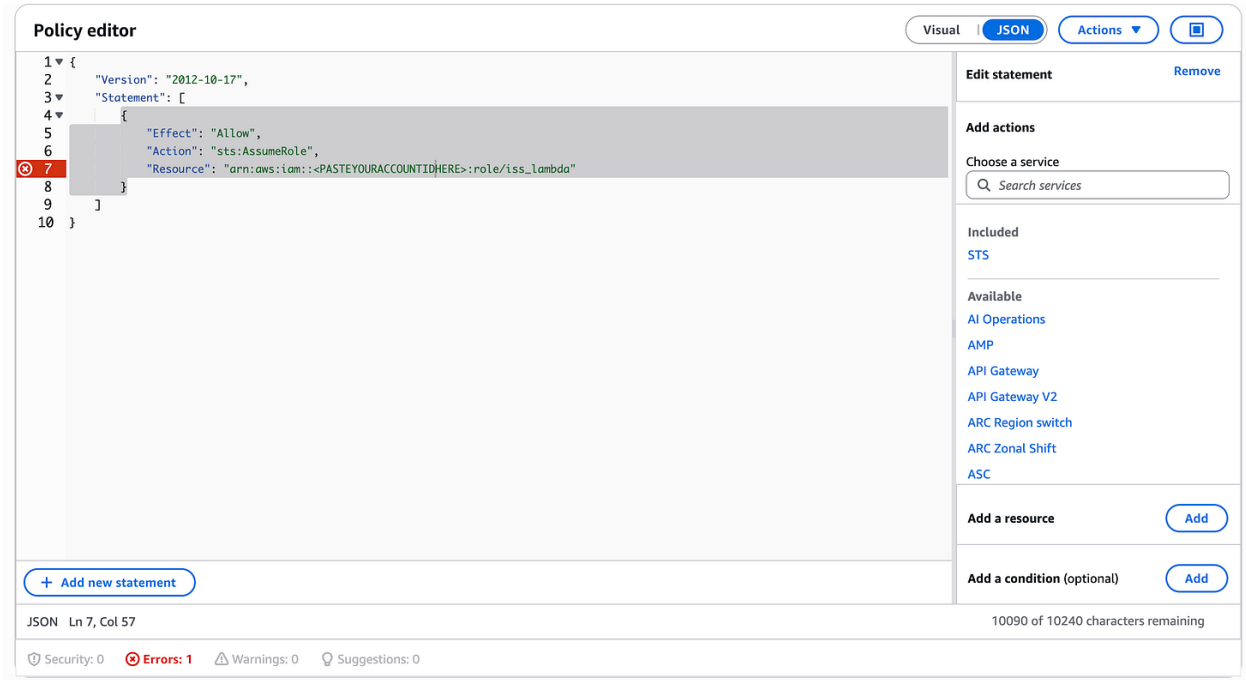
Time to configure the roles and policies.

Open the role of your Lambda and attach a policy that gives it sts:AssumeRole into your second AWS Account. Like this:



Just quickly, if you don't have a second AWS account you should make one here. I'm not going going to go through that here but it's pretty straight forward and can be done under AWS Organisations, you'll just need a second email address also.

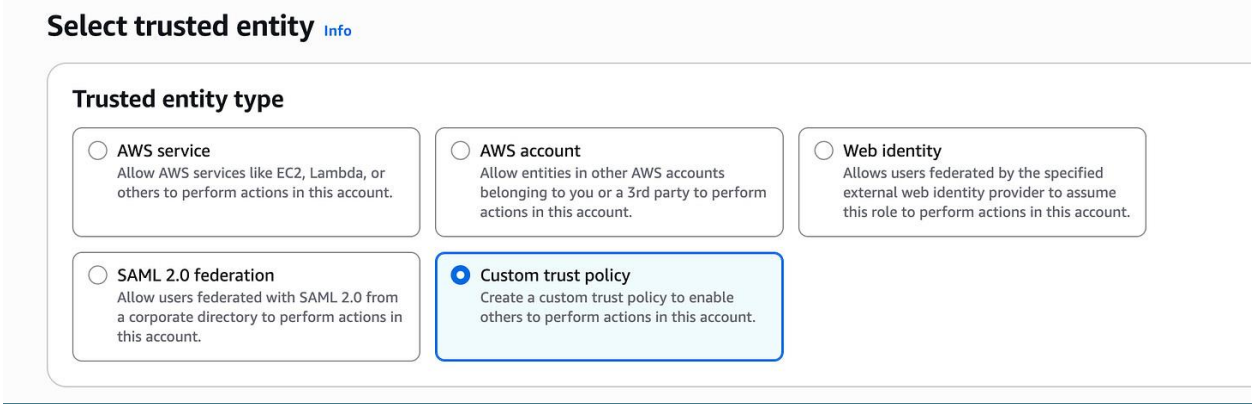
Create an inline policy



You'll want to give it the permissions here to assume the role we're going to deploy into your second account with the name and ID as the resources.

Log out of this account and log into your second new account.

Create a new IAM role and select "Custom Trust Policy".



Paste this in to allow Account A to assume the role here.

```
{
  “Version”: “2012-10-17”,
  “Statement”: [
    {
      “Effect”: “Allow”,
      “Principal”: {
        “AWS”: “arn:aws:iam::<ACCOUNT_A_ID>:root”
      },
      “Action”: “sts:AssumeRole”
    }
  ]
}
```

Attach the permissions your Lambda will need to **PUT** a file inside the S3 bucket.

```
{
  “Version”: “2012-10-17”,
  “Statement”: [
    {
```

```
        "Effect": "Allow",
        "Action": [
          "s3:PutObject",
          "s3:PutObjectAcl"
        ],
        "Resource": "arn:aws:s3:::YOUR_BUCKET_NAME/*"
      }
    ]
}
```

Paste this in to allow Account A to assume the role here.

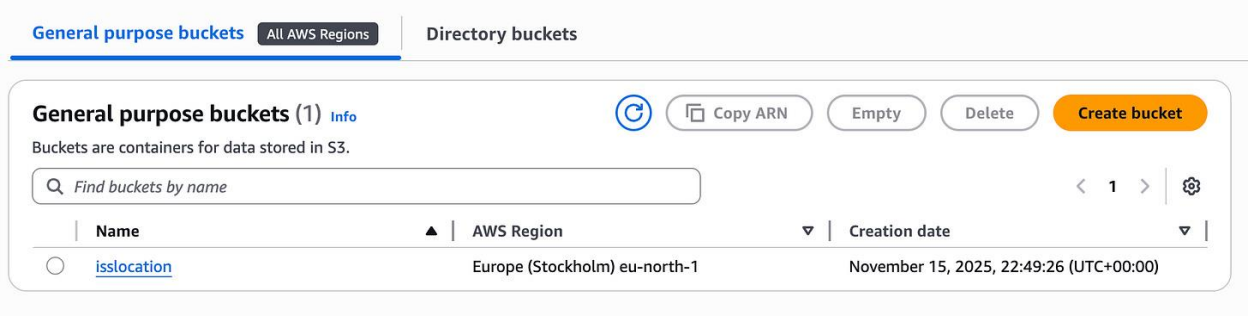
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::<ACCOUNT_A_ID>:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Attach the permissions your Lambda will need to **PUT** a file inside the S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::YOUR_BUCKET_NAME/*"
        }
    ]
}
```

We can update this in a second when we've made our bucket.

Navigate to S3, create a bucket with the basic configuration, note the name and update your role:



Step Four

Finally, let's go back to our Account A and update our script to assume the role in Account B and dump the JSON into the bucket we just made.

```
import urllib.request

import json

import boto3

from datetime import datetime


# Config

ROLE_ARN = "arn:aws:iam::<ACCOUNTB>:role/iss_lambda"

BUCKET_NAME = "isslocation"


def lambda_handler(event, context):

  # 1. Fetch ISS JSON

  with urllib.request.urlopen("http://api.open-notify.org/iss-now.json") as r:

    data = json.load(r)
```

```
# 2. Assume role in Account B
sts_client = boto3.client(”sts”)
assumed_role = sts_client.assume_role(
  RoleArn=ROLE_ARN,
  RoleSessionName=”LambdaToS3Session”
)

credentials = assumed_role[’Credentials’]

# 3. Create S3 client using assumed role credentials
s3_client = boto3.client(
  “s3”,
  aws_access_key_id=credentials[’AccessKeyId’],
  aws_secret_access_key=credentials[’SecretAccessKey’],
  aws_session_token=credentials[’SessionToken’]
)

# 4. Generate a filename based on timestamp
timestamp = datetime.utcnow().strftime(”%Y-%m-%dT%H-%M-%SZ”)
s3_key = f”iss-location-{timestamp}.json”

# 5. Upload to S3
s3_client.put_object(
  Bucket=BUCKET_NAME,
  Key=s3_key,
```

```
    Body=json.dumps(data),
    ContentType="application/json"
)

return {"status": "success", "s3_key": s3_key}
```

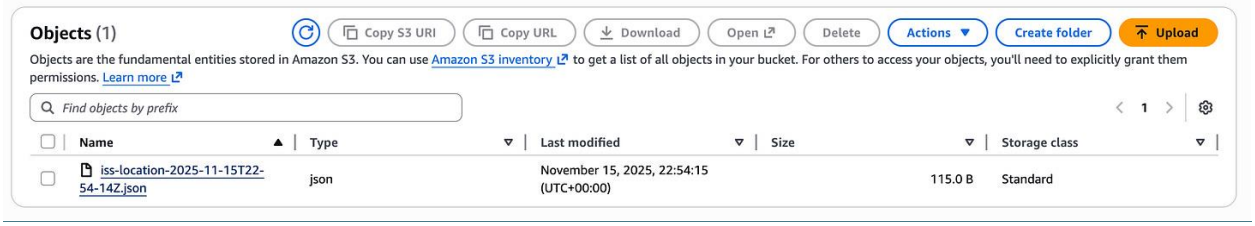
Finally, let's go back to our Account A and update our script to assume the role in Account B and dump the JSON into the bucket we just made.

```
import urllib.request
import json
import boto3
from datetime import datetime

# Config
ROLE_ARN = "arn:aws:iam::<ACCOUNTB>:role/iss_lambda"
BUCKET_NAME = "isslocation"

def lambda_handler(event, context):
    # 1. Fetch ISS JSON
    with urllib.request.urlopen("http://api.open-notify.org/iss-now.json") as
r:
        data = json.load(r)

    # 2. Assume role in Account B
    sts_client = boto3.client("sts")
    assumed_role = sts_client.assume_role(
        RoleArn=ROLE_ARN,
        RoleSessionName="LambdaToS3Session"
    )

    credentials = assumed_role['Credentials']

    # 3. Create S3 client using assumed role credentials
    s3_client = boto3.client(
        "s3",
        aws_access_key_id=credentials['AccessKeyId'],
        aws_secret_access_key=credentials['SecretAccessKey'],
        aws_session_token=credentials['SessionToken']
    )

    # 4. Generate a filename based on timestamp
    timestamp = datetime.utcnow().strftime("%Y-%m-%dT%H-%M-%SZ")
    s3_key = f"iss-location-{timestamp}.json"

    # 5. Upload to S3
    s3_client.put_object(
        Bucket=BUCKET_NAME,
        Key=s3_key,
        Body=json.dumps(data),
        ContentType="application/json"
    )

    return {"status": "success", "s3_key": s3_key}
```

Test the script

```
PROBLEMS    OUTPUT    CODE REFERENCE LOG    TERMINAL
Status: Succeeded
Test Event Name: test

Response:
{
  "status": "success",
  "s3_key": "iss-location-2025-11-15T22-54-14Z.json"
}
```

Nice! Let’s go check the bucket:



Congrats, you’ve just learnt one of the most important Cloud Security concepts! 🥕

Remember this is really helpful because you don’t necessarily need to own both the accounts you’re setting this up with. What if you’re using a product that’s hosted in an external AWS account not owned by you? You should never give the vendor access keys and you decide exactly what actions they get.