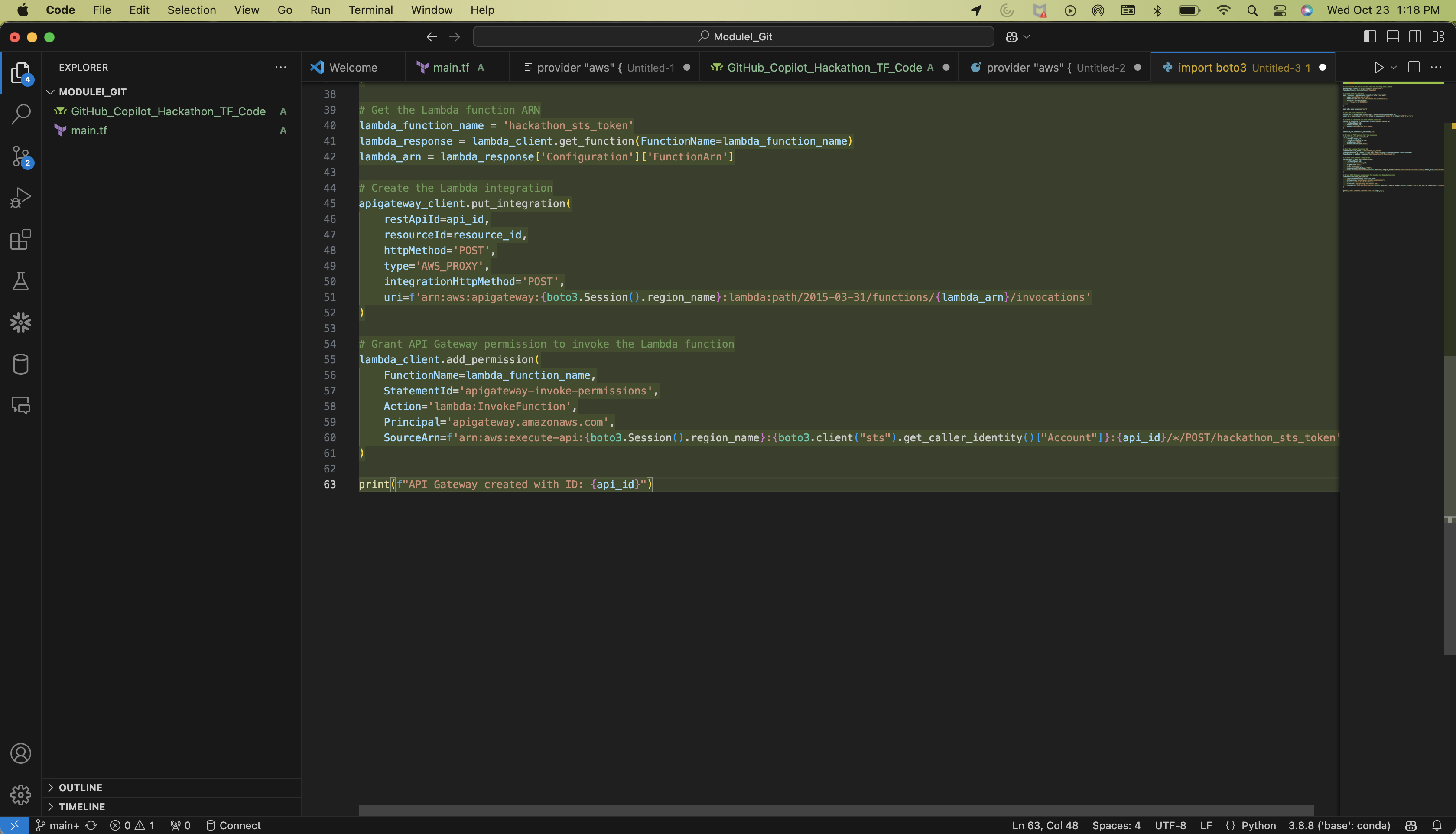
1. **create aws api gateway API called hackathon-aws-creds with lambda function hackathon\_sts\_token integration**

A screenshot of a computer

Description automatically generated



import boto3

# Initialize the boto3 client for API Gateway and Lambda

apigateway\_client = boto3.client('apigateway')

lambda\_client = boto3.client('lambda')

# Create the API Gateway

api\_response = apigateway\_client.create\_rest\_api(

name='hackathon-aws-creds',

description='API for hackathon AWS credentials',

endpointConfiguration={

'types': ['REGIONAL']

}

)

api\_id = api\_response['id']

# Get the root resource id

resources = apigateway\_client.get\_resources(restApiId=api\_id)

root\_id = next(item['id'] for item in resources['items'] if item['path'] == '/')

# Create a resource for the Lambda function

resource\_response = apigateway\_client.create\_resource(

restApiId=api\_id,

parentId=root\_id,

pathPart='hackathon\_sts\_token'

)

resource\_id = resource\_response['id']

# Create a POST method for the resource

apigateway\_client.put\_method(

restApiId=api\_id,

resourceId=resource\_id,

httpMethod='POST',

authorizationType='NONE'

)

# Get the Lambda function ARN

lambda\_function\_name = 'hackathon\_sts\_token'

lambda\_response = lambda\_client.get\_function(FunctionName=lambda\_function\_name)

lambda\_arn = lambda\_response['Configuration']['FunctionArn']

# Create the Lambda integration

apigateway\_client.put\_integration(

restApiId=api\_id,

resourceId=resource\_id,

httpMethod='POST',

type='AWS\_PROXY',

integrationHttpMethod='POST',

uri=f'arn:aws:apigateway:{boto3.Session().region\_name}:lambda:path/2015-03-31/functions/{lambda\_arn}/invocations'

)

# Grant API Gateway permission to invoke the Lambda function

lambda\_client.add\_permission(

FunctionName=lambda\_function\_name,

StatementId='apigateway-invoke-permissions',

Action='lambda:InvokeFunction',

Principal='apigateway.amazonaws.com',

SourceArn=f'arn:aws:execute-api:{boto3.Session().region\_name}:{boto3.client("sts").get\_caller\_identity()["Account"]}:{api\_id}/\*/POST/hackathon\_sts\_token'

)

print(f"API Gateway created with ID: {api\_id}")

1. **Create dynamodb table hackathon-user-table with provisoned capacity 2, userid and policies as columns**

A screenshot of a computer

Description automatically generated

import boto3

# Initialize a session using Amazon DynamoDB

dynamodb = boto3.resource('dynamodb')

# Create the DynamoDB table

table = dynamodb.create\_table(

TableName='hackathon-user-table',

KeySchema=[

{

'AttributeName': 'userid',

'KeyType': 'HASH' # Partition key

}

],

AttributeDefinitions=[

{

'AttributeName': 'userid',

'AttributeType': 'S'

}

],

ProvisionedThroughput={

'ReadCapacityUnits': 2,

'WriteCapacityUnits': 2

}

)

# Wait until the table exists.

table.meta.client.get\_waiter('table\_exists').wait(TableName='hackathon-user-table')

print("Table created successfully.")

1. **Create S3 bucket hackathon-bucket-1 with restricted S3 bucket policy**

A screenshot of a computer

Description automatically generated

import boto3

# Initialize a session using Amazon S3

s3 = boto3.client('s3')

# Create a bucket

bucket\_name = 'hackathon-bucket-1'

s3.create\_bucket(Bucket=bucket\_name)

# Define the bucket policy

bucket\_policy = {

"Version": "2012-10-17",

"Statement": [

{

"Sid": "AddPerm",

"Effect": "Deny",

"Principal": "\*",

"Action": "s3:\*",

"Resource": f"arn:aws:s3:::{bucket\_name}/\*",

"Condition": {

"Bool": {

"aws:SecureTransport": "false"

}

}

}

]

}

# Convert the policy to a JSON string

bucket\_policy = json.dumps(bucket\_policy)

# Set the new policy on the given bucket

s3.put\_bucket\_policy(Bucket=bucket\_name, Policy=bucket\_policy)

print(f"Bucket {bucket\_name} created with restricted policy.")

1. **Create IAM role hackathon-sts-system-role with permission to create IAM roles with name hackathon-\* , create and attach IAM policies starting with hackathon\* , sts:AssumeRole on IAM roles hackathon\*, RW access on Dynamodb table hackathon-user-table**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

import boto3

# Initialize a session using Amazon STS

session = boto3.Session()

iam\_client = session.client('iam')

# Define the trust policy for the role

trust\_policy = {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "sts.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

# Define the permissions policy for the role

permissions\_policy = {

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"iam:CreateRole",

"iam:CreatePolicy",

"iam:AttachRolePolicy",

"sts:AssumeRole",

"dynamodb:PutItem",

"dynamodb:GetItem",

"dynamodb:UpdateItem",

"dynamodb:DeleteItem"

],

"Resource": [

"arn:aws:iam::\*:role/hackathon-\*",

"arn:aws:iam::\*:policy/hackathon\*",

"arn:aws:dynamodb:\*:\*:table/hackathon-user-table"

]

}

]

}

# Create the IAM role

role\_response = iam\_client.create\_role(

RoleName='hackathon-sts-system-role',

AssumeRolePolicyDocument=json.dumps(trust\_policy),

Description='Role with permissions to manage hackathon resources'

)

# Attach the permissions policy to the role

policy\_response = iam\_client.put\_role\_policy(

RoleName='hackathon-sts-system-role',

PolicyName='hackathon-permissions-policy',

PolicyDocument=json.dumps(permissions\_policy)

)

print("Role ARN:", role\_response['Role']['Arn'])

5. Create AWS lambda function  hackathon\_sts\_token and attach IAM role hackathon-sts-system-role

A screenshot of a computer

Description automatically generated

import json

import boto3

def lambda\_handler(event, context):

sts\_client = boto3.client('sts')

assumed\_role\_object = sts\_client.assume\_role(

RoleArn="arn:aws:iam::123456789012:role/hackathon-sts-system-role",

RoleSessionName="AssumeRoleSession1"

)

credentials = assumed\_role\_object['Credentials']

return {

'statusCode': 200,

'body': json.dumps({

'AccessKeyId': credentials['AccessKeyId'],

'SecretAccessKey': credentials['SecretAccessKey'],

'SessionToken': credentials['SessionToken']

})

}

A screenshot of a computer

Description automatically generated