Project Delivery

BeatDrift-CLI Automation Tool Implementation

COSC595
Trimester 1 2022

Jason O. Aboh
Paldeep Kaur
Sumit Khokhar

Table of Contents

6.3.3 Terraform Plan

1.0. Project Description

Creating a CLI tool which will be used to build a pipeline to address drift in an Infrastructure as Code (IaC) environment.

1.1. Client

Client: Mark Wallis, Director of Hunter Orbit Pty Ltd.

Email: mark@hunterorbit.com.au

1.2. Project Team

The team members are:

Name	Email id (s)
1. Jason Aboh	jaboh@myune.edu.au , bigbossw107@gmail.com
2. Paldeep Kaur	pkaur21@myune.edu.au , paldeepkaur1995@gmail.com
3. Sumit Khokhar	skhokha2@myune.edu.au , sumitkhokhar359@gmail.com

1.3. Project github link

https://github.com/Jase-The-Ace/beatdrift-cli

2.0. Stakeholders

The stakeholders of the project are as follows:

- Client: **Mark Wallis**
- Unit Coordinator (COSC 595): **Dr. Edmund Sadgrove**
- Team Member (Project Lead): **Jason O. Aboh**
- Team Member: **Sumit Khokhar**
- Team Member: **Paldeep Kaur**

3.0 Project Objectives

The objectives of the project are as below:

- Creating a test AWS environment.

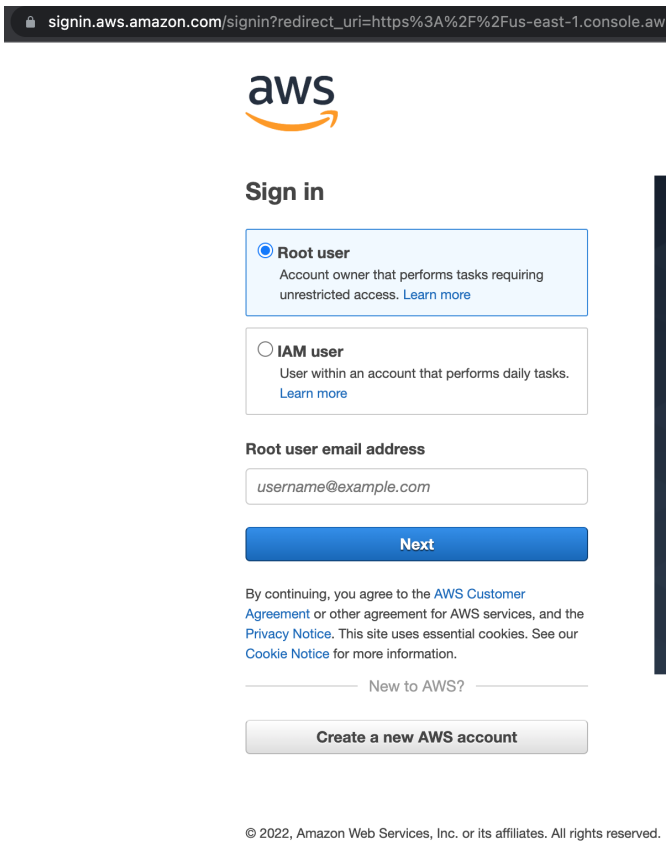
- Crawl the AWS environment to collect the resources.
- Writing scripts in Terraform and crawling resources in Terraform.
- Drift detection in the CLI environment between two resources (AWS and Terraform) and generating JSON output or HTML reports.

3.1 Creating Resources in AWS for testing (crawling)

After creation of an aws account and set up of credentials, to access the given aws; navigate to the aws Iam console website with the designated region at the beginning of the url link, the below link takes us to the “us-east-1” region:

https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1#/home

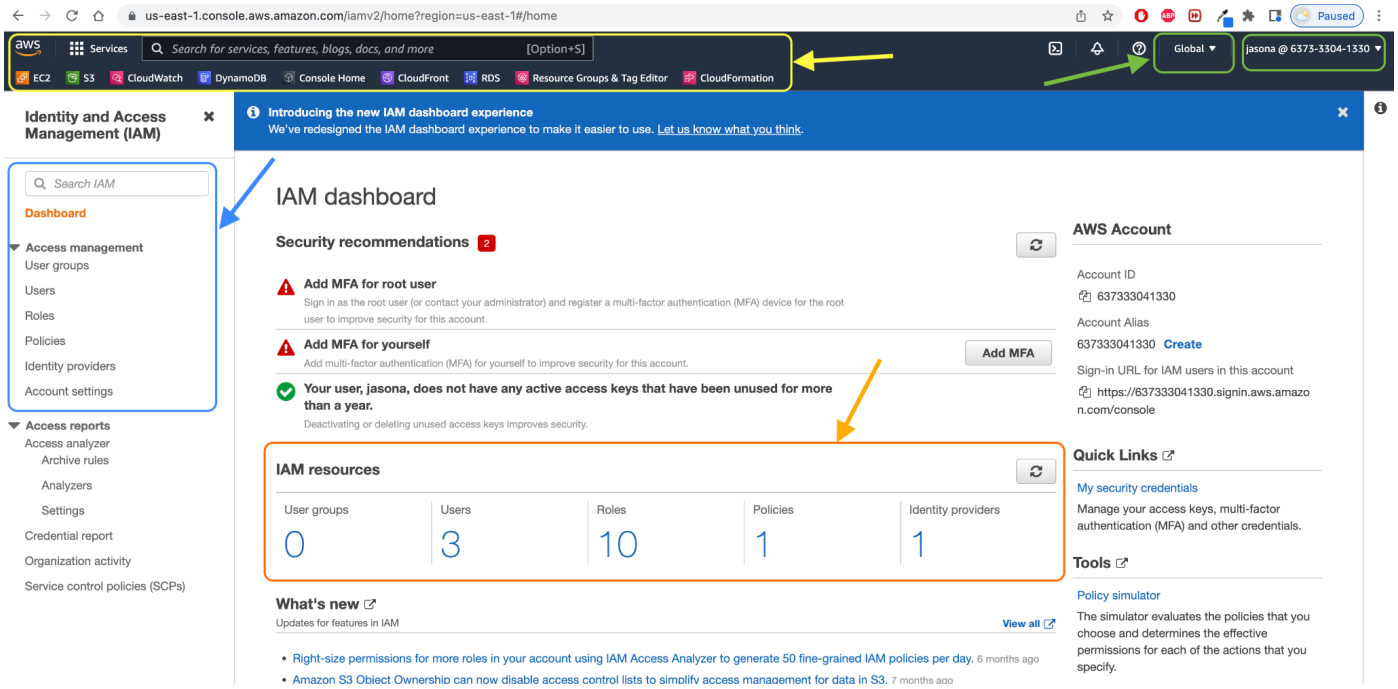
The user shall arrive at the page below; which they will be required to enter the credentials for the AWS account they have access to:



The user will need to input credentials for a “Root” user account (performs tasks with unrestricted access), or as a “IAM” user account (a user within an account that performs daily tasks).

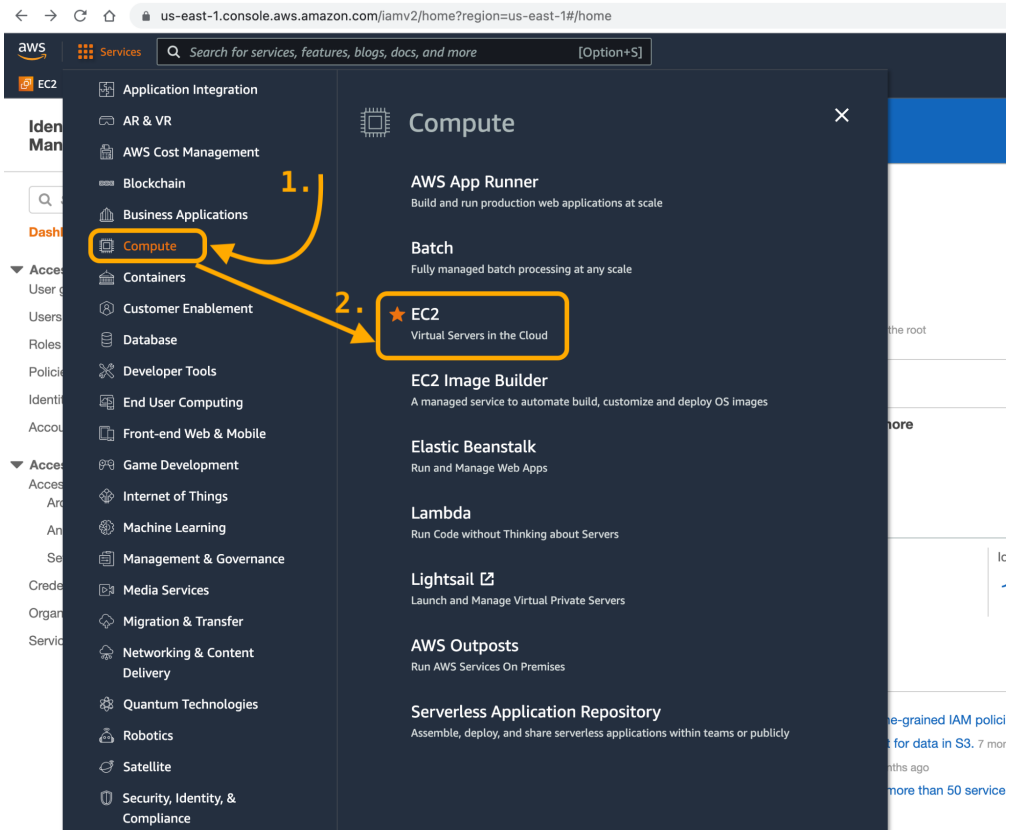
If the user supplies an alias (account name) or account ID number rather than the email address of a “Root” user account, they are taken to the IAM user page to sign in using an alias or account ID as shown below:

On successful sign in, the user arrives at the IAM dashboard as shown below:

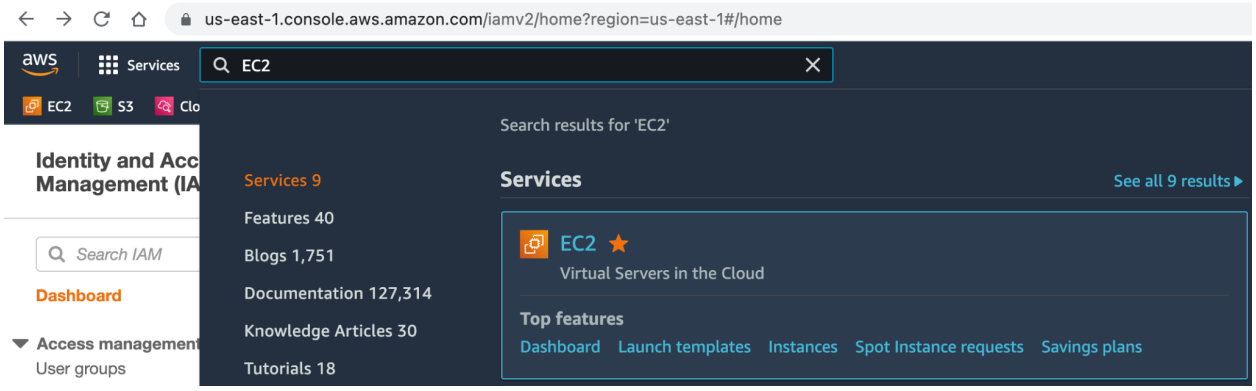


Please note that the IAM access management settings are located to the left within the blue rectangle, access to the AWS services is located to the top left within the yellow rectangle, access to IAM Users, Roles, Policies and Identity providers are located within the orange rectangle in the middle of the page; then finally, the current account username and ID , along with the region it is signed into, is located at the top right of the screen within the light green rectangle.

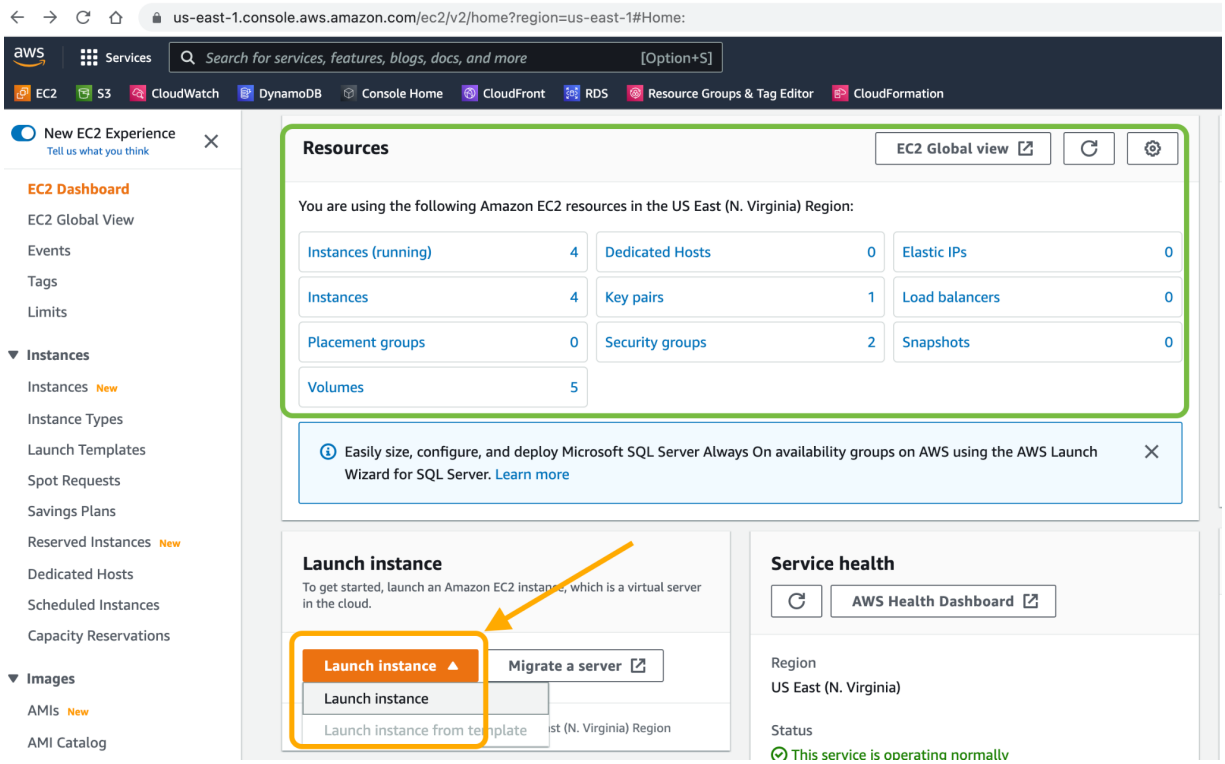
AWS resources can be created by either navigating to the services tab at the top right and clicking on the service sections within the scrollable options on the left, or searching for the service of interest to create:



Or by searching for “EC2” in the search bar located at the top left of the page:



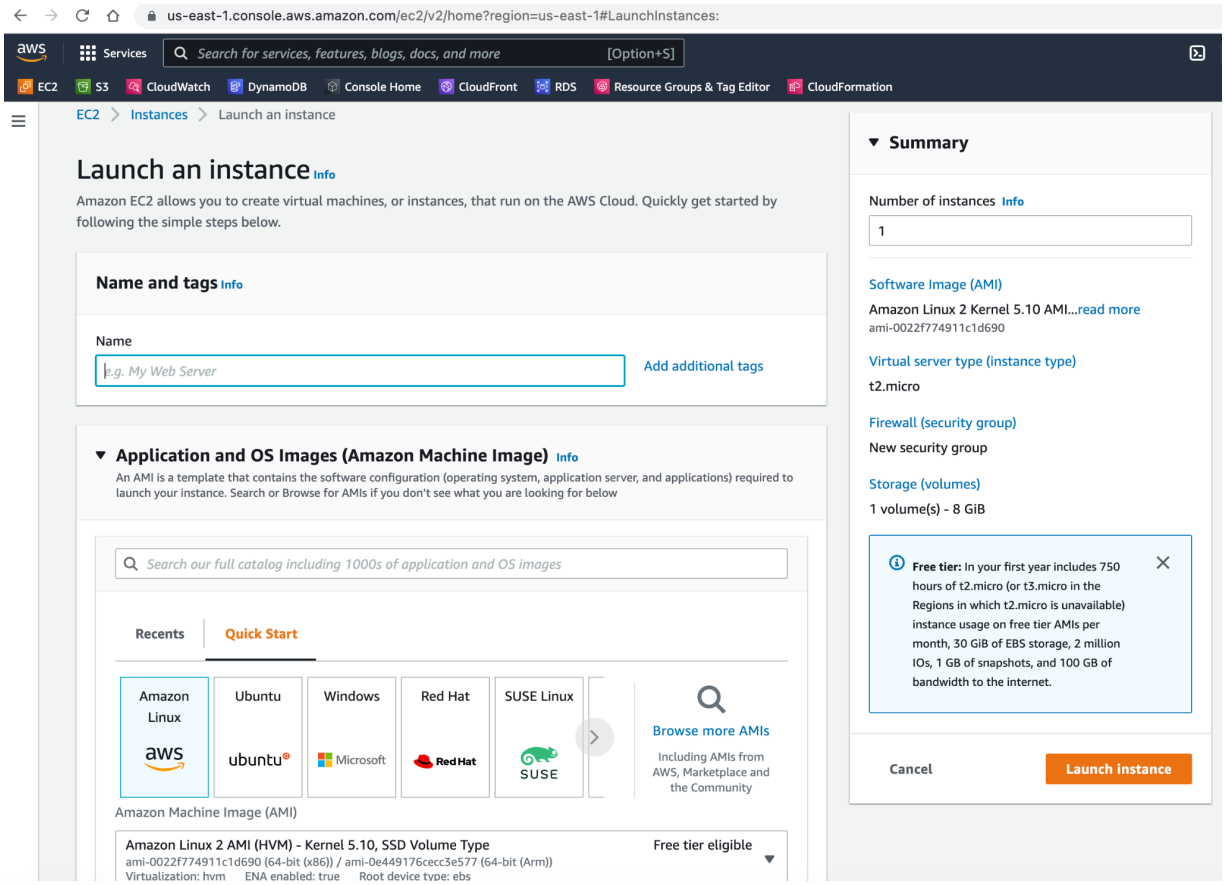
After selecting the “EC2” service (which is used to create virtual servers in the cloud); the user is then able to create an EC2 instance by clicking on the “Launch instance” button as shown below:



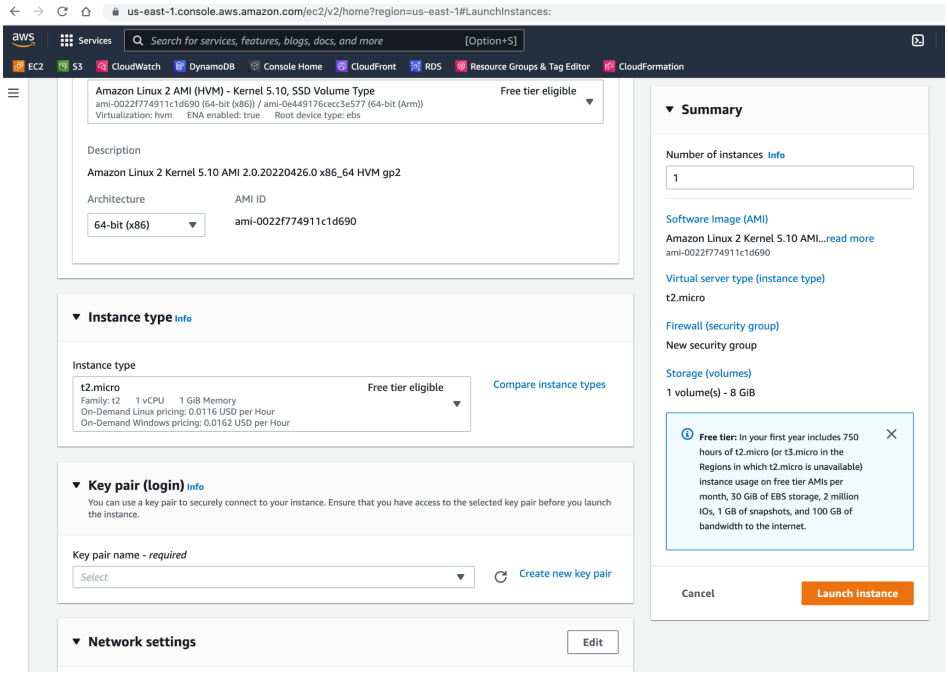
After clicking on the "Launch Instance" button, the user is directed to the form page below to configure the resource they intend to create, for EC2 instances the below is the form page the user must use to configure the instance:

At the top of the form, the user is able to assign a Name /Tag to the instance resource; A resource tag is a label that the user assigns to an AWS resource. Each tag consists of a key and an optional value, both of which the user defines - Tags allow the user to locate resources in the account via the use of the Tag Editor (which is used to list out the available tagged resources within a region - if specified).

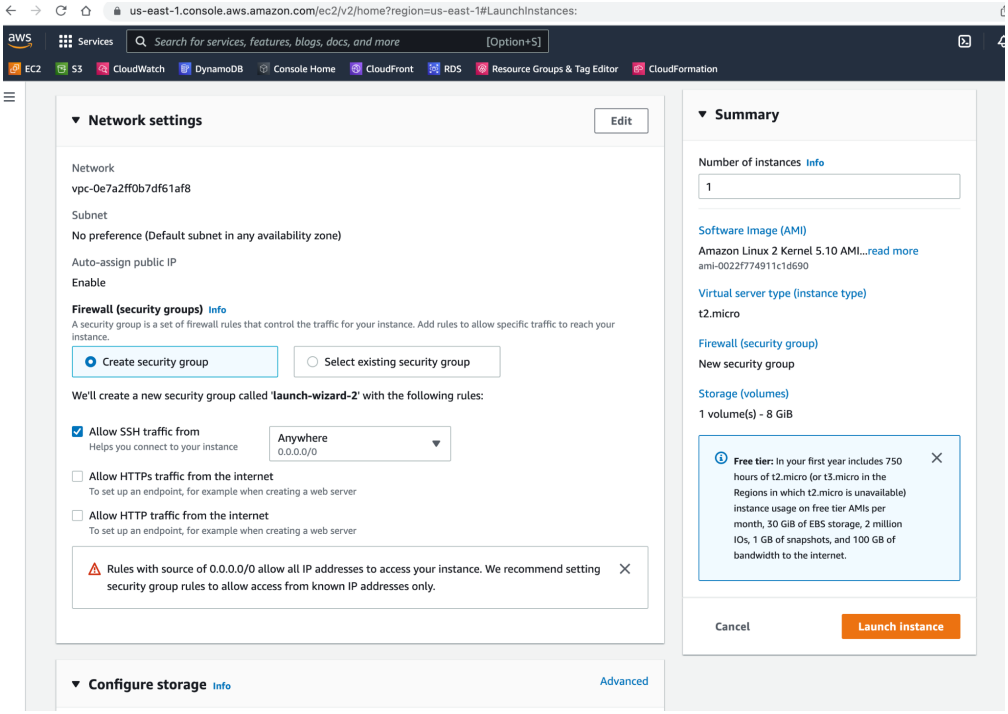
The user is also able to select an application and Operating system image, and on the right of the screen, the user can input the number of instances they intend to create:



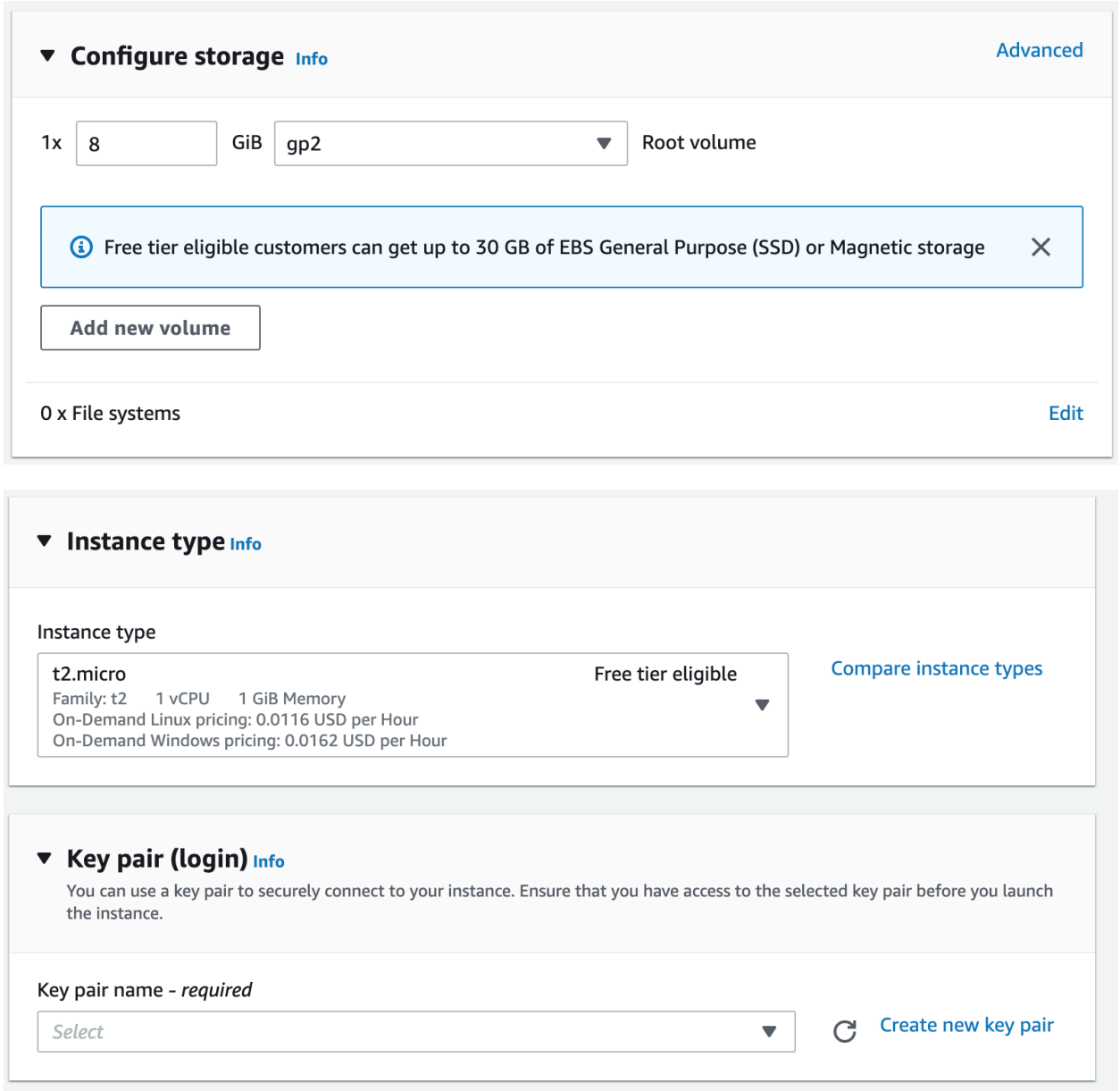
The user has the ability to configure instance types and key pairs:

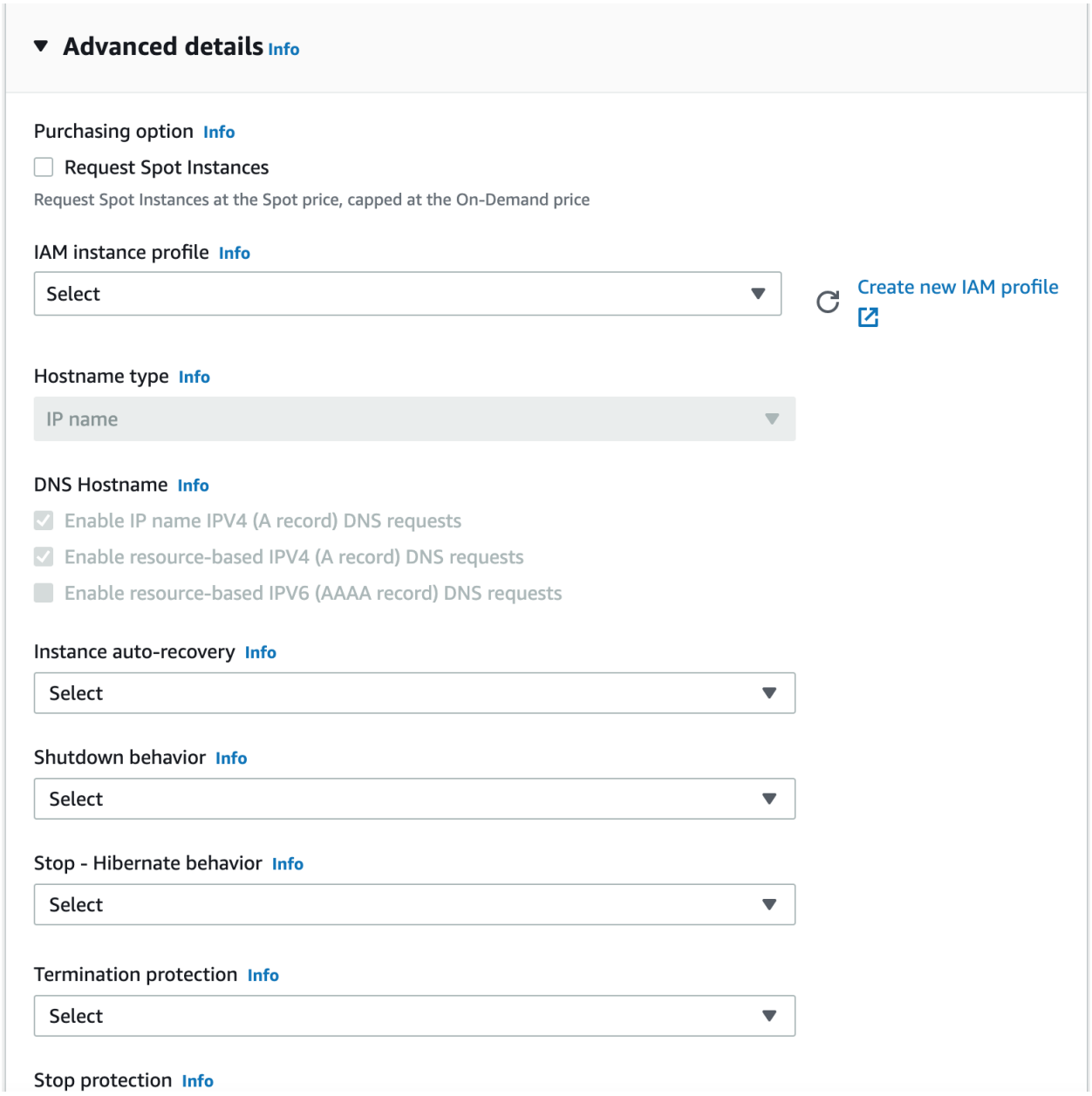


And also can configure Network settings for the EC2 instance being created:

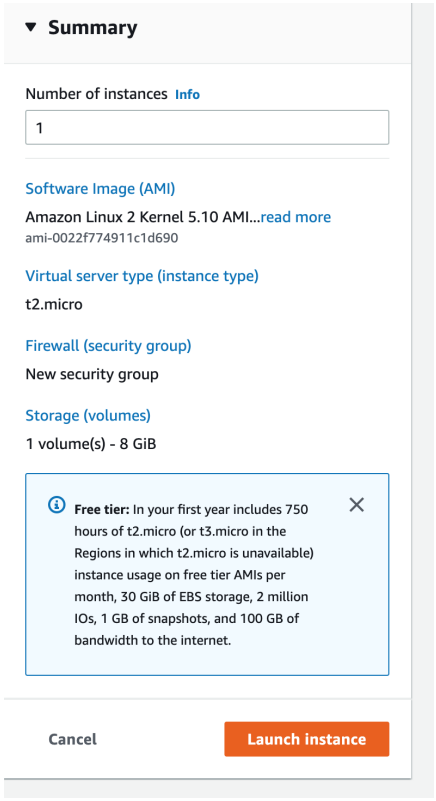


And set other configurations such as the below:

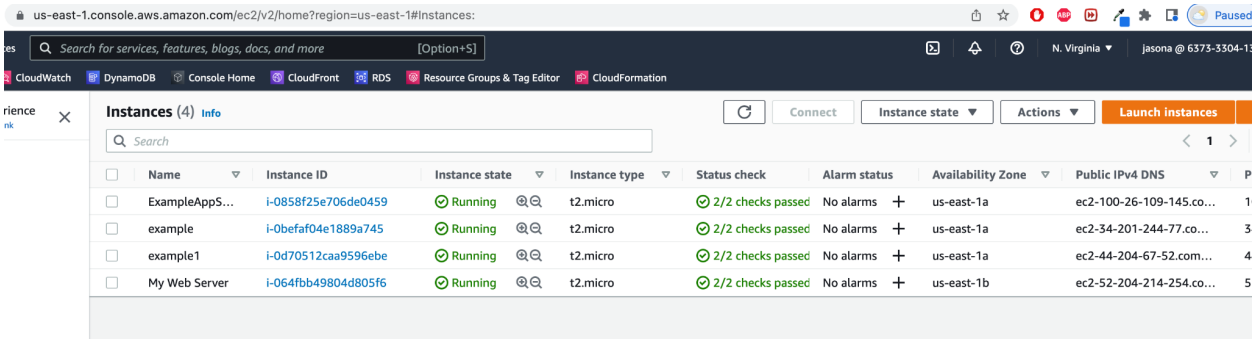




When done with the settings; the user can click the "Launch Instance" button to create the resource:

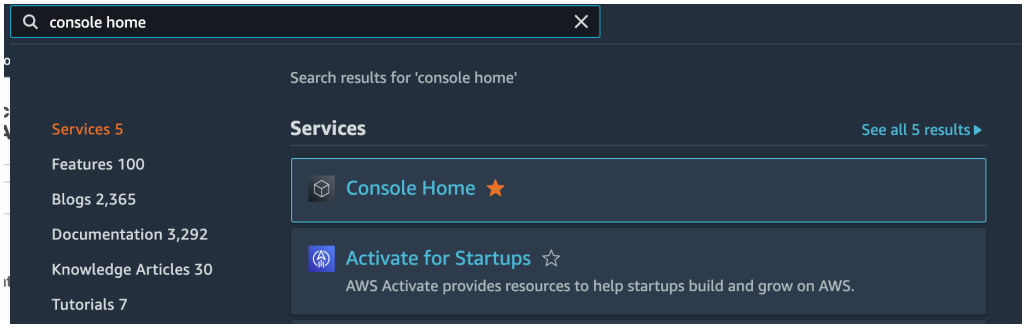


The new instance will then appear with its status in the list on the EC2 instance page, along with the other previously created instances:

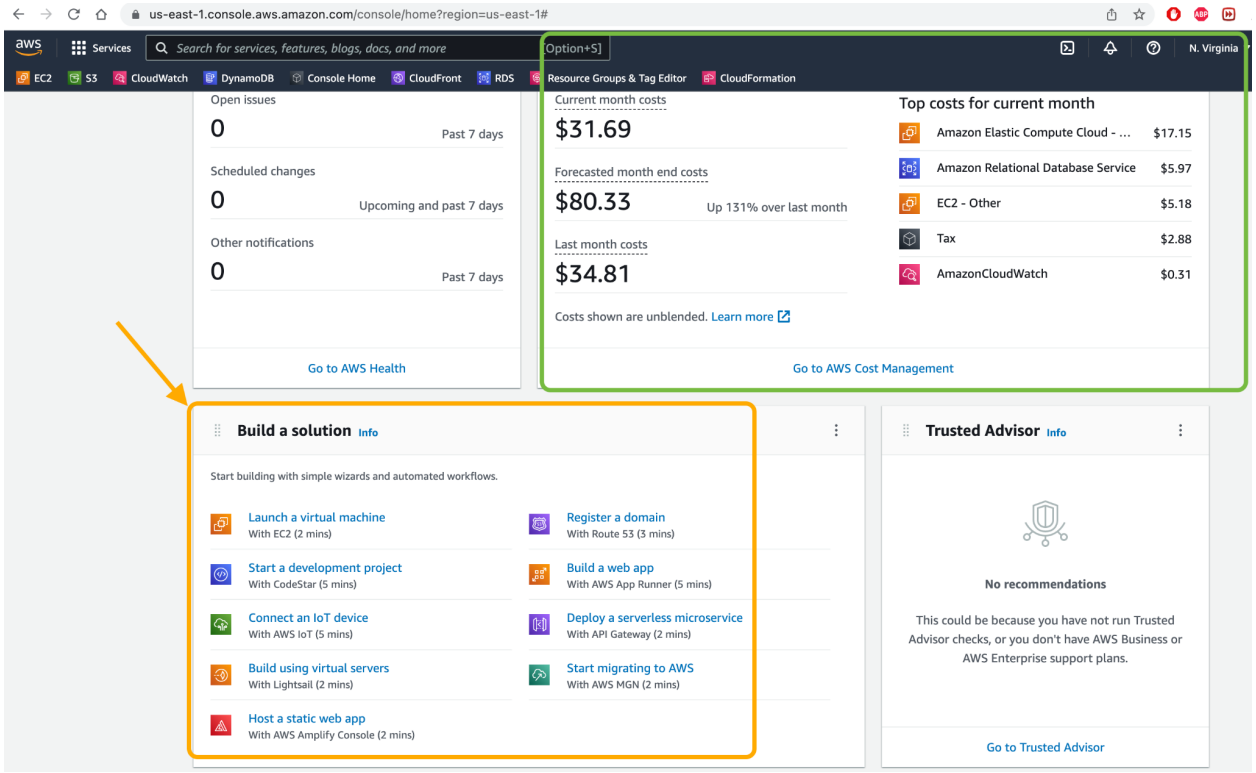


The user can also navigate to the AWS Management console where there is a dedicated section with links to build some common AWS resources (including the estimated time it would take to build on average).

This is accomplished by simply searching for “Console home” at the top left search bar of the page



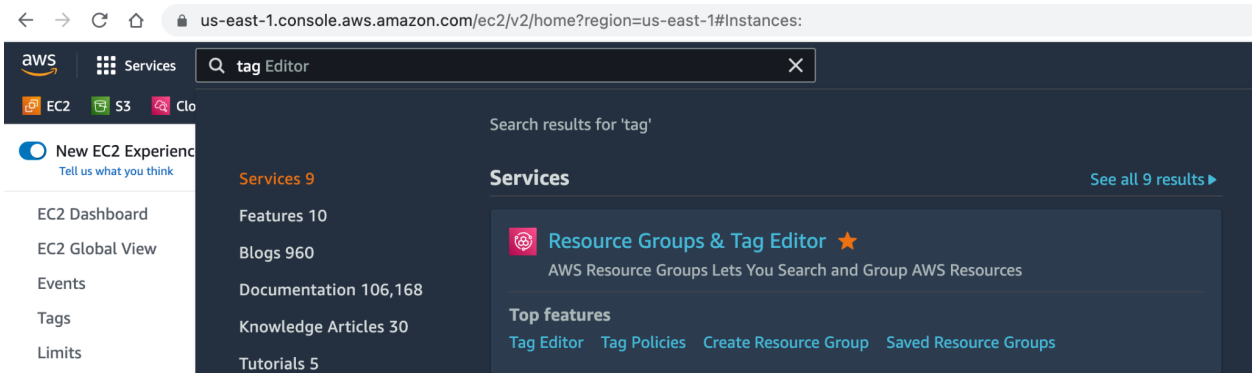
Then clicking on "Console Home" in the services results, to navigate to the "AWS Management Console". On the "AWS Management Console" hope page, the user can scroll down to utilize the provided links to begin the creation of some common AWS resources located at the bottom left of the screen in the orange rectangle highlight:



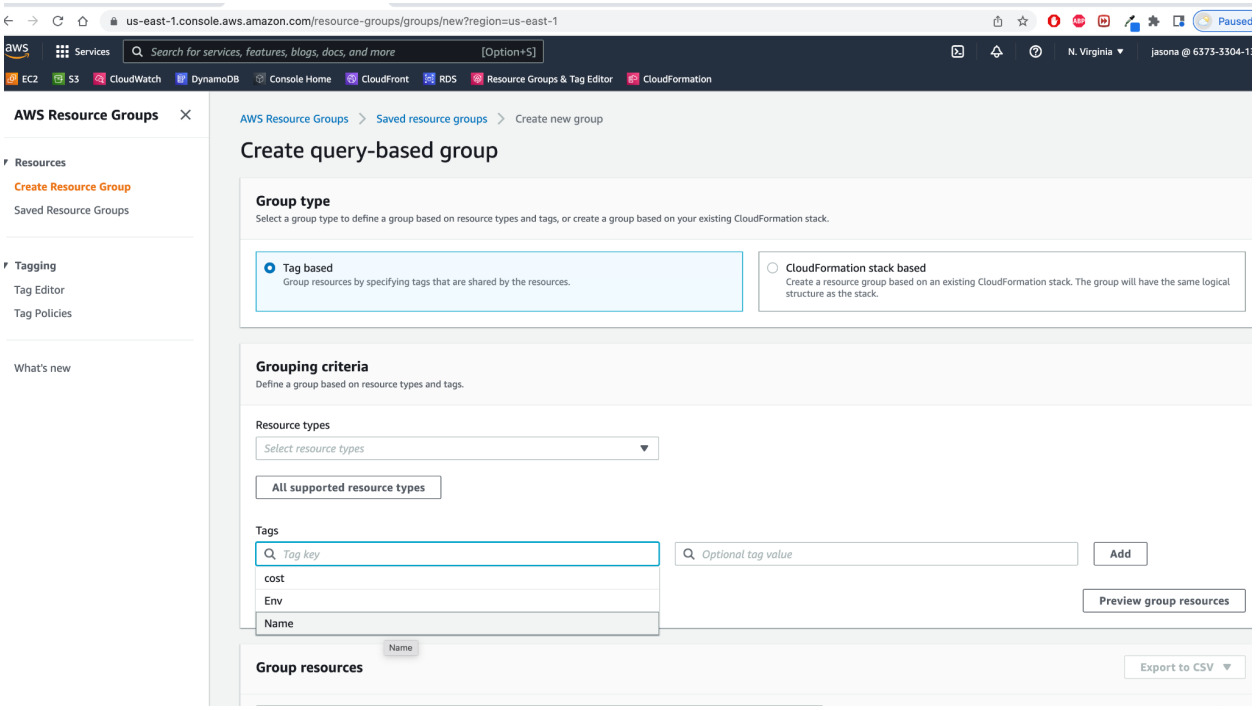
The user should note that the costs for the current month's use of AWS resources and services are displayed at the right of the page within the green rectangle highlight.

The user can utilise the tag manager to list out created resources which have been tagged and add it to a group of resources, which they can monitor:

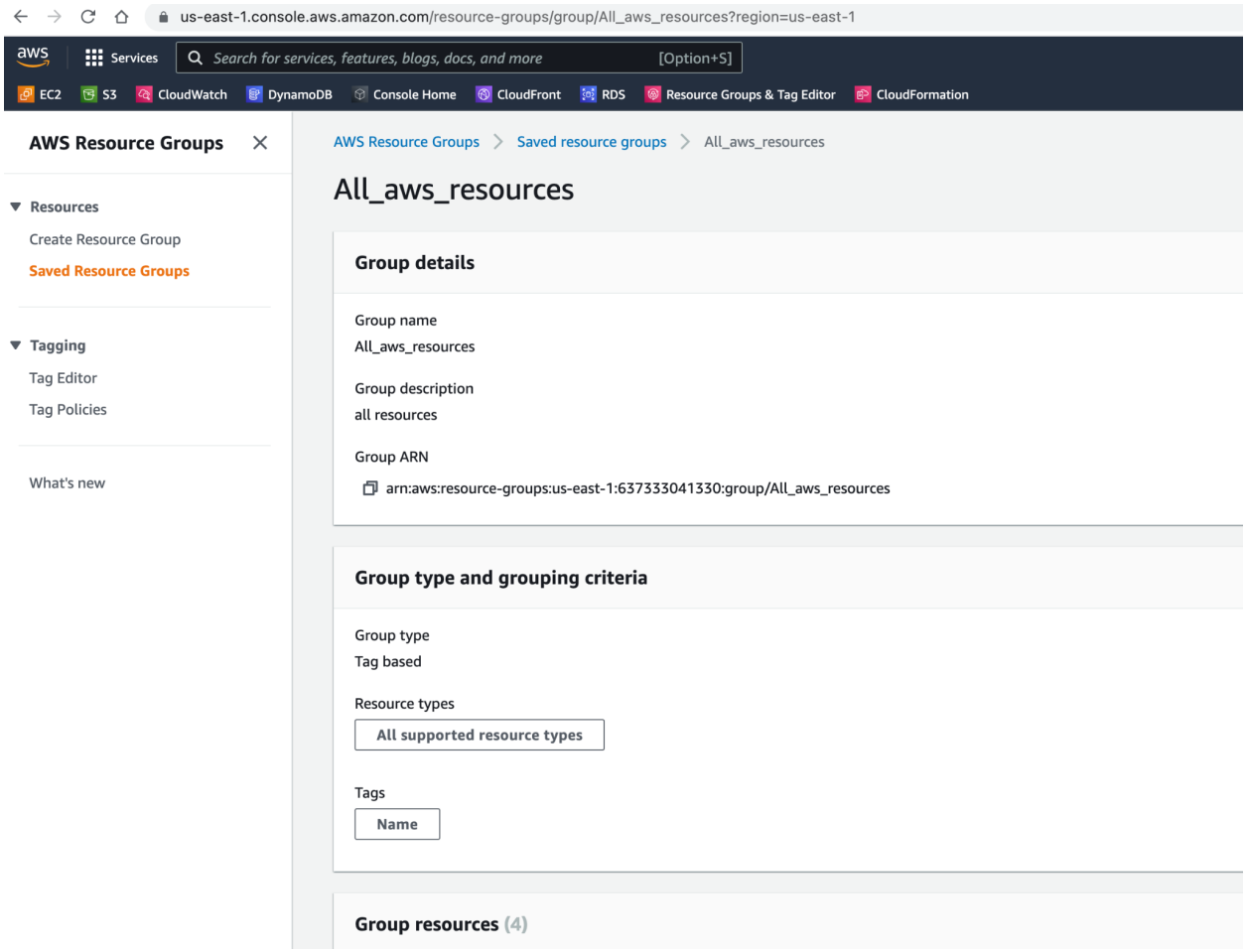
Search for the tag manager in the top bar:

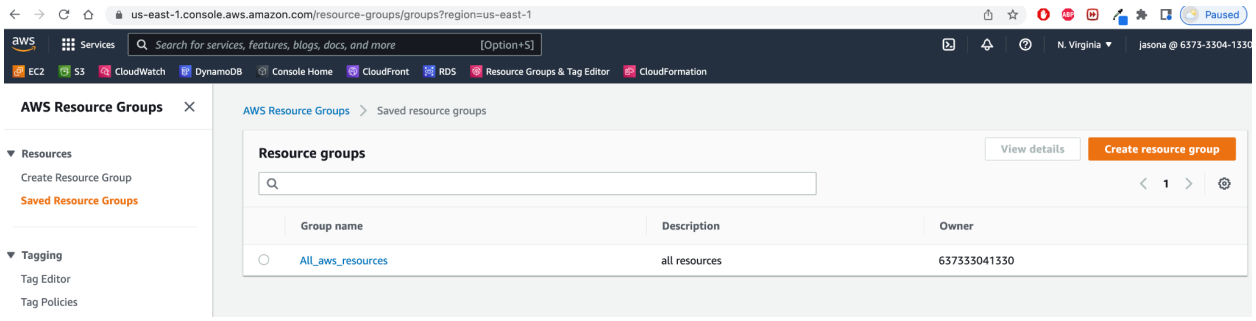


When the user clicks into this option, they are able to create groups of tagged resources:

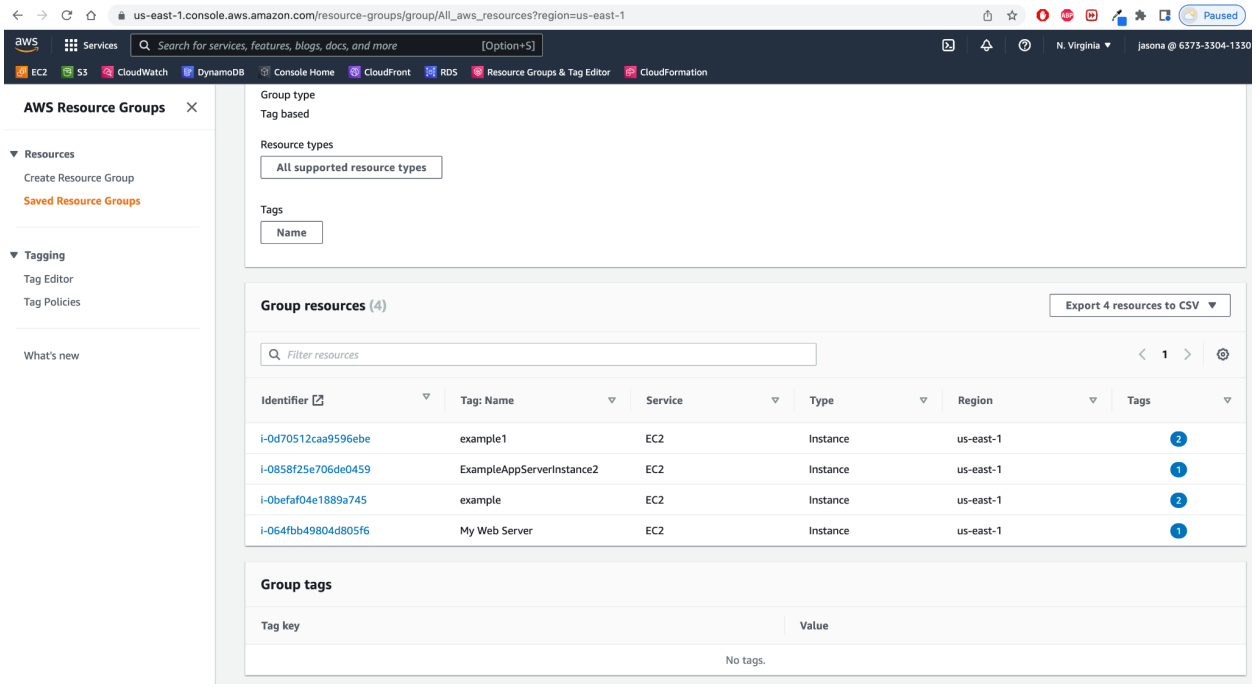


After filling in the fields and saving the group, the users will now have a saved group list of tagged resources:





which they can click into and monitor on the aws web app while cross-checking with the aws cli crawler:



3.2 Configure AWS Environment in CLI

The configuration of the AWS environment in the Command Line Interface is quite simple. It involves the use of amazon's "aws cli", which needs to be installed on the user's machine and configured with the user's account details and credentials.

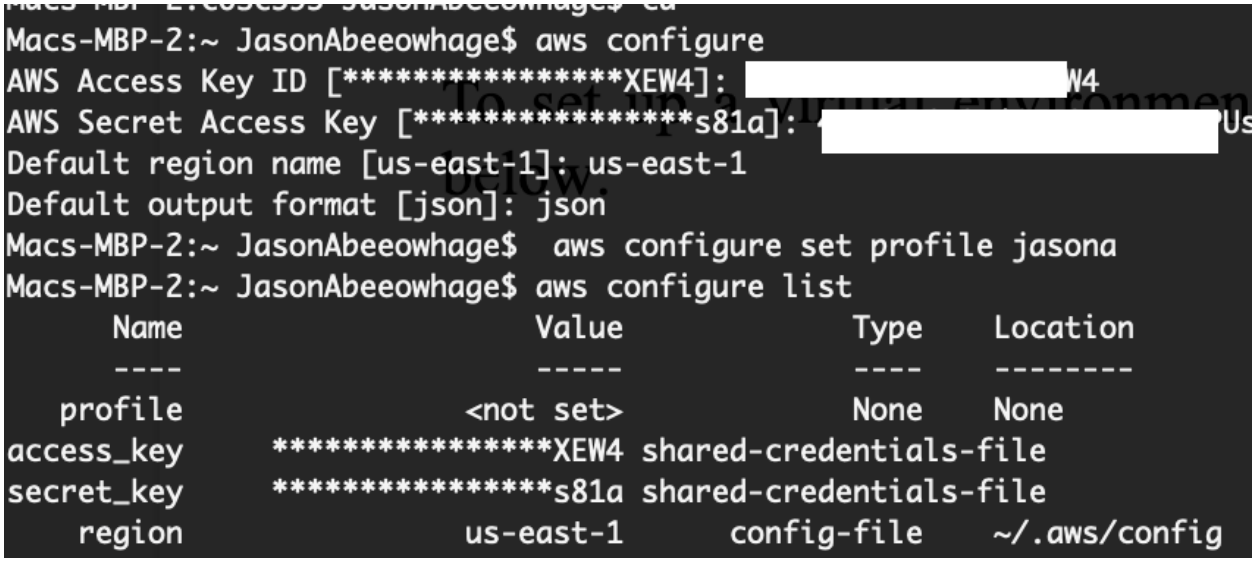
After the user accesses the directory of choice with

cd /directoryOfChoice

Input the following command and hit the return/enter button in the command line interface:

aws configure

You will be prompted and enter the corresponding credentials supplied from the AWS account you are using :



After entering the correct credentials for AWS Access Key ID, AWS Secret Access Key, Default region name, and output format; the user's aws cli environment is configured.

The user is also able to list out the AWS accounts that have been configured in the environment by entering :

aws configure list

As shown in the screenshot above.

3.3 Crawling AWS for resources with BeatDrift-CLI tool

Crawling AWS for specific or all resources can be done with the use of the "BeatDrift-CLI" tool.

The user simply needs to download the source code zip file and then unzip this file,

Then go into the "beatdrift-cli-master_copy" directory from the home directory it was downloaded into by typing in:

cd ./beatdrift-cli-master_copy"

within the directory of their choice (This can also be a virtual environment). The environment in which the python script is run, needs to have the required installed system software environment tools and versions as specified in the 'setup.py' file:

Programming language version: Python 2', Python 2.7', Python 3', or Python 3.6'

Boto3 version: 1.16.57

App_json_file_cache version: 0.2.2

To install the latest version of a package enter:

pip install 'PackageName'

To install a specific version, type the package name followed by the required version enter:

pip install 'PackageName ==1.4'

The user should note that there are compatibility issues with the latest version of boto3, due to several changes made that would affect the ability to run BeatDrift-cli as intended; until updates are made, users are advised to stick to the requirements stated.

pip CLI python packaging - version (or equivalent python package manager)

Can be used to install the specific versions of these packages, if they are still available; the user can also search the web to install the specific versions of the required libraries, to ensure the script runs as intended.

Tip:

Care must be taken to ensure the user's machine is set to the current date and time to avoid denial of access to list the available resources, due to invalid AWS credentials authentication caused by a non- automatically set time and date on the machine/computer; an example of this authentication error is shown below:

Once the environment is set up correctly; cd into the "BeatDrift-CLI" folder via the command line to execute the python scripts;

Please enter :

chmod +x __main__.py

Then :

cp __main__.py ~/bin/beatdrift-cli-master

The user would then be able to run the program using commands such as:

beatdrift-cli query --region us-east-1 --service ec2 --directory ./listdataCheck2/

However, since full integration was not acheived in the project, if you are having problems with the execution or have prompts about missing modules, you can utilize the 'aws-list-all' package incorporated by BeatDrift-cli (with credit given in the notice to the author) to achieve the same results of crawling AWS resources

and storing them as json files within the designated directory/folder within the project's folder:

AWS resources can be crawled by installing and using the "aws-list-all" package

If you are using a virtual environment simply enter the below code:

mkvirtualenv -p $(which python3) aws

pip install aws-list-all

After installation, The user is able to query specific AWS resources by typing in the following.

aws-list-all query --region us-east-1 --service ec2 --directory ./listdataCheck2/demo

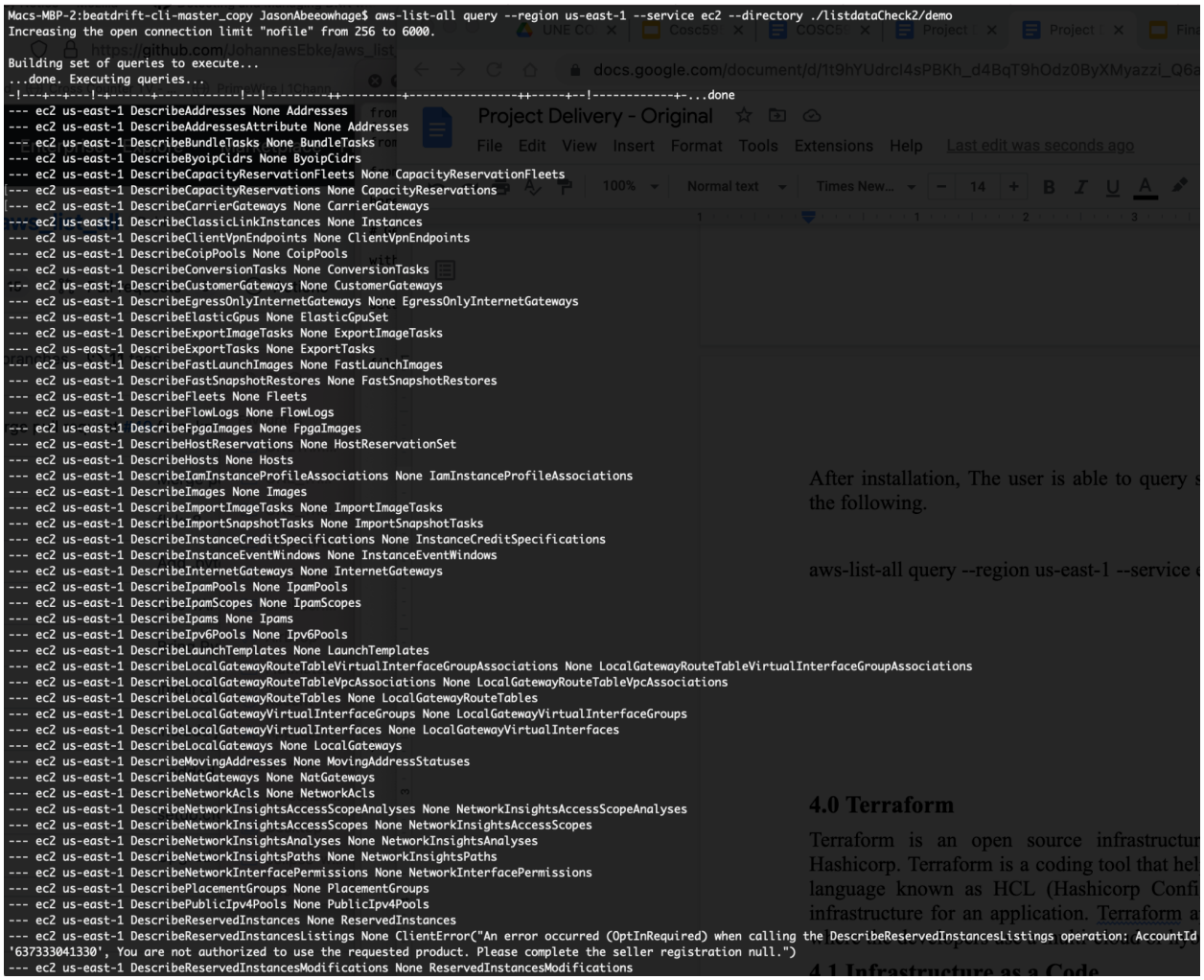
This command line argument crawls the provided aws account for all resources contained within the specified parameters:

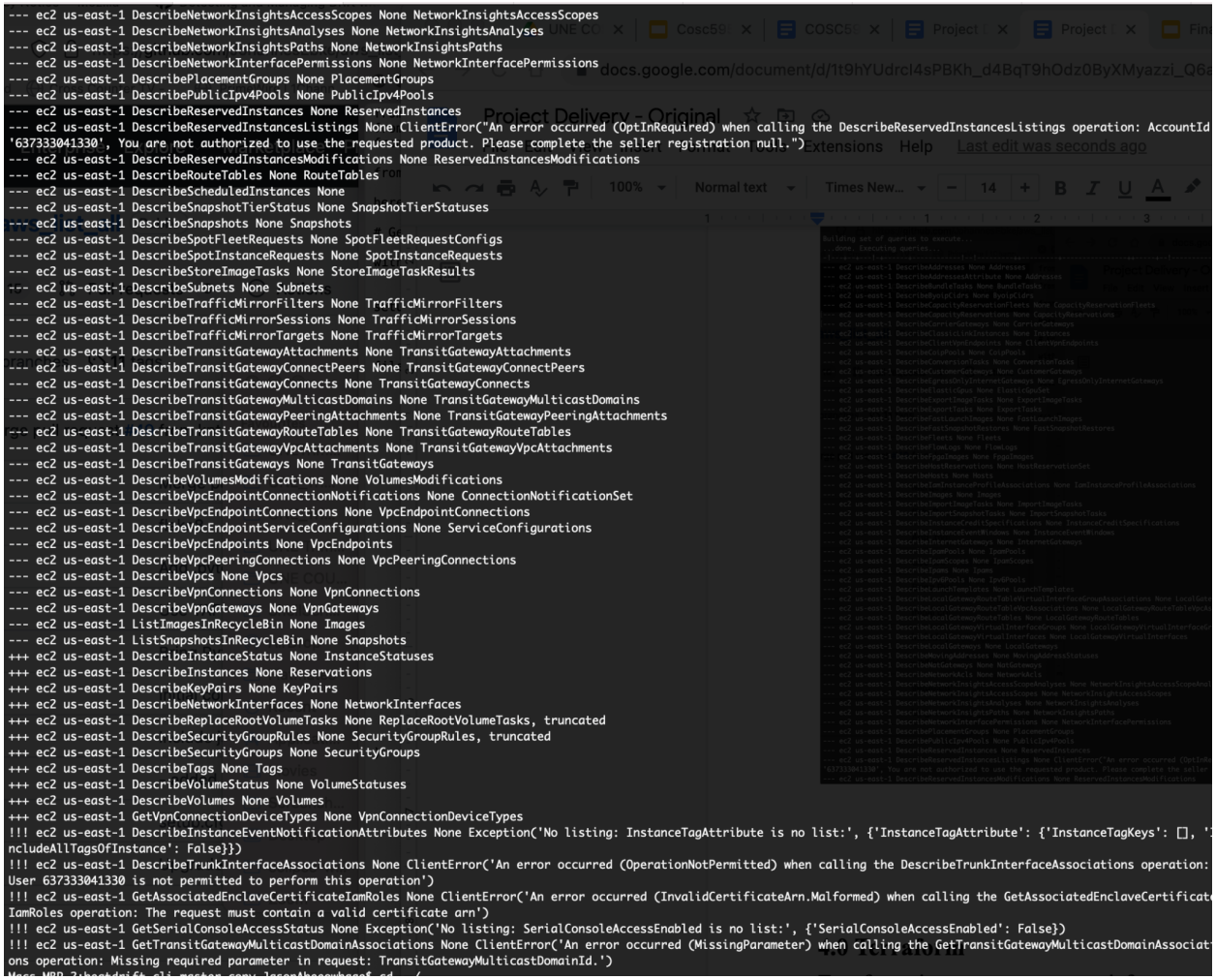
(1) The main command is "query"
(2) Parameter 1 - region : us-east-1: this parameter
(3) Parameter 2 - service : ec2
    - This parameter tells the script to only crawl ec2 resources.

(4) Parameter 3 - Directory: ./listdataCheck2/demo

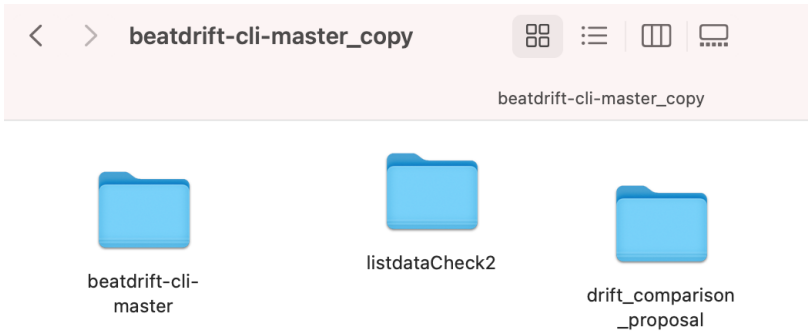
    - This parameter tells the script where to store the json files created to store the information about the AWS resources it has detected and listed.

More parameters like "-verbose" can be added and repeated for emphasis on expected outcome.


Macs-MBP-2:beatdrift-cli-master_copy JasonAbeeowhage$ aws-list-all query --region us-east-1 --service ec2 --directory ./listdataCheck2/demo
Increasing the open connection limit "nofile" from 256 to 6000.
Building set of queries to execute...
...done. Executing queries...
...done
--- ec2 us-east-1 DescribeAddresses None Addresses
--- ec2 us-east-1 DescribeAddressesAttribute None Addresses
--- ec2 us-east-1 DescribeBundleTasks None BundleTasks
--- ec2 us-east-1 DescribeByoipCidrs None ByoipCidrs
--- ec2 us-east-1 DescribeCapacityReservationFleets None CapacityReservationFleets
--- ec2 us-east-1 DescribeCapacityReservations None CapacityReservations
--- ec2 us-east-1 DescribeCarrierGateways None CarrierGateways
--- ec2 us-east-1 DescribeClassicLinkInstances None Instances
--- ec2 us-east-1 DescribeClientVpnEndpoints None ClientVpnEndpoints
--- ec2 us-east-1 DescribeCoipPools None CoipPools
--- ec2 us-east-1 DescribeConversionTasks None ConversionTasks
--- ec2 us-east-1 DescribeCustomerGateways None CustomerGateways
--- ec2 us-east-1 DescribeEgressOnlyInternetGateways None EgressOnlyInternetGateways
--- ec2 us-east-1 DescribeElasticGpus None ElasticGpuSet
--- ec2 us-east-1 DescribeExportImageTasks None ExportImageTasks
--- ec2 us-east-1 DescribeExportTasks None ExportTasks
--- ec2 us-east-1 DescribeFastLaunchImages None FastLaunchImages
--- ec2 us-east-1 DescribeFastSnapshotRestores None FastSnapshotRestores
--- ec2 us-east-1 DescribeFleets None Fleets
--- ec2 us-east-1 DescribeFlowLogs None FlowLogs
--- ec2 us-east-1 DescribeFpgaImages None FpgaImages
--- ec2 us-east-1 DescribeHostReservations None HostReservationSet
--- ec2 us-east-1 DescribeHosts None Hosts
--- ec2 us-east-1 DescribeIamInstanceProfileAssociations None IamInstanceProfileAssociations
--- ec2 us-east-1 DescribeImages None Images
--- ec2 us-east-1 DescribeImportImageTasks None ImportImageTasks
--- ec2 us-east-1 DescribeImportSnapshotTasks None ImportSnapshotTasks
--- ec2 us-east-1 DescribeInstanceCreditSpecifications None InstanceCreditSpecifications
--- ec2 us-east-1 DescribeInstanceEventWindows None InstanceEventWindows
--- ec2 us-east-1 DescribeInternetGateways None InternetGateways
--- ec2 us-east-1 DescribeIpamPools None IpamPools
--- ec2 us-east-1 DescribeIpamScopes None IpamScopes
--- ec2 us-east-1 DescribeIpams None Ipams
--- ec2 us-east-1 DescribeIpv6Pools None Ipv6Pools
--- ec2 us-east-1 DescribeLaunchTemplates None LaunchTemplates
--- ec2 us-east-1 DescribeLocalGatewayRouteTableVirtualInterfaceGroupAssociations None LocalGatewayRouteTableVirtualInterfaceGroupAssociations
--- ec2 us-east-1 DescribeLocalGatewayRouteTableVpcAssociations None LocalGatewayRouteTableVpcAssociations
--- ec2 us-east-1 DescribeLocalGatewayRouteTables None LocalGatewayRouteTables
--- ec2 us-east-1 DescribeLocalGatewayVirtualInterfaceGroups None LocalGatewayVirtualInterfaceGroups
--- ec2 us-east-1 DescribeLocalGatewayVirtualInterfaces None LocalGatewayVirtualInterfaces
--- ec2 us-east-1 DescribeLocalGateways None LocalGateways
--- ec2 us-east-1 DescribeMovingAddresses None MovingAddressStatuses
--- ec2 us-east-1 DescribeNatGateways None NatGateways
--- ec2 us-east-1 DescribeNetworkAcls None NetworkAcls
--- ec2 us-east-1 DescribeNetworkInsightsAccessScopeAnalyses None NetworkInsightsAccessScopeAnalyses
--- ec2 us-east-1 DescribeNetworkInsightsAccessScopes None NetworkInsightsAccessScopes
--- ec2 us-east-1 DescribeNetworkInsightsAnalyses None NetworkInsightsAnalyses
--- ec2 us-east-1 DescribeNetworkInsightsPaths None NetworkInsightsPaths
--- ec2 us-east-1 DescribeNetworkInterfacePermissions None NetworkInterfacePermissions
--- ec2 us-east-1 DescribePlacementGroups None PlacementGroups
--- ec2 us-east-1 DescribePublicIpv4Pools None PublicIpv4Pools
--- ec2 us-east-1 DescribeReservedInstances None ReservedInstances
--- ec2 us-east-1 DescribeReservedInstancesListings None ClientError("An error occurred (OptInRequired) when calling the DescribeReservedInstancesListings operation: AccountId
'637333041330', You are not authorized to use the requested product. Please complete the seller registration null.")
--- ec2 us-east-1 DescribeReservedInstancesModifications None ReservedInstancesModifications
Project Delivery - Original
File Edit View Insert Format Tools Extensions Help Last edit was seconds ago
After installation, The user is able to query
the following.
aws-list-all query --region us-east-1 --service
4.0 Terraform
Terraform is an open source infrastructur
Hashicorp. Terraform is a coding tool that hel
language known as HCL (Hashicorp Confi
infrastructure for an application. Terraform a

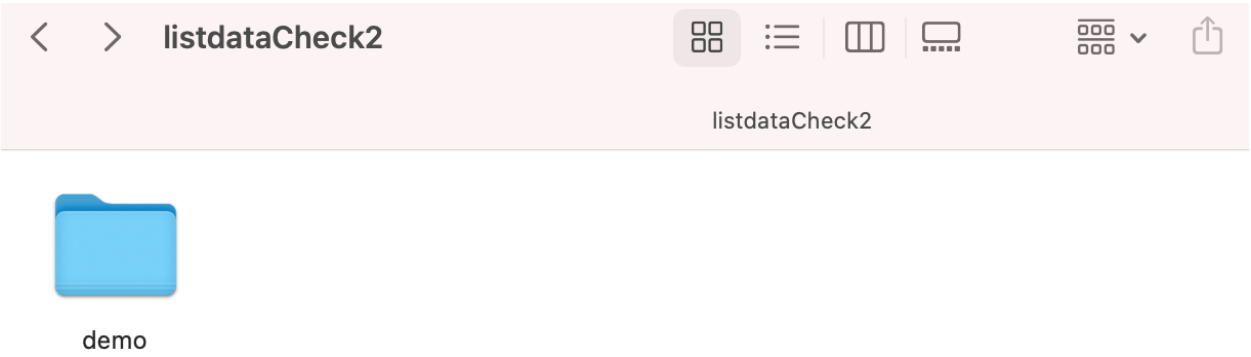


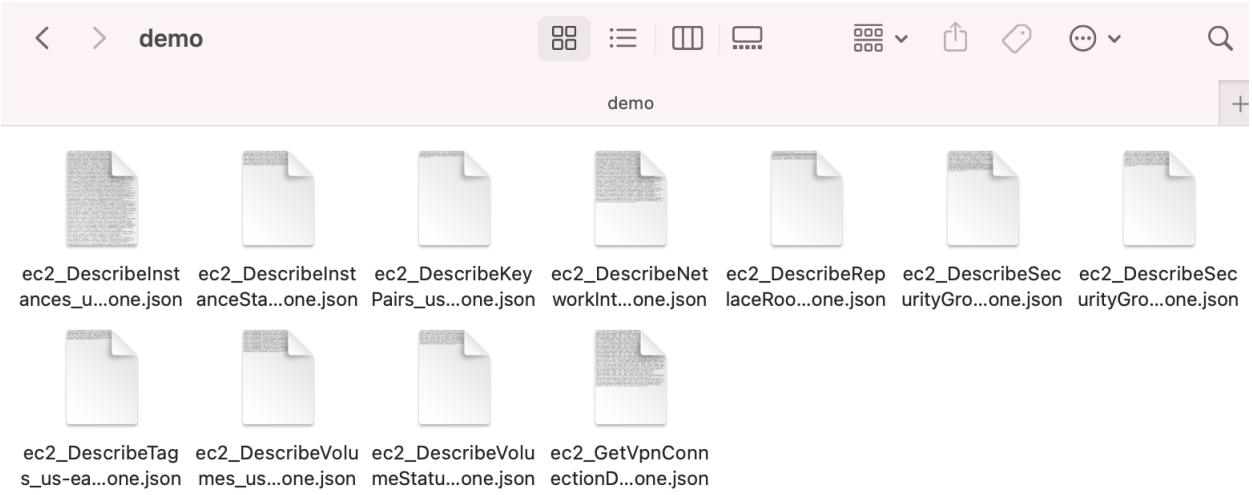
The data is listed with the "+" symbols, indicating resources that were found in the AWS account, the "-" symbol indicated resources not found in the AWS account and the "!!!" symbol indicates a warning.



The json files then appear within the created "listdataCheck2" folder

In the second created folder directory named “demo”:





The contents of the files created are of “JSON” format and contain the details of each crawled AWS resource; including properties such as “ARN” (Amazon Resource Names) which uniquely identify AWS resources. We require an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies, Amazon Relational Database Service (Amazon RDS) tags, and API calls.

When the query is run without a filter on services:

aws-list-all query --region us-east-1 --directory ./listdata2/demo2

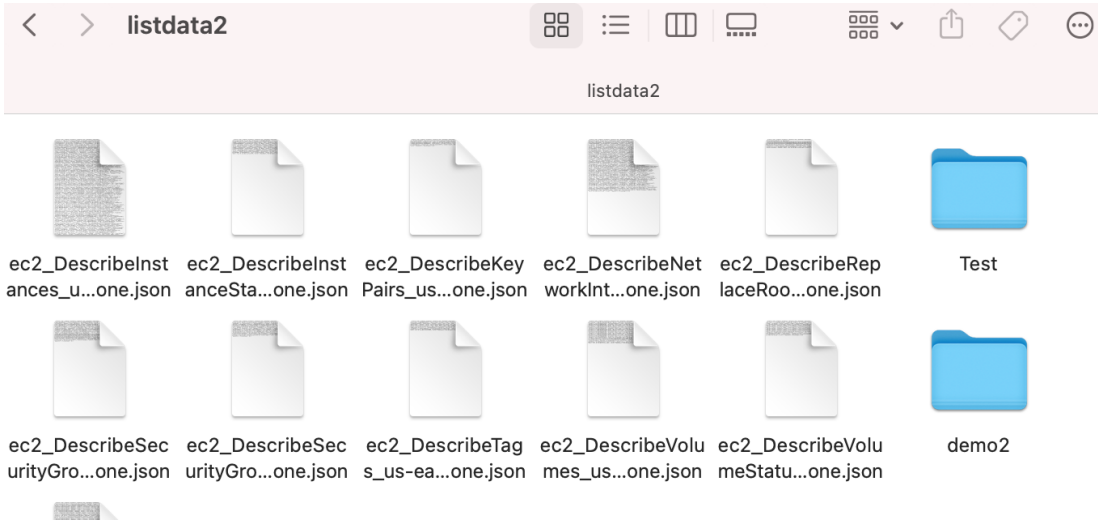
it takes a much longer time to crawl the AWS account for all resources in the specified region the difference in speed is

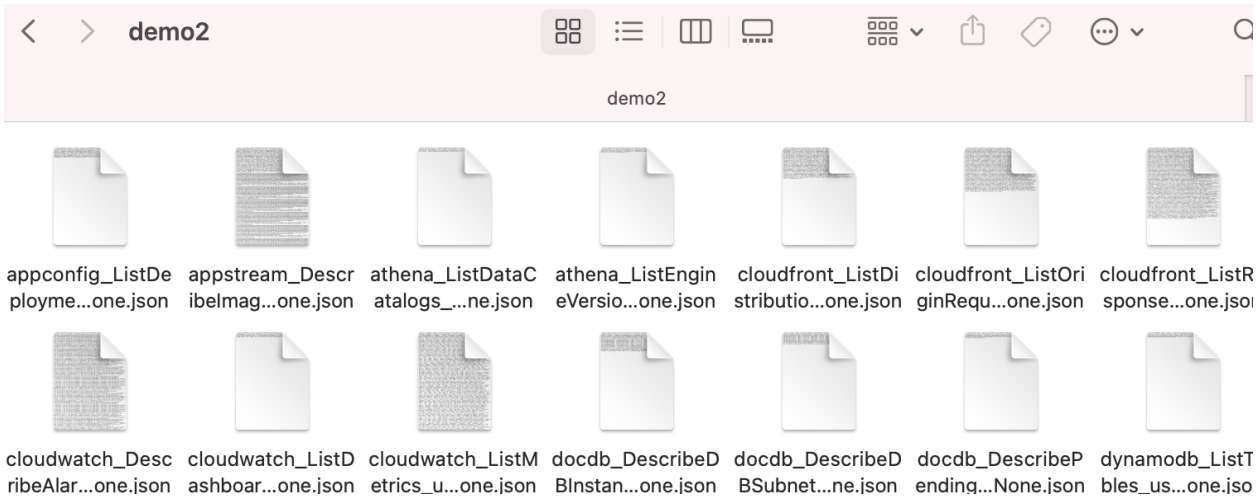
Benchmarked at

~ 4-6 minutes as opposed to the usual 45 seconds to 1 minute that it usually takes to crawl the AWS account for specific services.

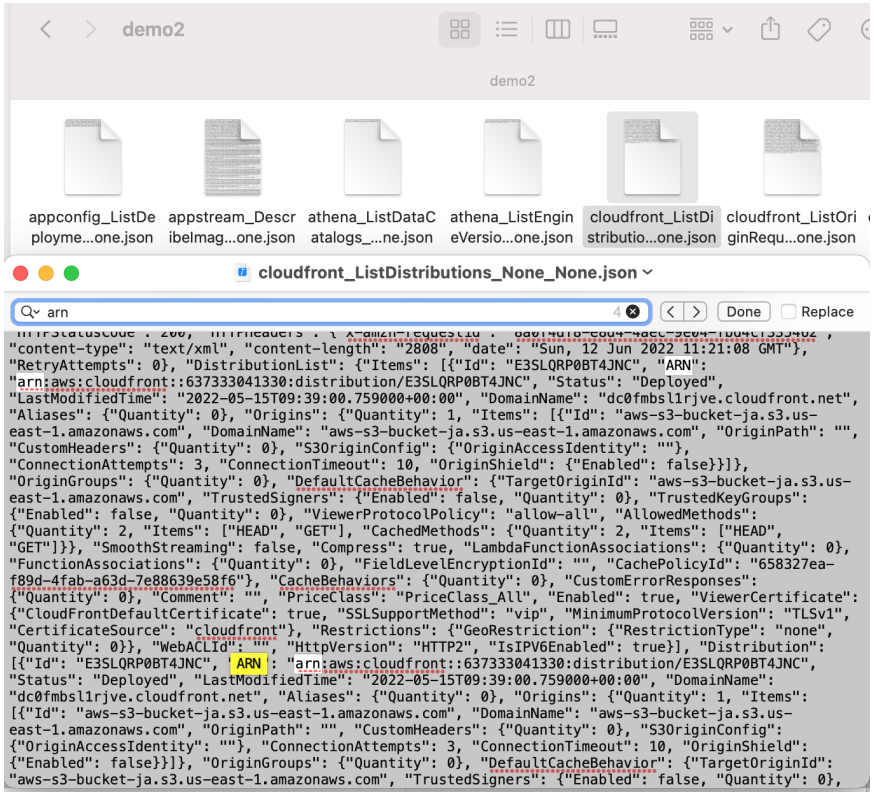
This is due to the scripts utilisation of a list of "service regions" stored in 'service_regions.json' file and "endpoint hosts" stored in the 'endpoints_hosts.json' file for the script to search through, when specific services are not specified in the command line argument.

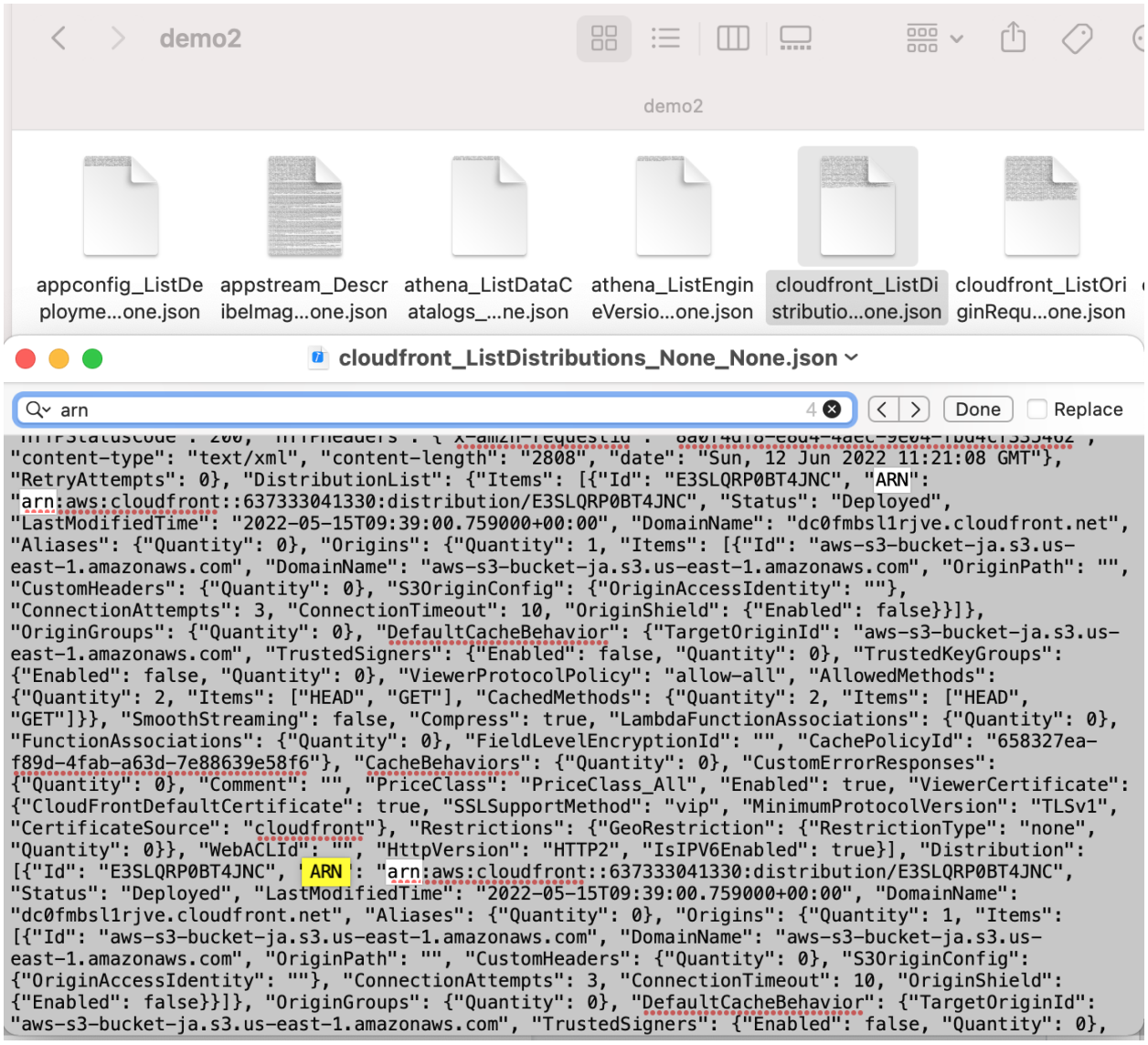
With the second command , we get more resources listed and stored in json file formats within the demo2 folder located in the "**./listdata2/demo2**" directory:





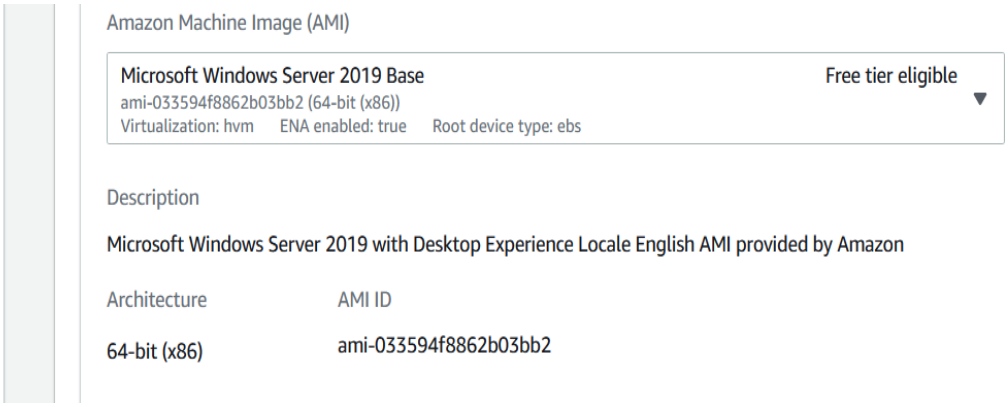
And within these json files are the “ARNs” of the crawled AWS resources which can be used to compare against the ARN’s within terraform’s tfstate file to detect infrastructural drift.





4.0 Creating of EC2 instance on AWS using Terraform module

The Terraform module is used in creating different instances on the AWS platform**.** For creating instances in AWS platform some of the features of AWS platform are required and these are region, AMI (Amazon Machine Image) and also key names. The region used as us-east-1 is obtained from the AWS platform whereas the AMI id is the unique id generated in the AWS platform when any new instance is launched**.** AMI id is the unique id which keeps varying from different operating systems and regarding this it can be said that the AMI id of any AWS instance launched in Ubuntu Operating System is different from that of the AMI id of AWS machine launched in Windows Operating system**.**



From the above image it has been visualised that AMI is created in a Windows machine and the above AMI is created when launching an EC2 instance in AWS platform. The above AMI is required when creating EC2 instance in AWS using the Terraform module.

Like the AMI image there is also another parameter known as “Key Name” which is called in AWS and implemented in the Terraform environment to create EC2 instance in AWS.

Before implementation of code in the Terraform module, the first and the foremost important step that is to be executed is downloading the current version of Terraform and also installing extension of Terraform in Visual studio to work with Terraform modules.

4.1. Source code (creating EC2 instance on AWS using Terraform module)

```
resource "aws_instance" "Count" {
 ami           = "ami-033594f8862b03bb2"
 instance_type = "t2.micro"
 key_name = "key342"

 tags = {
   Name = "Webappli"
 }
}
```

With the help of the source code visualised above , an EC2 instance is created in the AWS platform using the Terraform module. When an EC2 instance is created in AWS using the above script using the terraform module then automatically a

Terraform state file gets created in the folder where the terraform configuration file is stored. The Terraform state file gets created for every instance that are created in AWS using the Terraform module.

4.2. Source code (creating S3 bucket instance on AWS using Terraform module)

We can use Terraform to provision multiple S3 buckets that can later be managed by Terraform. The code to provision S3 bucket is shown below

```
resource "aws_s3_bucket" "terraform_s3_bucket" {

  bucket = var.bucket_name

  acl    = var.acl_value

}
```

From the above code, it is observed that the resource is aws_s3_bucket and the name of the S3 is terraform_s3_bucket. The two arguments for the resource are bucket and acl.

The code below shows the variables referenced in the main terraform file for bucket creation. These are in a file called variables.tf. This file contains declaration values for the variables.

```
variable "bucket_name" {

  default = "terraform_s3_bucket"

}

variable "acl_value" {

  default = "private"

}
```

The screenshot below shows Terraform trying to create the S3 bucket.

```
Terraform will perform the following actions:

  # aws_s3_bucket_acl.terraform_s3_bucket will be created
  + resource "aws_s3_bucket_acl" "terraform_s3_bucket" {
      + acl    = "private"
      + bucket = "terrform_s3_bucket"
      + id     = (known after apply)

      + access_control_policy {
          + grant {
              + permission = (known after apply)

              + grantee {
                  + display_name  = (known after apply)
                  + email_address = (known after apply)
                  + id            = (known after apply)
                  + type          = (known after apply)
                  + uri           = (known after apply)
                }
            }

          + owner {
              + display_name = (known after apply)
              + id           = (known after apply)
            }
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_s3_bucket_acl.terraform_s3_bucket: Creating...
aws_s3_bucket_acl.terraform_s3_bucket: Still creating... [10s elapsed]
```

With the help of the source code present above an S3 bucket instance is created . Similar to that of an EC2 instance when creating an S3 bucket instance, a terraform state file is created.

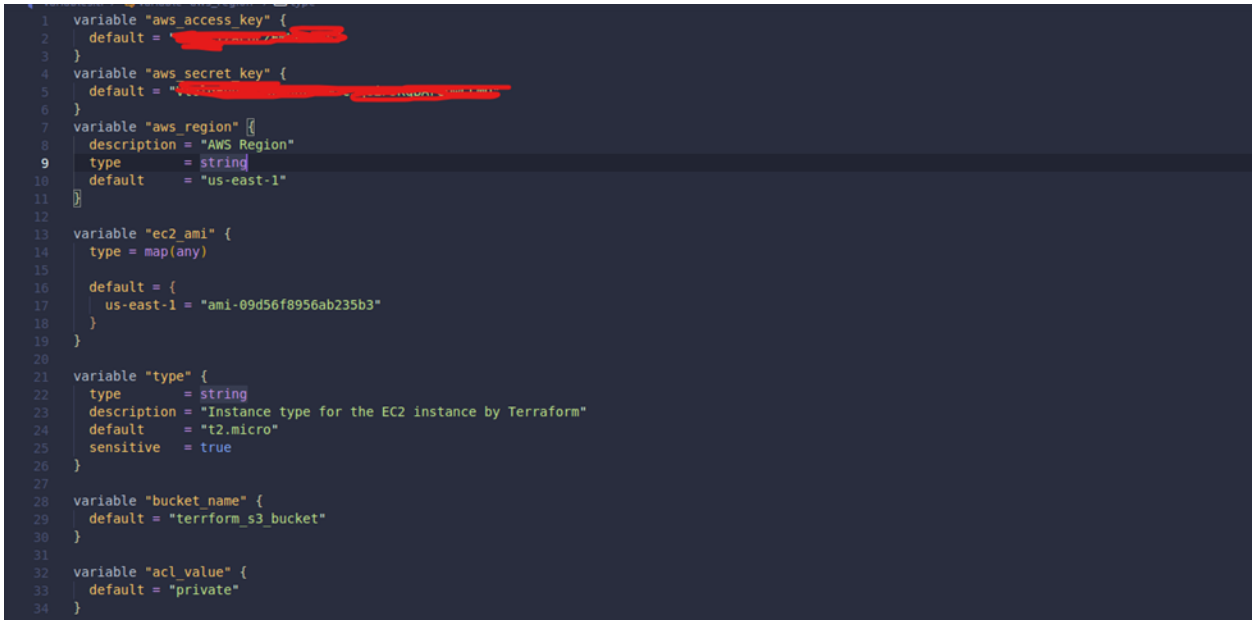
5.0 Detecting and Managing drift with Terraform CLI

The CLI tool is built using the Python programming language. The core Terraform commands are wrapped and implemented as functions. The code base for the CLI tool is divided into two:

- Terraform code that contains all the necessary scripts to provision resources such as EC2 instances, S3 buckets, security groups etc. The main code for Terraform (terraform.tf). The script is used to create various AWS resources and the file that contains the variables (variables.tf).

terraform.tf

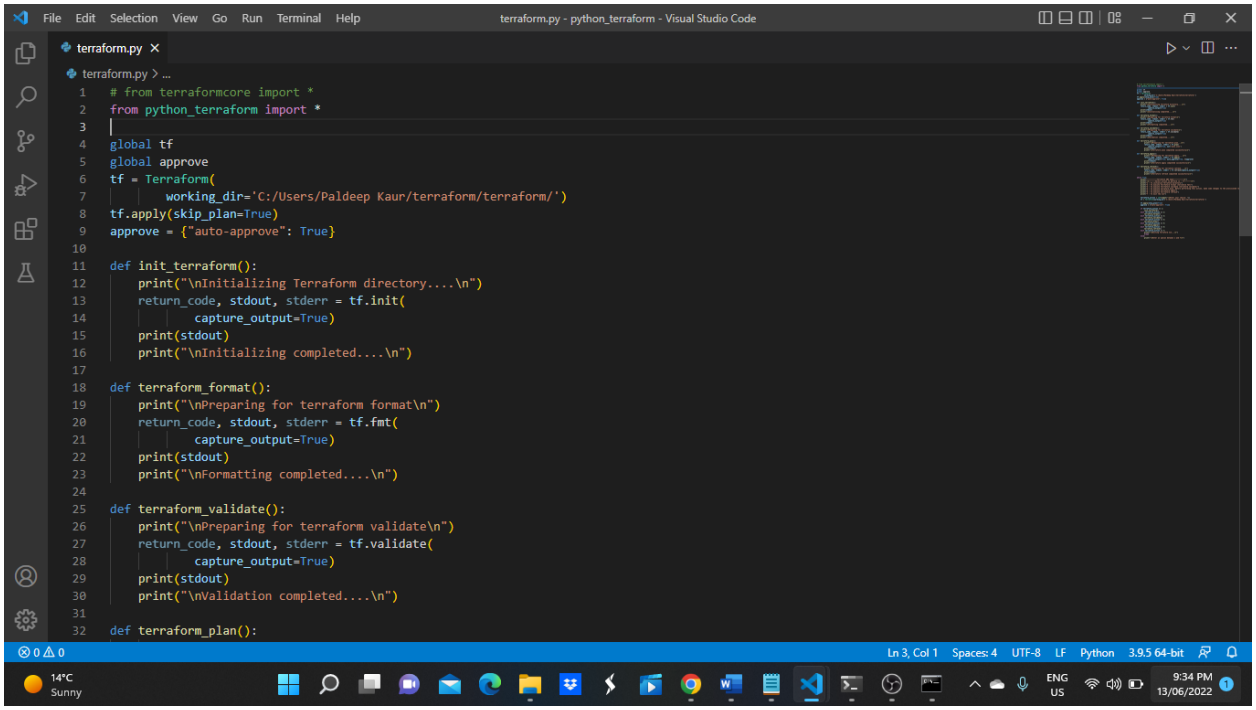
```
provider "aws" {
  profile    = "default"
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
  region     = var.aws_region
}

resource "aws_instance" "terraform_ec2" {
  ami           = lookup(var.ec2_ami, var.aws_region)
  instance_type = var.type
  count         = 1
  tags = {
    Env  = "dev"
    Name = "terraform_ec2_instance-${count.index}"
  }
}

resource "aws_s3_bucket_acl" terraform_s3_bucket {
  bucket = var.bucket_name
  acl    = var.acl_value
}
```

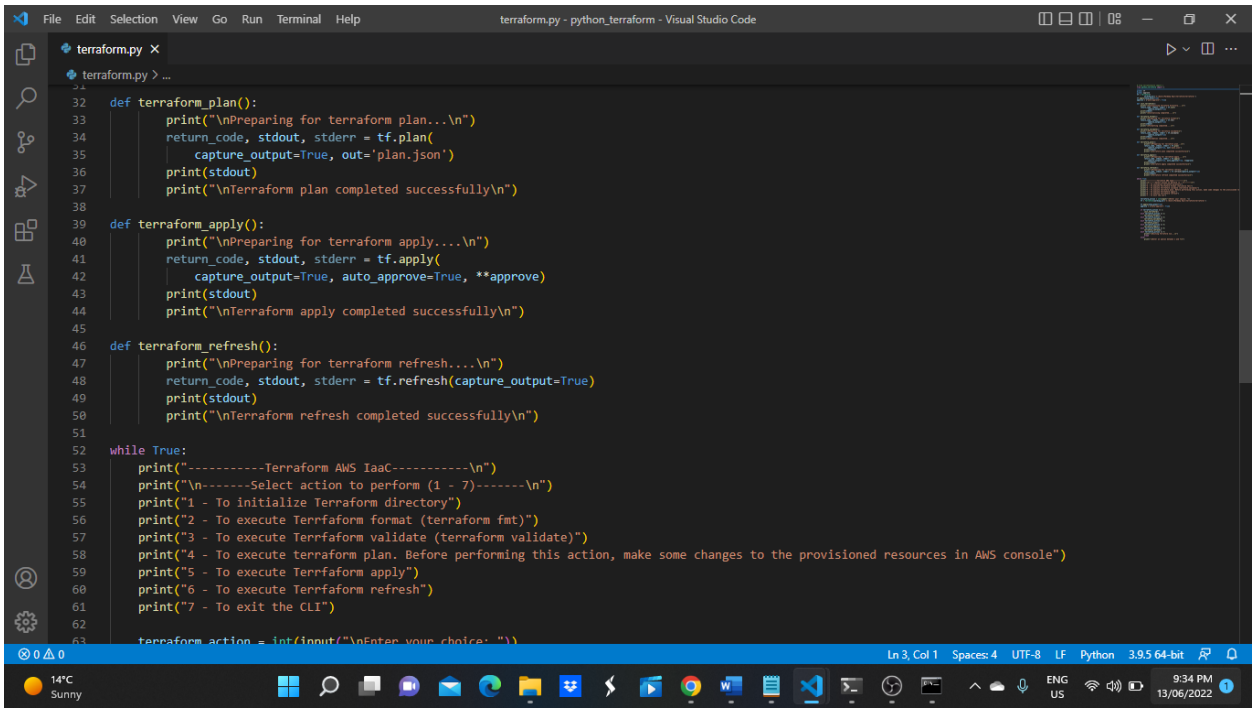


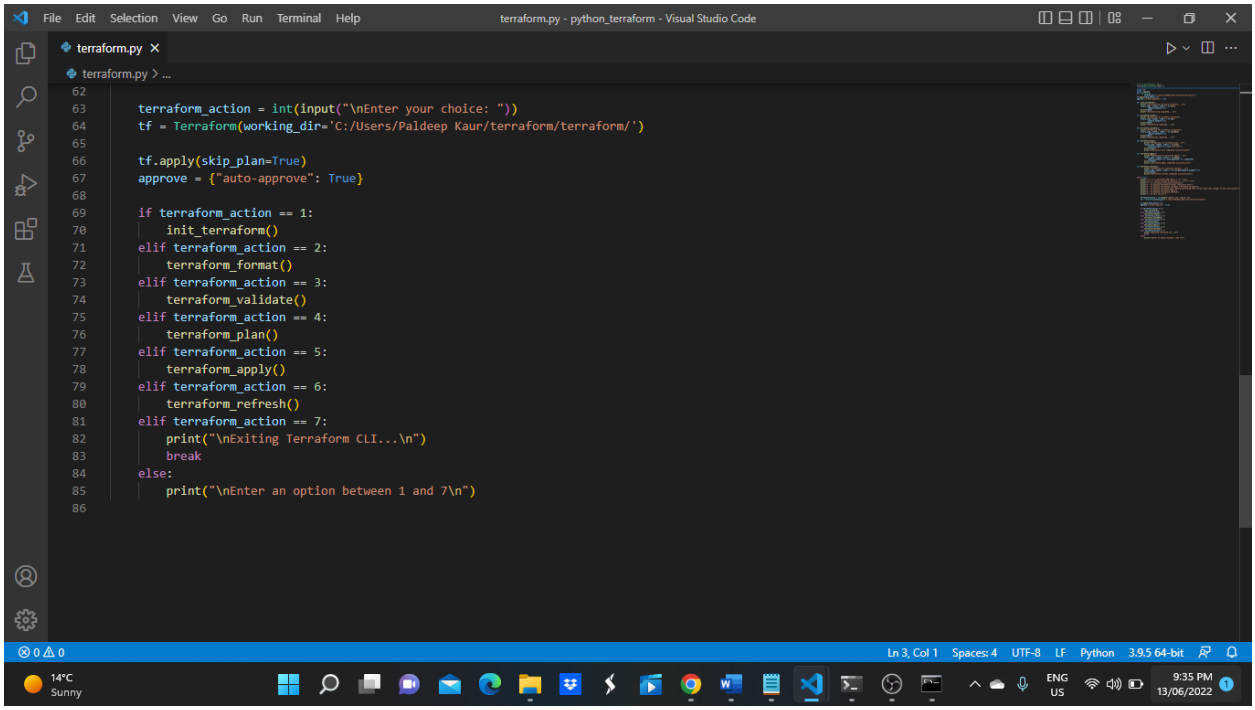
```
variable "aws_access_key" {
  default = [redacted]
}
variable "aws_secret_key" {
  default = [redacted]
}
variable "aws_region" {
  description = "AWS Region"
  type        = string
  default     = "us-east-1"
}

variable "ec2_ami" {
  type = map(any)

  default = {
    us-east-1 = "ami-09d56f8956ab235b3"
  }
}

variable "type" {
  type        = string
  description = "Instance type for the EC2 instance by Terraform"
  default     = "t2.micro"
  sensitive   = true
}

variable "bucket_name" {
  default = "terrform_s3_bucket"
}

variable "acl_value" {
  default = "private"
}
```

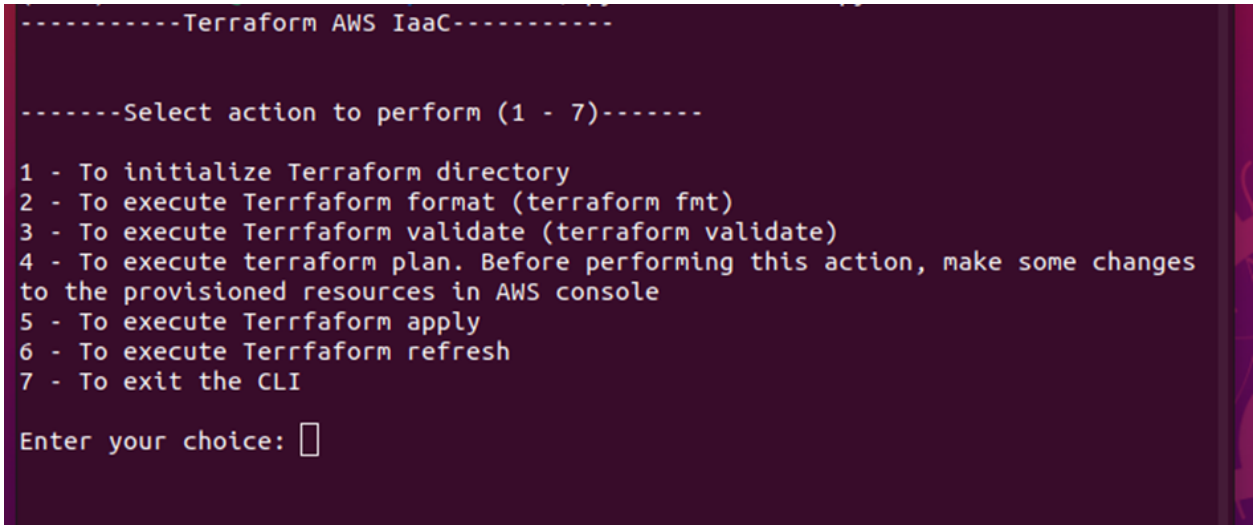
Image 5: Variable.tf

- Python code that contains the scripts for the CLI tool and the necessary functions which are explained below. After setting the two codebases, the CLI tool can be run just like any other Python program. Terraform wrapper is a script that gets checked in with your Terraform project's source code and automatically loads the appropriate version of Terraform for the particular project.


File Edit Selection View Go Run Terminal Help
terraform.py - python_terraform - Visual Studio Code
terraform.py
terraform.py > ...
# from terraformcore import *
from python_terraform import *
global tf
global approve
tf = Terraform(
working_dir='C:/Users/Paldeep Kaur/terraform/terraform/')
tf.apply(skip_plan=True)
approve = {"auto-approve": True}
def init_terraform():
print("\nInitializing Terraform directory....\n")
return_code, stdout, stderr = tf.init(
capture_output=True)
print(stdout)
print("\nInitializing completed....\n")
def terraform_format():
print("\nPreparing for terraform format\n")
return_code, stdout, stderr = tf.fmt(
capture_output=True)
print(stdout)
print("\nFormatting completed....\n")
def terraform_validate():
print("\nPreparing for terraform validate\n")
return_code, stdout, stderr = tf.validate(
capture_output=True)
print(stdout)
print("\nValidation completed....\n")
def terraform_plan():
Ln 3, Col 1 Spaces: 4 UTF-8 LF Python 3.9.5 64-bit
14°C Sunny
ENG US
9:34 PM 13/06/2022


File Edit Selection View Go Run Terminal Help
terraform.py - python_terraform - Visual Studio Code
terraform.py
terraform.py > ...
def terraform_plan():
print("\nPreparing for terraform plan...\n")
return_code, stdout, stderr = tf.plan(
capture_output=True, out='plan.json')
print(stdout)
print("\nTerraform plan completed successfully\n")
def terraform_apply():
print("\nPreparing for terraform apply....\n")
return_code, stdout, stderr = tf.apply(
capture_output=True, auto_approve=True, **approve)
print(stdout)
print("\nTerraform apply completed successfully\n")
def terraform_refresh():
print("\nPreparing for terraform refresh....\n")
return_code, stdout, stderr = tf.refresh(capture_output=True)
print(stdout)
print("\nTerraform refresh completed successfully\n")
while True:
print("----------Terraform AWS IaaC----------\n")
print("\n-------Select action to perform (1 - 7)-------\n")
print("1 - To initialize Terraform directory")
print("2 - To execute Terrfaform format (terraform fmt)")
print("3 - To execute Terrfaform validate (terraform validate)")
print("4 - To execute terraform plan. Before performing this action, make some changes to the provisioned resources in AWS console")
print("5 - To execute Terrfaform apply")
print("6 - To execute Terrfaform refresh")
print("7 - To exit the CLI")
terraform_action = int(input("\nEnter your choice: "))
Ln 3, Col 1 Spaces: 4 UTF-8 LF Python 3.9.5 64-bit
14°C Sunny
ENG US
9:34 PM 13/06/2022





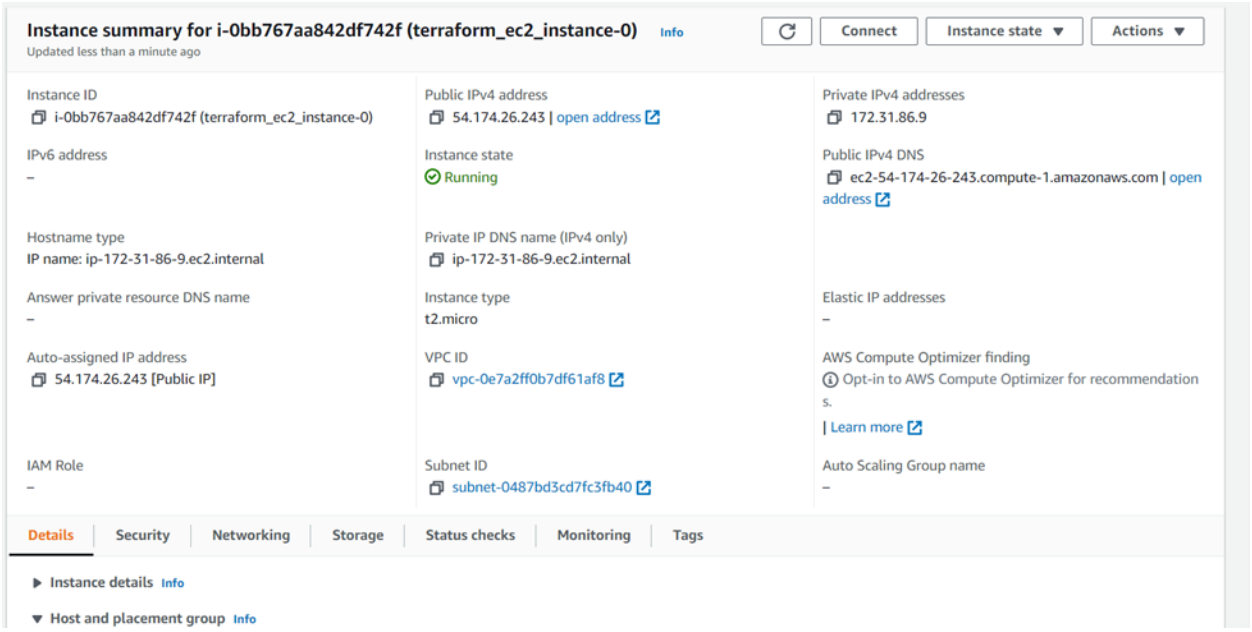
5.1. Initialization of Terraform Directory: For Terraform to work, the working directory must be initialized. This is achieved through a command terraform init. In the CLI, this is represented by 1 in the menu.

```
Enter your choice: 1


Initializing Terraform directory....


Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.15.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.


Initializing completed....
```

5.2. Validation of Terraform Code: Terraform always checks if the code written is valid and it will report errors in the code if it detects them. It is therefore important to validate the code. This is achieved through a function that wraps the terraform validate command. If no errors are found, an output like the one below is shown.

```
Enter your choice: 3

Preparing for terraform validate

Success! The configuration is valid.


Validation completed....
```

5.3. Resources Provision: AWS resources are provisioned using terraform apply function. The CLI tool has a function terraform_plan that generates a plan to be applied to the infrastructure. In addition to the output above, a JSON file (plan.json) is also generated. This file has all the execution plans to be applied by the CLI tool.




```
-------Select action to perform (1 - 7)-------

1 - To initialize Terraform directory
2 - To execute Terrfaform format (terraform fmt)
3 - To execute Terrfaform validate (terraform validate)
4 - To execute terraform plan. Before performing this action, make some changes to the provisioned resources in AWS console
5 - To execute Terrfaform apply
6 - To execute Terrfaform refresh
7 - To exit the CLI

Enter your choice: 4

Preparing for terraform plan...


Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.terraform_ec2[0] will be created
  + resource "aws_instance" "terraform_ec2" {
      + ami                                  = "ami-09d56f8956ab235b3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = (sensitive)
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
```

```
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Env"  = "dev"
          + "Name" = "terraform_ec2_instance-0"
        }
      + tags_all                             = {
          + "Env"  = "dev"
          + "Name" = "terraform_ec2_instance-0"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + capacity_reservation_specification {
          + capacity_reservation_preference = (known after apply)

          + capacity_reservation_target {
              + capacity_reservation_id                = (known after apply)
              + capacity_reservation_resource_group_arn = (known after apply)
            }
        }

      + ebs_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + snapshot_id           = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }

      + enclave_options {
```

```
plan.json  terraform.tf  terraform.tfstate  terraform.tfstate.backup  test.json  variables.tf
```

6.0. Drift Detection

For the resources/services provisioned by Terraform, any change made to a resource through the AWS console or other platforms apart from Terraform is detected as a drift. Examples of these drifts include changing the tag name of an EC2 instance. Terraform keeps two states about infrastructure. These are desired states which hold the configured states of the resources and show how Terraform wants the states of the resources to be. The next state is the current state that shows the real state of the infrastructure. If a change is made to a resource through a platform that is not Terraform, this state will be updated. Terraform then keeps comparing these two states and if it detects any difference, it will report this. These are detected by Terraform, which later tries to revert the changes back to match the desired state.

If we edit the tag name of one of the instances, terraform will detect this and report it as shown below.