Cloud Automation using Terraform (IAC) How-to Guide

By: Shawn Luyke

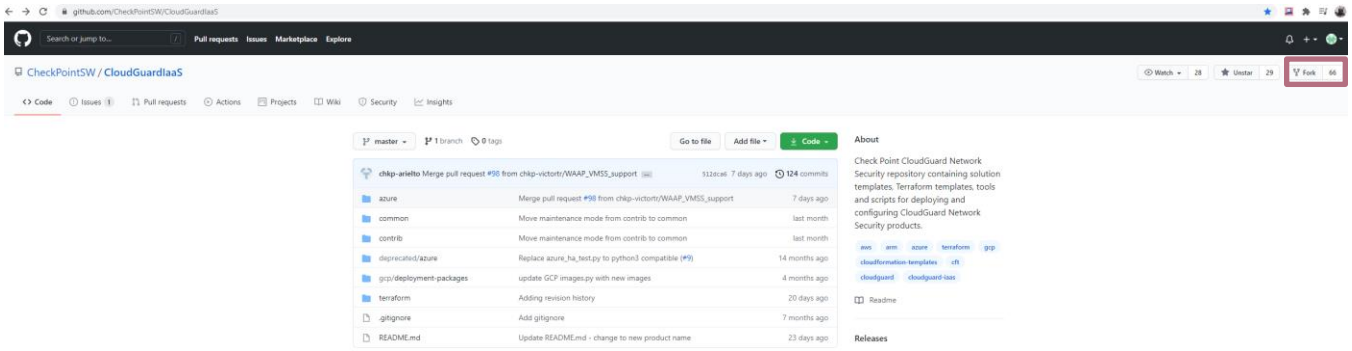
Contents

Overview

The following document will guide you on how to use Github, Terraform, and Visual Studio Code to deploy Check Point Gateways in AWS, Azure, and GCP. The document will also demonstrate how to auto-provision a Check Point Gateway to a Check Point Manager using the cloud controller “CME”

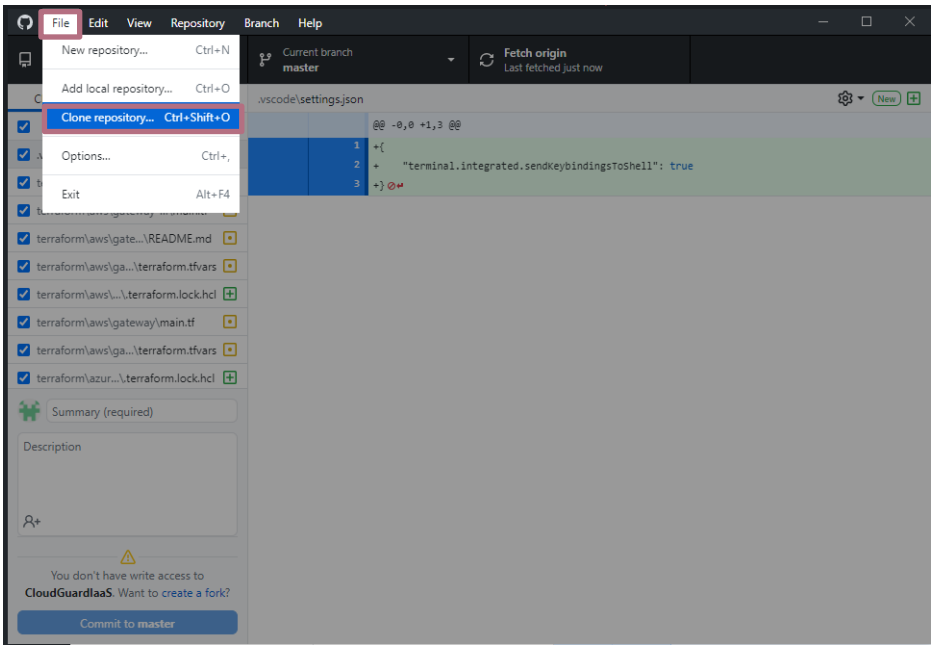
GitHub Repository

- Create yourself a Github account.
- Login and browse to the following URL - https://github.com/CheckPointSW/CloudGuardIaaS
- Once at the CheckPointSW/CloudGuardIaaS, fork the repo to your personal repositories library

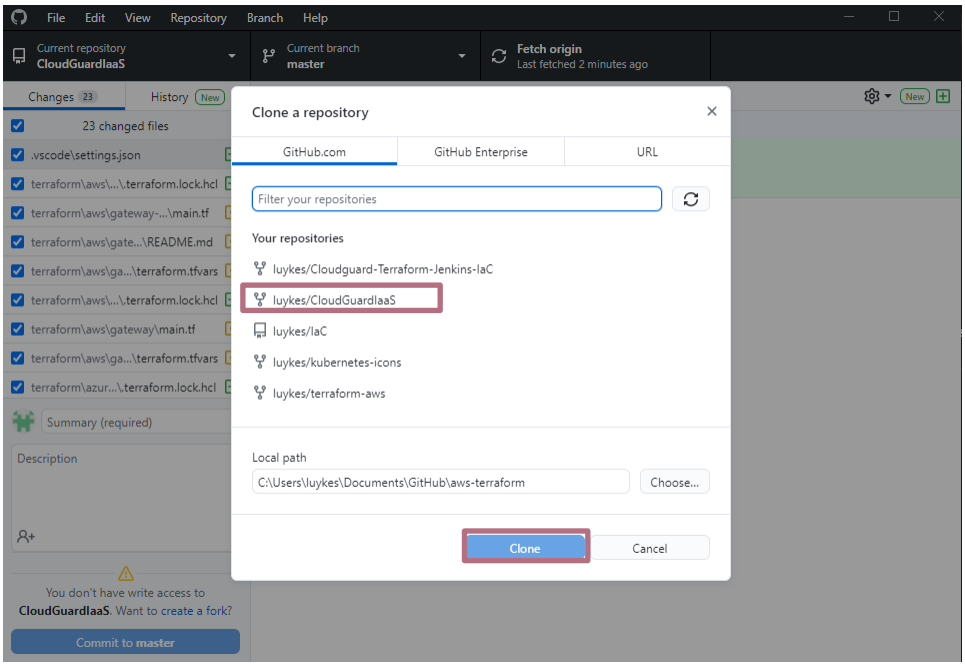


GitHub Desktop

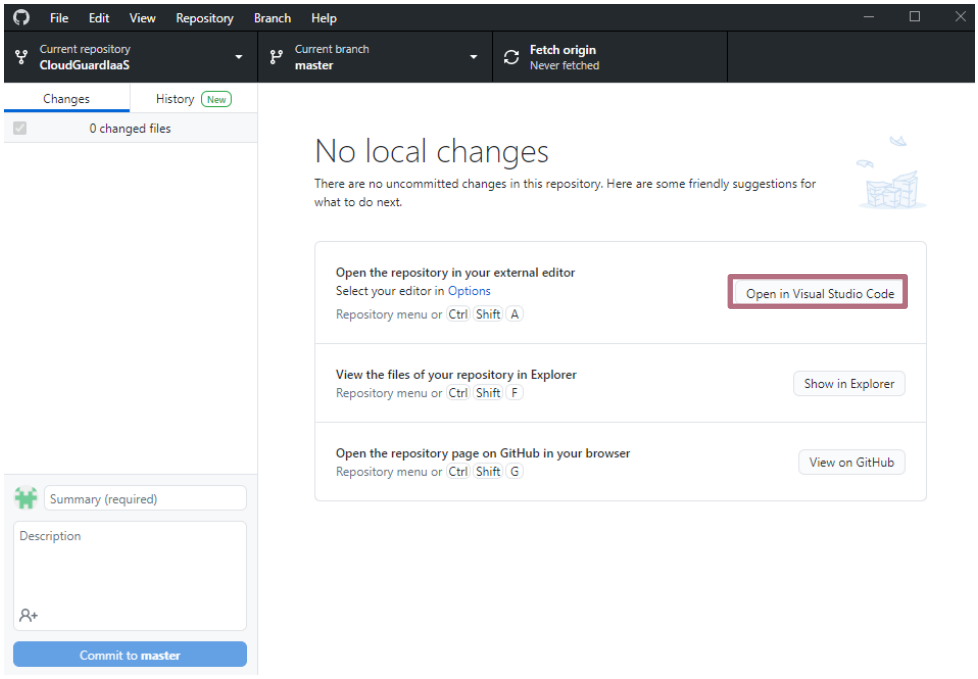
- Download Github Desktop from the following link – https://desktop.github.com/
- Once Github Desktop is installed, log in with your newly created Github credentials.
- Clone the repo you forked in the previous step.



- Select the appropriate repo from the list



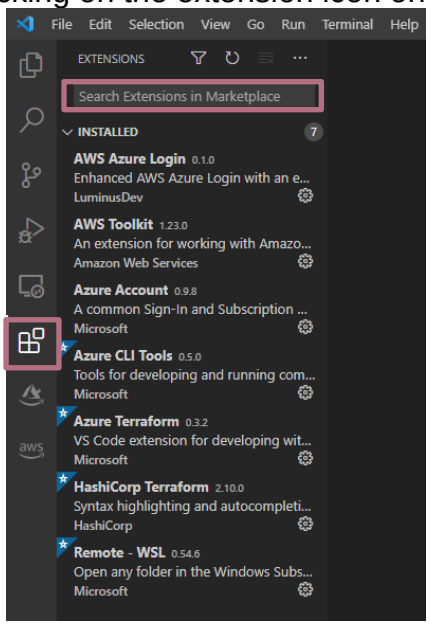
- Once cloning has been completed to your local machine, you can now open GitHub code with Visual Studio Code.



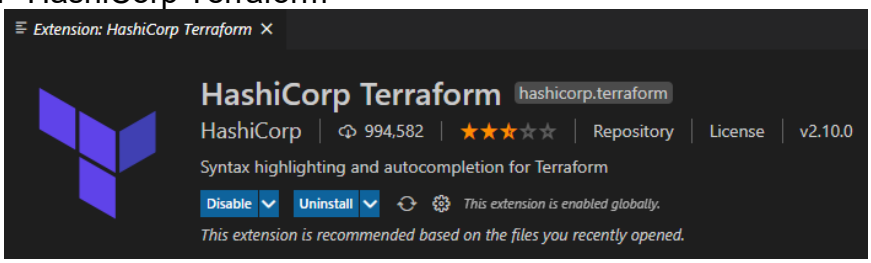
Visual Studio Code

Following packages and add-on needs to be installed on your machine/Visual Studio Code:

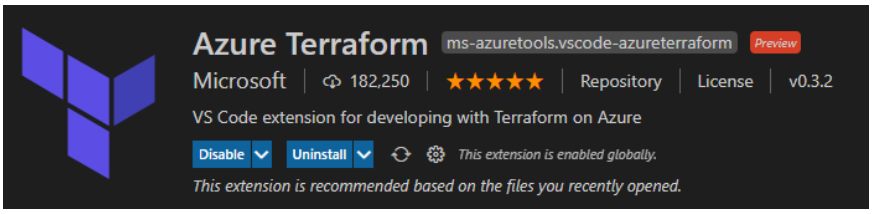
- Terraform will need to be installed on your local machine. Please follow the following link to accomplish this
  - Getting Started with Terraform on Windows: Install, Setup and Demo: https://adamtheautomator.com/terraform-windows/#The_Easy_Way
- Once Terraform is successfully installed the following packages need to be installed within Visual Studio Code.
- Search for an extension by clicking on the extension icon on the left of Visual Studio Code.



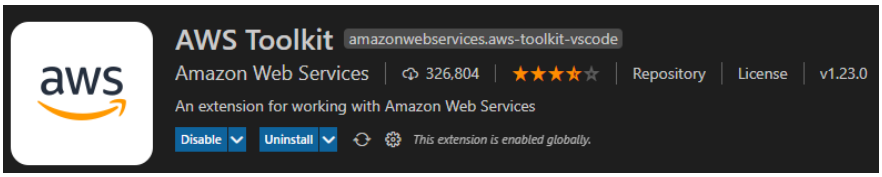
- Search for “HashiCorp Terraform”



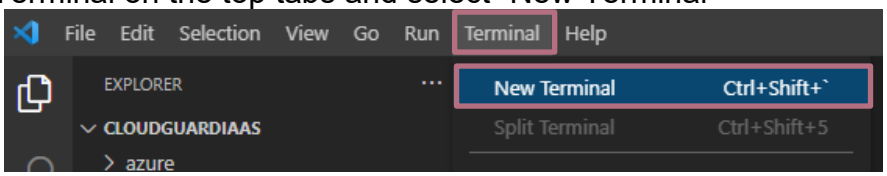
- Search for “Azure Terraform”



- Search for “AWS Toolkit”

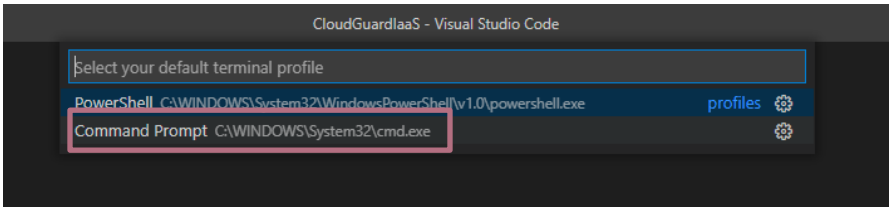


- By default terminal in Terraform use Powershell, this needs to be changed to CMD
  - Click on Terminal on the top tabs and select “New Terminal”



  - A new terminal will appear at the bottom. Left of the terminal screen is a drop-down

  - Open drop-down and select “Select Default Profile” and then choose “cmd: as the default terminal.



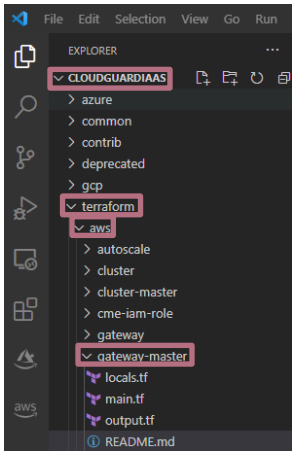
  - NOTE: Visual Studio Code will need to be restarted for this change to take effect.

- Reason: Powershell does not understand “&&” signs. When deploying gateways in a new AWS VPC the”&&” is being used to auto-approve the newly created VPC before it starts deploying the rest of the infrastructure.

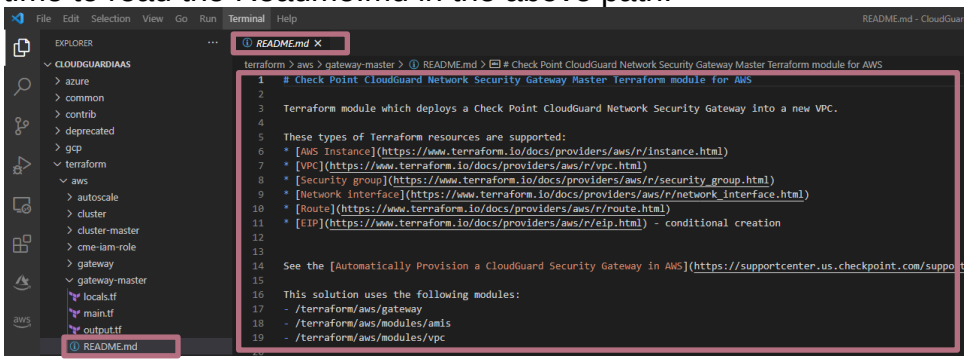
```
- Create or modify the deployment:
    - Due to terraform limitation, the apply command is:
    ```
    terraform apply -target=aws_route_table.private_subnet_rtb -auto-approve && terraform apply
    ```
    >Once terraform is updated, we will update accordingly.
```

Choose your deployment:

- Will use **AWS** as an example: You want to deploy AWS gateways in a New VPC
- Browse to the appropriate file location on the left of Visual Studio Code

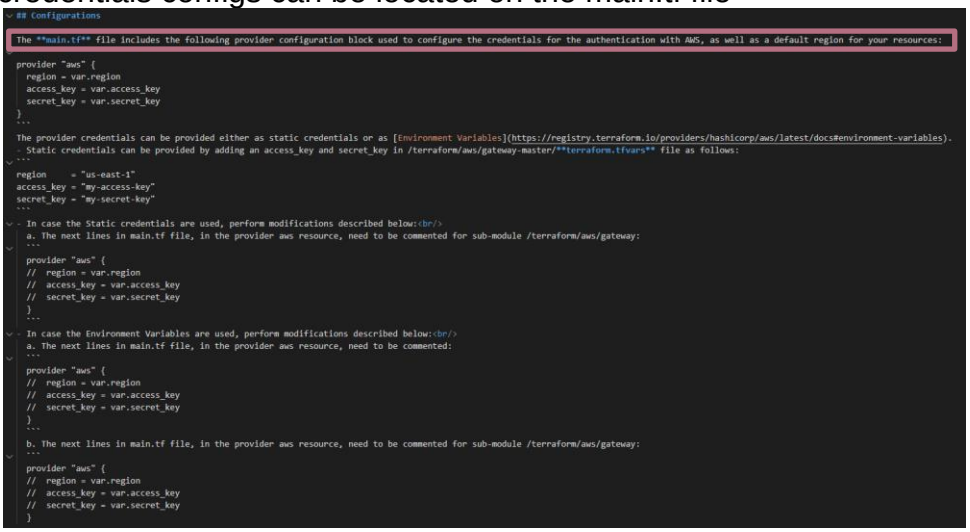


- Take some time to read the Readme.md in the above path.



  - Follow the instructions closely
  - Take note of terraform.tvars inputs in the readme.md

- The root key credentials can be provided in a few different ways within terraform. Please follow the Readme.md for these details.
    - The credentials configs can be located on the main.tf file



Configurations

The **main.tf** file includes the following provider configuration block used to configure the credentials for the authentication with AWS, as well as a default region for your resources:

```
provider "aws" {
  region = var.region
  access_key = var.access_key
  secret_key = var.secret_key
}
```

The provider credentials can be provided either as static credentials or as [Environment Variables](https://registry.terraform.io/providers/hashicorp/aws/latest/docs#environment-variables).
- Static credentials can be provided by adding an access_key and secret_key in /terraform/aws/gateway-master/**terraform.tfvars** file as follows:

```
region     = "us-east-1"
access_key = "my-access-key"
secret_key = "my-secret-key"
```

- In case the Static credentials are used, perform modifications described below:
  a. The next lines in main.tf file, in the provider aws resource, need to be commented for sub-module /terraform/aws/gateway:

```
provider "aws" {
//  region = var.region
//  access_key = var.access_key
//  secret_key = var.secret_key
}
```

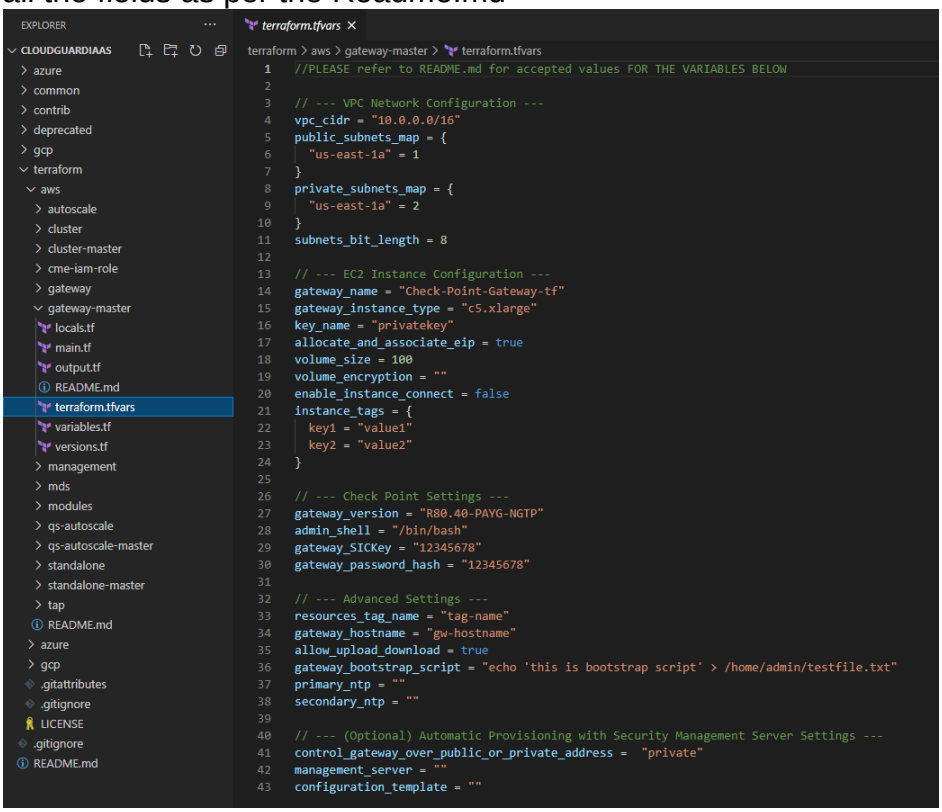
- In case the Environment Variables are used, perform modifications described below:
  a. The next lines in main.tf file, in the provider aws resource, need to be commented:

```
provider "aws" {
//  region = var.region
//  access_key = var.access_key
//  secret_key = var.secret_key
}
```

  b. The next lines in main.tf file, in the provider aws resource, need to be commented for sub-module /terraform/aws/gateway:

```
provider "aws" {
//  region = var.region
//  access_key = var.access_key
//  secret_key = var.secret_key
}
```

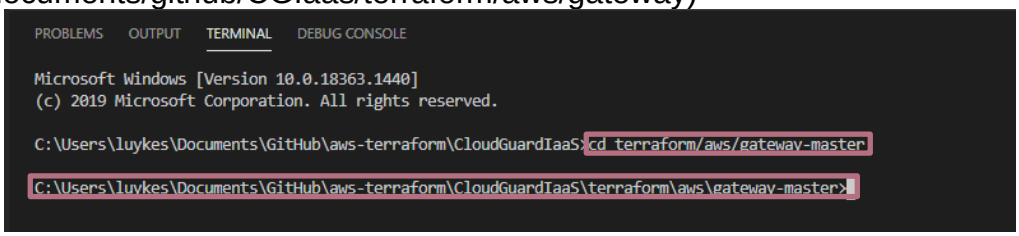
- Then go to terraform.tfvars on the left tab
    - Fill in all the fields as per the Readme.md



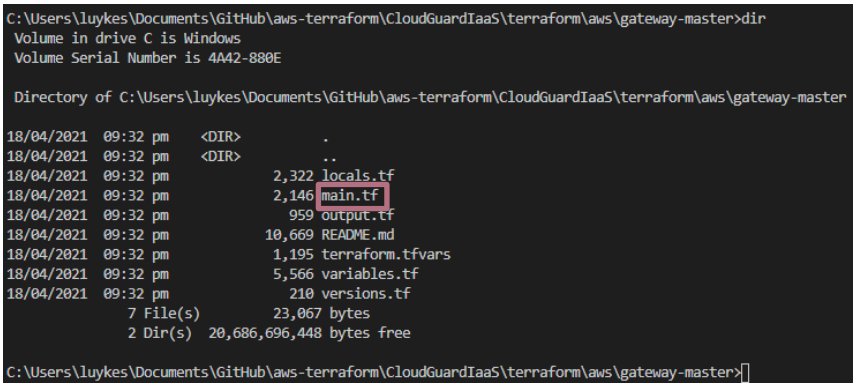
    - Take note of the colors
        - Blue = Variable desroption
        - Orange = Accepted values as per the Readme.md
        - Green = Have been comment out of the code.
        - Some variables are either, “String”, “Number” or “Bool”.
        - String uses inverted commas “ ”. Numbers and Bool do not use “ ”.

Deployment:

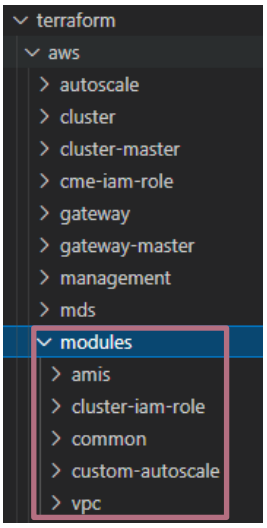
- Now that all the variables have been filled in, now the fun can start.
- Open a new terminal and go to file path of the deployment package (example: cd:/user/documents/github/CGIaas/terraform/aws/gateway)



- When running the directory command all of the following files need to display, especially main.tf



- These terraform scripts rely on modules. The modules need to be part of the entire file directory. (If cloning was done successfully these modules will be part of the folder directory)



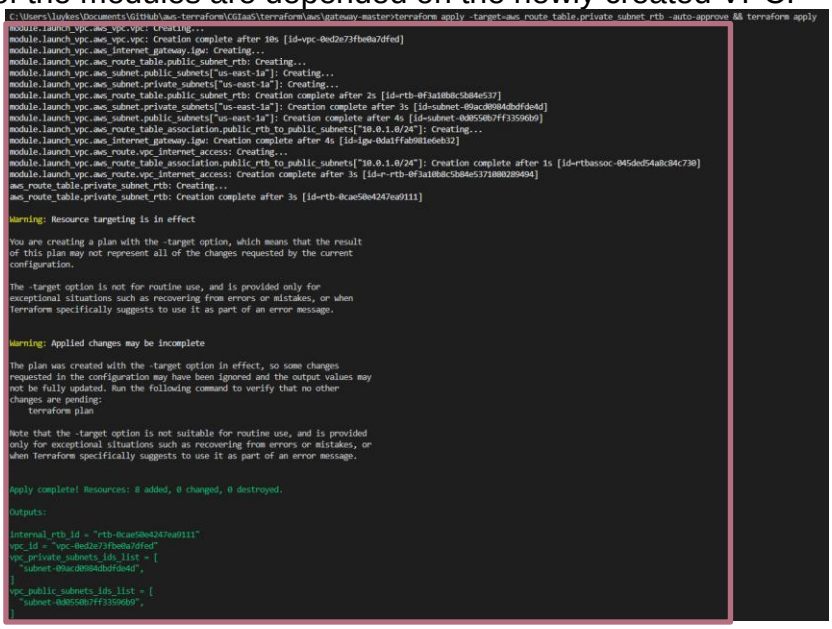
- Now you will need to run “terraform init”. This command will initialise the necessary terraform files and modules that are being used in this script.

```
C:\Users\luykes\Documents\GitHub\aws-terraform\CloudGuardIaaS\terraform\aws\gateway-master>terraform init
Initializing modules...
- launch_gateway_into_vpc in ..\gateway
- launch_gateway_into_vpc.amis in ..\modules\amis
- launch_gateway_into_vpc.common_eip in ..\modules\common\elastic_ip
- launch_gateway_into_vpc.common_gateway_instance in ..\modules\common\gateway_instance
- launch_gateway_into_vpc.common_gateway_instance.validate_instance_type in ..\modules\common\instance_type
- launch_gateway_into_vpc.common_internal_default_route in ..\modules\common\internal_default_route
- launch_gateway_into_vpc.common_permissive_sg in ..\modules\common\permissive_sg
- launch_gateway_into_vpc.validate_gateway_version in ..\modules\common\version_license
- launch_gateway_into_vpc.validate_instance_type in ..\modules\common\instance_type
- launch_vpc in ..\modules\vpc
- validate_gateway_version in ..\modules\common\version_license
- validate_instance_type in ..\modules\common\instance_type

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.24.1"...
- Finding hashicorp/http versions matching "~> 2.0.0"...
- Finding latest version of hashicorp/null...
- Installing hashicorp/aws v3.24.1...
- Installed hashicorp/aws v3.24.1 (signed by HashiCorp)
- Installing hashicorp/http v2.0.0...
- Installed hashicorp/http v2.0.0 (signed by HashiCorp)
- Installing hashicorp/null v3.1.0...
- Installed hashicorp/null v3.1.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.

C:\Users\luykes\Documents\GitHub\aws-terraform\CloudGuardIaaS\terraform\aws\gateway-master>
```

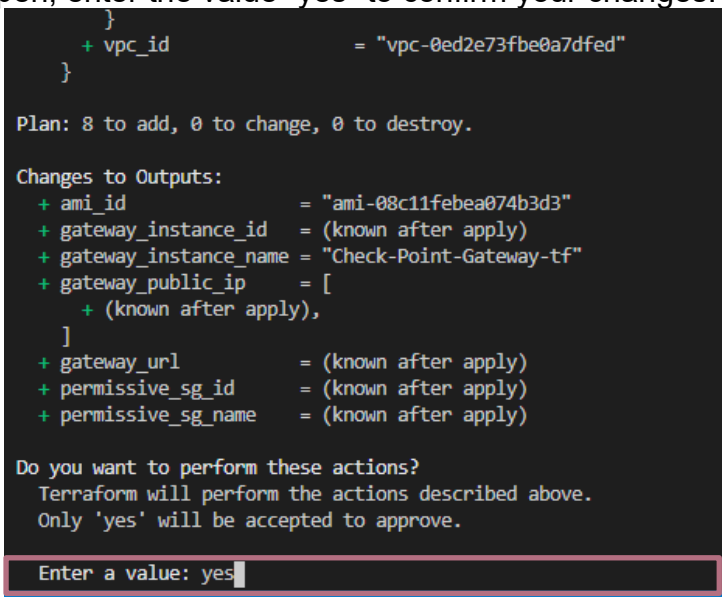
- If all variables have been filled in correctly the “terraform apply” needs to be run. Terraform will now run checks to see if there are any errors within the script.
    - Note: “terraform apply” did not work. This is because of the terraform limitation in being used in this example. Under normal circumstance “terraform plan” will work

```
C:\Users\luykes\Documents\GitHub\aws-terraform\CGIaaS\terraform\aws\gateway-master>terraform apply

Error: Invalid count argument

  on ..\modules\common\internal_default_route\main.tf line 2, in resource "aws_route" "internal_default_route":
   2:   count = local.internal_route_table_condition

The "count" value depends on resource attributes that cannot be determined
until apply, so Terraform cannot predict how many instances will be created.
To work around this, use the -target argument to first apply only the
resources that the count depends on.


C:\Users\luykes\Documents\GitHub\aws-terraform\CGIaaS\terraform\aws\gateway-master>terraform apply -target=aws_route_table.private_subnet_rtb -auto-approve && terraform apply
```

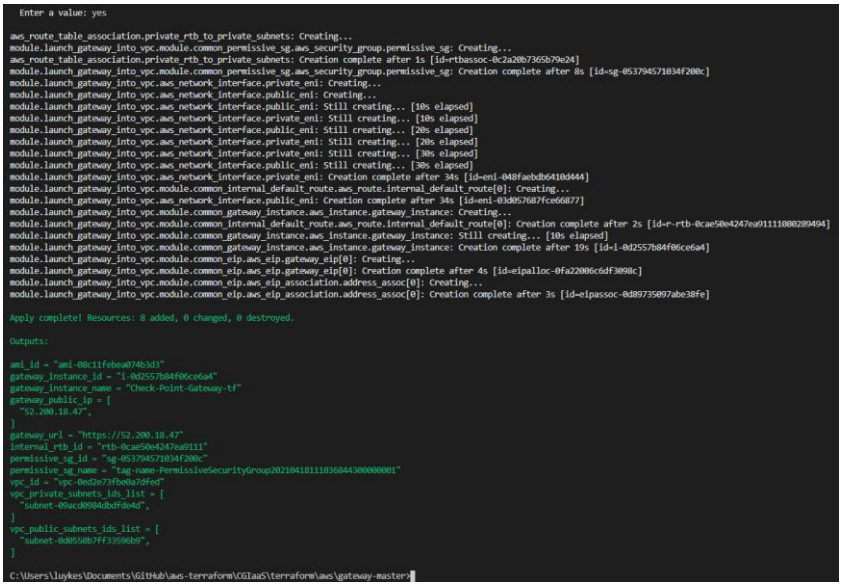

- With the “terraform apply” being run. It will start to deploy the VPC before hand. Once that has been done. Terraform will supply you with an output on whats going to be deployed in the AWS environment.
    - The rest of the modules are depended on the newly created VPC:



    - Terraform will supply you with the following output, when ready and happy with what is going to happen, enter the value “yes” to confirm your changes.



- Now terraform will be deploying and creating the resources into the AWS account as per the credentials specified in the main.tf file.



- Login in to your AWS account to view the newly created Check Point GW's

Auto-provision into an existing manager

On existing manager:

- Install CME package on the manager following SK157492
  - https://supportcenter.checkpoint.com/supportcenter/portal?eventSubmit_doGoviewsolutiondetails=&solutionid=sk157492&partition=Advanced&product=CloudGuard
  - Admin Guide - https://sc1.checkpoint.com/documents/IaaS/WebAdminGuides/EN/CP_CME/Content/Topics-Cloud_Management_Extension_CME/Installing_and_Updating_CME.htm?tocpath=_____3
  - AWS Manager - https://supportcenter.checkpoint.com/supportcenter/portal?eventSubmit_doGoviewsolutiondetails=&solutionid=sk130372#Setting%20up%20Automatic%20Provisioning

- Cloud Controller has been to be configured on the manager.
  - The below command will initialise a controller and template within the manager.
  - *"autoprov_cfg init AWS -mn <MANAGEMENT-NAME> -tn <CONFIGURATION-TEMPLATE-NAME> -otp <SIC-KEY> -ver <VERSION> -po <POLICY-NAME> -cn <CONTROLLER-NAME> -r <AWS-REGIONS> -ak < AKIA**********> -sk < AoG90*********>"*
  - The template will need to be configured. Here we specify which blades need to be enabled for the gateway at hand.
  - *autoprov_cfg set template -tn < CONFIGURATION-TEMPLATE-NAME> -appi -uf -pp 8080 -hi -ips -ab -av -ia*
  - If the CME has been installed and the controller and template have been configured. You should see something similar.


```
[Expert@mgmt-aws:0]# service cme test
Successfully initialized logger for CME SERVICE
run_local Executing: ['/opt/CPcme/menu/additions/autoprov_cfg.py', '--upgrade'] with:
        data=None
        env=None
Return code= 0
        Output=
Testing the configuration file loads...
run_local Executing: ['/opt/CPcme/menu/additions/autoprov_cfg.py', '--dump'] with:
        data=None
        env=None
Return code= 0
        Output={
  "controllers": {
    "AWS-US": {
      *****,
      "class": "AWS",
      "regions": [
        "us-east-1"
      ],
      *****
    },
    "Azure-CP": {
      "class": "Azure",
      "credentials": {
        *****,
        *****,
        "grant_type": "client_credentials",
        "tenant": "23d95250-5e04-4996-a7ef-44c2840a5a90"
      },
      "environment": "AzureCloud",
      "subscription": "bc83fdfa-b1d9-4c54-86e2-b8239d06060c"
    }
  },
  "delay": 30,
  "management": {
    "host": "localhost",
    "name": "R80"
  },
  "templates": {
    "template": {
      "anti-bot": true,
      "anti-virus": true,
      "application-control": true,
      "https-inspection": true,
      "identity-awareness": true,
      "ips": true,
      *****,
      "policy": "Standard",
      "proxy-ports": [
        "8080"
      ],
      "url-filtering": true,
      "version": "R80.40"
    }
  }
}
```

- Now that the controller and template have been configured. You can run “tail -f /var/log/CPcme/cme.log” on the manager. This will display if the manager has seen any new gateways’ in the cloud account.

```
[Expert@mgmt-aws:0]# tail -f /var/log/CPcme/cme.log
2021-04-18 11:40:19,209 CME_SERVICE  INFO No gateways were found in controller Azure-CP
2021-04-18 11:40:19,209 CME_SERVICE  INFO
2021-04-18 11:40:19,209 CME_SERVICE  INFO
2021-04-18 11:40:19,419 CME_SERVICE  INFO There are no gateways known by the management at the end of the iteration
2021-04-18 11:40:19,419 CME_SERVICE  INFO **********    End of the iteration number 13 for gateway instances. Iteration time: 0:00:03.427191    **********
2021-04-18 11:40:19,419 CME_SERVICE  INFO
2021-04-18 11:40:49,444 CME_SERVICE  INFO **********    Starting loop iteration number 14 for gateway instances **********
2021-04-18 11:40:50,039 CME_SERVICE  INFO There are no gateways known by the management at the beginning of the iteration
2021-04-18 11:40:51,188 CME_SERVICE  INFO No gateways were found in controller AWS-US
2021-04-18 11:40:51,188 CME_SERVICE  INFO
2021-04-18 11:40:52,539 CME_SERVICE  INFO No gateways were found in controller Azure-CP
2021-04-18 11:40:52,539 CME_SERVICE  INFO
2021-04-18 11:40:52,539 CME_SERVICE  INFO
2021-04-18 11:40:52,733 CME_SERVICE  INFO There are no gateways known by the management at the end of the iteration
2021-04-18 11:40:52,733 CME_SERVICE  INFO **********    End of the iteration number 14 for gateway instances. Iteration time: 0:00:03.289056    **********
2021-04-18 11:40:52,734 CME_SERVICE  INFO
^C
[Expert@mgmt-aws:0]#
```


Variables in Terraform for Gateways to Auto-deploy:

- For the manager to automate this process, you will need to specify a couple of variables on the terraform.tfvars file in your Visual Studio Code.

```
// --- (Optional) Automatic Provisioning with Security Management Server Settings ---
control_gateway_over_public_or_private_address =  "public"
management_server = "<MANAGEMENT-NAME>"
configuration_template = "<CONFIGURATION-TEMPLATE-NAME>"
```

*"autoprov_cfg init AWS -mn <MANAGEMENT-NAME> -tn <CONFIGURATION-TEMPLATE-NAME> -otp <SIC-KEY> -ver <VERSION> -po <POLICY-NAME> -cn <CONTROLLER-NAME> -r <AWS-REGIONS> -ak < AKIA**********> -sk < AoG90*********>"*

  - Note: The SIC key in the terraform.tfvars need to be the same on the controller configuration of the manager.

```
// --- Check Point Settings ---
gateway_version = "R80.40-PAYG-NGTP"
admin_shell = "/bin/bash"
gateway_SICKey = "12345678"
gateway_password_hash = "12345678"
```

*"autoprov_cfg init AWS -mn <MANAGEMENT-NAME> -tn <CONFIGURATION-TEMPLATE-NAME> -otp <SIC-KEY> -ver <VERSION> -po <POLICY-NAME> -cn <CONTROLLER-NAME> -r <AWS-REGIONS> -ak < AKIA**********> -sk < AoG90*********>"*

Auto-provisioning working

- Small clip of how an AWS gateway being deployed in a NEW VPC and auto-provisioned on an existing manager in AWS
  - https://youtu.be/6c78dENuZAM

