##########Table of content ##############

Overview

User guide to setup terraform on a developer work station, create a project to deploy aws resources using nbn IaC FW and create a jenkins pipeline to deploy aws resources using terraform.

In this user guide, EC2 Terraform modules is used from nbn IaC FW within github to provision an on-demand EC2 instance and a jenkins declarative pipeline is created to automate deployment using CICD framework.

Requirements

SNo Systems

1

Access to github IaC FW Libraries

https://github.com/nbnco/tooling-standardization-iac-terraform-framework

2 Access to jenkins server - in this use case we are using Jenkins97

Steps

Step No Description

1.

Install Terraform in your linux(Ubuntu)

sudo apt-get update && sudo apt-get install -y gnupg software-properties-common

wget -O- https://apt.releases.hashicorp.com/gpg | \ gpg --dearmor | \

sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg

gpg --no-default-keyring \

--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \

--fingerprint

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \

https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | \

sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update

sudo apt-get install terraform

2.

Clone https://github.com/nbnco/tooling-standardization-iac-terraform-framework  in your local directory.

git clone https://github.com/nbnco/tooling-standardization-iac-terraform-framework

3.

Create a new Repo in local environment.

create a new directory

mkdir robina\_terraformusecase chmod 777

you may do this manually as well from visual studio with the options to create new folders etc.

Create a new directory on your local machine where you want to initialize the repository.

Open a terminal or command prompt and navigate to the newly created directory using the `cd` command.

Initialize a new Git repository in the directory by running the following command: "git init "

Create some files or add existing files to the repository using the `git add` command. For example, to add all files in the current directory, you can run: ``` git add . ```

Commit the changes by running the `git commit` command with a commit message explaining the initial commit: ``` git commit -m "Initial commit" ```

Next, you need to create a remote repository on a hosting service like GitHub with the same name as that of this new directory created above. Once created, copy the url of the repo from Github.

Add the remote repository URL as a remote to your local repository using the `git remote add` command. For example, if the remote repository URL is `https://github.com/your-username/your-repo.git`, you can run: ``` git remote add origin <URL of your remote repo> ```

Push the local repository to the remote repository using the `git push` command. For example, to push the `master` branch to the `origin` remote repository, you can run: ``` git push -u origin master ```

After completing these steps, your local repository will be pushed to the remote repository, and you can start collaborating with others by pushing and pulling changes to and from the remote repository.

The command you need to use to initialize a new Git repository on your local machine is: ``` git init ```By running this command in your terminal or command prompt, Git will create a new, empty repository in the current directory.

After initializing the repository, you can add files, make commits, and then push the changes to a remote repository hosted on a service like GitHub, GitLab, or Bitbucket.

4.

Create a feature branch on you local and check it out.

git branch featurebranch1

git checkout featurebranch1

Copy the EC2 module from the standard library https://github.com/nbnco/tooling-standardization-iac-terraform-framework/tree/master/modules/ec2

The main purpose here is to get the ec2 folder available in the new repo so that modifications can be made as per users/consumers requirements.

You may imply any method to copy the folder into the new local repo - such as Create yourself EC2 folder (main.tf,output.tf and variables.tf) if you are comfortable.

Push the changes to github

git add .

git commit -m"added ec2 module in the repo"

git push

5.

Make changes to the main.tf in your repo and ensure the following values are present as per user requirements.

There are README files present in github for each module. The list of inputs/variable available for every module is listed in this README file.

example EC2 module README :- https://github.com/nbnco/tooling-standardization-iac-terraform-framework/blob/master/modules/ec2/README.md

provider "aws" {

region = var.region

}

# Get snapshot ID

data "aws\_ami" "test\_ami" {

most\_recent = true

filter {

name = "tag:Name"

values = ["2022-08 nbn-centos7-cm2base"]

}

}

module "test\_ec2\_on\_demand" {

source = "github.com/nbnco/tooling-standardization-iac-terraform-framework/modules/ec2"

name = "cici-np-helix-test-ec2-on-demand" # makes sure it is unique name

ami\_id = data.aws\_ami.test\_ami.id

instance\_type = var.instance\_type

ssh\_key\_name = var.ssh\_key\_name

security\_groups = var.security\_groups

vpc\_security\_group\_ids = var.vpc\_security\_group\_ids

subnet\_id = var.subnet\_id

availability\_zone = var.availability\_zone

user\_data = var.user\_data

termination\_protection = false

instance\_profile = var.instance\_profile

associate\_public\_ip\_address = var.associate\_public\_ip\_address

# ebs related things

ebs\_name = "cici-np-helix-test-ec2-on-demand-ebs-vol"

ebs\_size\_gib = var.ebs\_size\_gib

ebs\_snapshot\_id = var.ebs\_snapshot\_id

ebs\_type = var.ebs\_type

ebs\_encrypted = var.ebs\_encrypted

ebs\_kms\_key\_arn = var.ebs\_kms\_key\_arn

tags = var.tags

}

use source = "github.com/nbnco/tooling-standardization-iac-terraform-framework/modules/ec2"

in your main.tf(new repo)

You can trim your main.tf to include only the required values. Changes can be made to the variables are per requirements.

Commit these changes into github and push them.

6.

You will need AWS credential setup as environment variables.

We will need the correct role and credentials to access aws. For this there is a assume role pipeline present in jenkins that will generate temporary credentials available for 30 - 40 mins.

You may use your own aws credentials however it would depend on the policies you are assigned and if you can create a EC2 instance for example or not. The best practice is to use a assume-role and work through it.

We can run the job from any of the assume role pipeline and check the output for the credentials and then copy paste them as environment variables in the local environment.

Open jenkins server and run the appropriate assume role pipeline- see the snip in example column on the right.

Copy paste the credentials in your local machine(ubuntu if using linux)

7.

Run Terraform Init command from with in the directory you have main.tf (of the new repo)

cd <path to directory where main.tf is located>

terraform init

8.

Now run :-

terraform plan

terraform validate

terraform apply - this will output the EC2 instance id which can be used to check the instance in aws.

At the end run terraform destroy to kill the EC2 instance which was created for testing purpose.

9.

Now create a unit test using terratest.

Make sure Golang is installed in ubuntu if not then follow the instructions in :-https://golangdocs.com/install-go-linux

Steps terratest description

Create a new folder in your new repo created above

expand

Create a new file in this new terratest folder

The name of file should end with test.go

This is the requirement from go language.

Use the following code in this new test.go file

package test

import (

"testing"

//"github.com/stretchr/testify/assert"

"github.com/gruntwork-io/terratest/modules/terraform"

)

func TestTerraDemo(t \*testing.T){

terraformOptions := terraform.WithDefaultRetryableErrors(t, &terraform.Options{

TerraformDir: "../ec2",

})

defer terraform.Destroy(t, terraformOptions)

terraform.InitAndApply(t, terraformOptions)

}

Import block - we import all the terratest modules required to run the test

Function in the package test should start with Test<userdefined string>

example: func TestTerraDemo(t \*testing.T) << here testing the module imported

terraformOptions is a parameter which is used so that we can retry the attempts to connected to the terraform remote source in case the connection is not stable

TerraformDir: argument is used to define which directory will terratest find main.tf(consumer defined)

We also make use of defer command in front of terraform.Destroy function in order to give enough time to the consumer to look at the created ec2 instance in this case in aws.

terraform.InitAndApply() is then used to start the terraform init and apply commands.

Run the terratest

rk@LTNBNPF36BX4N:~/githubrepos/robina\_terraformusecase$ cd terratest\_demo/

rk@LTNBNPF36BX4N:~/githubrepos/robina\_terraformusecase/terratest\_demo$

rk@LTNBNPF36BX4N:~/githubrepos/robina\_terraformusecase/terratest\_demo$go mod init demo\_test.go

rk@LTNBNPF36BX4N:~/githubrepos/robina\_terraformusecase/terratest\_demo$ go test -v

go mod init demo\_test.go

Above command creates a new `go.mod` file in the current directory. The `go.mod` file is used to manage the dependencies of your Go module. It keeps track of the required packages and their versions, ensuring reproducibility and compatibility in your project.

go test -v

When you run this command in your Go project directory, it will search for test files (files ending with `\_test.go`) and execute all the test functions within those files. The output will include the names of the tests being run and any additional information or logs that you have included in your test functions.

10.

Remember to push this terratest folder along with its file to github for your records, following the similar approach as you did with the new repo

Using Jenkinsfile to run the declarative pipeline

Save the jenkins code below into a new file or you may choose to call the existing file Jenkins file link in github.

Jenkins file explanation and changes required.

Stage

Code Description

Declaring the library

library("cpt-jenkins-shared-library@release-3.X")

This library contains couple of predefined functions used in the jenkins file.

1) utils.enableContainerDotNetRcFileForGitHub([:])

2) aws\_auth.assumeRole()

Side Car Container 'terraform'

agent {

kubernetes {

label 'terraform'

inheritFrom 'default'

defaultContainer 'jnlp'

yaml """

kind: Pod

metadata:

name: jenkins\_pod

namespace: jenkins97

labels:

jenkins: slave

spec:

containers:

- name: terraform

image: cdtools-docker.apro.nbnco.net.au/nbn/hashicorp/terraform:latest

command:

- cat

tty: true

"""

}//end of kubernetes side car container

} //end of agent

In this side car code a kubernetes pod with label 'terraform' is being created.

The container name here is 'terraform' and rest of the code in the stages are run in this container block/

The namespace here is jenkins97 as https://jenkins97.nbnco.net.au server was used to run the code. Please verify the name space according to the jenkins instance being used to run the code.

Image used is the latest one present in nbn's jfrog artifactory pro. You may choose the one available at the time and make necessary changes.

tty: true

The `tty` option stands for "teletype" and it determines whether a TTY (terminal) should be allocated for the container. When `tty` is set to `true`, it allows interactive terminal sessions to be established with the container.

Checkout

stage('checkout') {

steps {

container('terraform') {

checkout([$class: 'GitSCM', branches: [[name: '\*/master']], extensions: [], userRemoteConfigs: [[credentialsId: 'github-pat-cred', url: 'https://github.com/nbnco/robina\_terraformusecase.git']]])

}//end of container

}//end of steps

}end of stage checkout terraform

In this stage we check out the new repo where we have created a copy of EC2 folder.

Please note the credentials to github are important and can be obtained by the jenkins server you are planning to use.

make the necessary changes to the checkout stage to pull the new repo from github that you created in the above steps previously

In the checkout stage code at present  the credentialsId: 'github-pat-cred' for github access via jenkins. Please ensure this is valid at the time.

Terraform version

//stage to check the version of terrafor- this also ensures terraform is installed in the container

stage('terraform version') {

steps {

container('terraform') {

sh ('terraform version')

}//end of container

}//end of steps

}end of stage version terraform

It is advisable to check the terraform installation is completed or not by running any of the terraform commands. We have used terraform version command to check the version inside the terraform container.

Different versions of Terraform may introduce new features, syntax changes, or bug fixes.

It is crucial to ensure that your Terraform configuration files are compatible with the version you are using.

Checking the version helps you identify if any updates or changes are required to your configuration files to work correctly with the specific Terraform version.

Terraform init

stage('init terraform') {

steps {

container('terraform') {

script {

utils.enableContainerDotNetRcFileForGitHub([:]) // function to enable github use the credentials in netrc file

}//end of script

sh('terraform -chdir=ec2/ init')

}//end of container

}//end of steps

}//end of stage init terraform

text

The use of utils.enableContainerDotNetRcFileForGitHub([:]) function present in nbn shared libraries is done here. This function basically writes the github credentials in local netrc file which is used by github to download the source mentioned in main.tf of new repo.

Also note this function demands an environment variable to be setup which will provide the name of jenkins credentialid for github.

environment{

GITHUB\_PAT\_CREDENTIAL\_ID = "github-pat-cred" // environment variable needed for function utils.enableContainerDotNetRcFileForGitHub

} //end of environment

This stage initialises the terraform.

By running `terraform init`, you set up the necessary dependencies and configurations for your Terraform project. It prepares your working directory to be able to plan, apply, and manage your infrastructure using Terraform commands.

Initializes the backend: Terraform uses a backend to store the state of your infrastructure.

The `terraform init` command initializes the backend configuration specified in your Terraform configuration files (typically in a `backend.tf` file).

This includes setting up the remote storage, authentication, and other necessary configurations.https://developer.hashicorp.com/terraform/language/settings/backends/configuration

Downloads provider plugins

Initializes the local state: Terraform maintains a state file that keeps track of the resources it manages. The `terraform init` command initializes the local state, creating the initial state file or verifying the existing one.

Terraform build and destroy

  stage('terraform build and destroy') {

steps {

container('terraform') {

echo "Testing assume role"

script {

aws\_auth.assumeRole('arn:aws:iam::201329364825:role/cici-gocd-agent-terraform-testing-role') {

echo "I'm inside assume role block with this role!"

sh('terraform -chdir=ec2/ plan')

sh('terraform -chdir=ec2/ apply -auto-approve')

sh('sleep 60')

sh('terraform -chdir=ec2/ destroy -auto-approve')

}//eof aws

}//eo script

} //end of container

}//end of steps

} //end of stage

The use of aws\_auth.assumeRole('arn:aws:iam::201329364825:role/cici-gocd-agent-terraform-testing-role')  function present in shared library is done here. This function allows terraform to assume the role 201329364825:role/cici-gocd-agent-terraform-testing-role  required to perform work on aws instance. The role and its values may vary as per the requirements of the job as per the policies set for these roles. In this case the role mentioned allows user to create a EC2 instance.

Terraform build stage has terraform plan and apply commands being used inside aws\_auth.assumeRole(). This is due to the fact that the longevity of the aws credentials prevails only in the scope of this function.

Terraform plan and apply command are run from local checked out directory ec2 as the main.tf is present in this directory.

There is a use of sleep statement here to pause the execution for further checks.

Terraform destroy command is used to destroy the instances created for testing purpose. The user may chose to keep the instance and use the ec2 instance name spun up in the job. See the jenkins output below in this page.