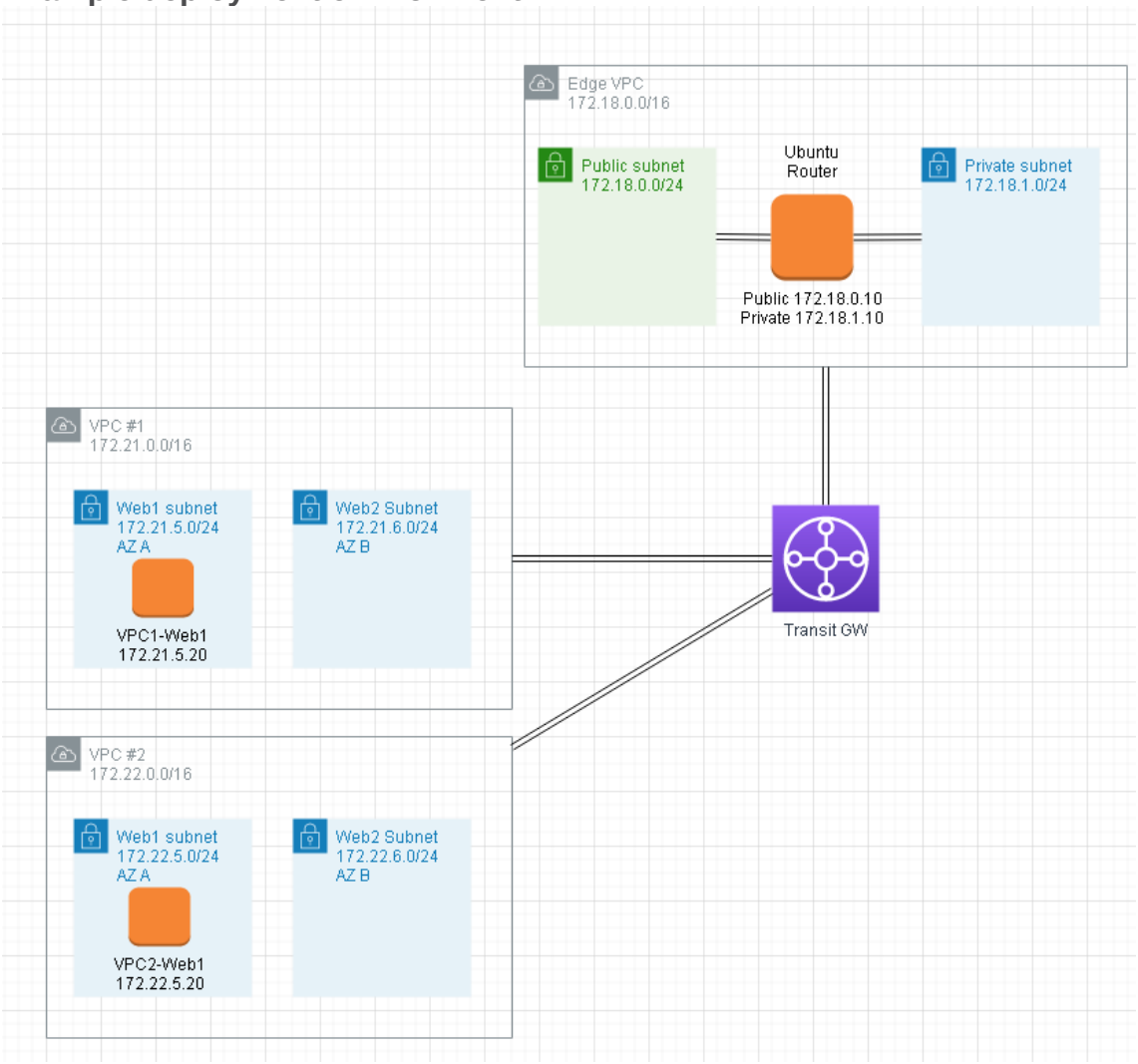
AWS TGW Terraform - Cloud Ready Lab #2

Ryan Darst
Solution Architect
March 26, 2020
Version 1.0

In this lab, we will use a number of technologies to show the ease of use of Terraform to deploy a new Transit Gateway deployment. This deployment will include a Transit Gateway (TGW), a number of VPC's (subnets/route tables/etc), and a few small linux instances for webservers and an Ubuntu router setup on the edge VPC for internet access.

Example deployment environment



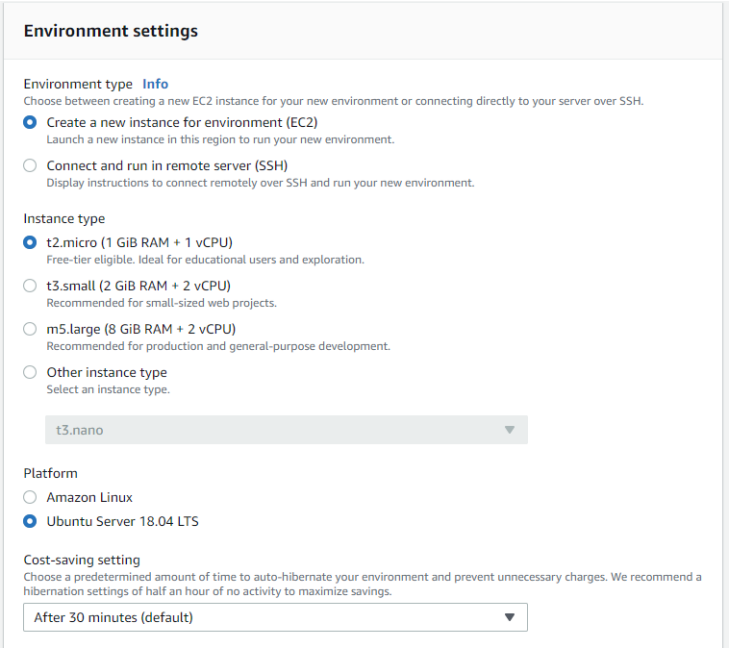
Step 1 – Setup a Cloud9 environment

This step is optional if you already have an AWS CLI environment configured for Terraform in place already. Requirements are to have git/Terraform/aws cli configured for use.

Objective: Setup a consistent environment to launch our configuration. Cloud9 will launch a small Ubuntu instance that we can run commands from and execute Terraform from.

If you would like to setup a new environment to execute your changes from Cloud9 is an easy to setup system. Please follow the steps below to get started.

1. Login to your AWS Account and under services select Cloud9
2. Click the Create environment icon to create your new instance
3. Give your environment a name – In this demo I used Cloud9_Ohio in this demo
4. Use the following options and click the next step button
    - Create a new instance
    - Type – t2.micro
    - Platform – Ubuntu 18.04 LTS
    - Disable after 30 minutes of inactivity



5. Click the next button to create the environment. It will be ready in a few minutes.

Step 2 – Setup Terraform

Objective: Download and Install Terraform on our Cloud9 Instance. Download the TGW template from the public github repo.

1. Review the Cloud9 Interface and take note of the following
    a. The bottom of the screen has a terminal connected to your instance. This instance already has the AWS CLI installed and can be used with the account that was used to setup this Cloud9 instance.
    b. The folders on this instance are available on the left side of the window.
    c. In the center of the screen, files can be edited as needed when selected from the picker on the left.
    d. It is also possible to drag/drop files into the left file navigation area

2. Run the following command to download the Terraform TGW example files and setup script for Terraform. This will pull the files and place them into a new directory based on the repo name.

```
git clone https://github.com/rdarst/Terraform-TGW-Example.git
```

3. Change directory to the new directory

```
cd Terraform-TGW-Example
```

4. Run the following command to download Terraform and have it installed on the Cloud9 Instance. This will move the terraform executable to /usr/local/bin

```
bash setup_terraform.sh
```

5. Execute terraform on the command line and make sure it gives you output.

6. Make sure you have an AWS key setup and know the name of the AWS SSH key. If not, please create a new one in the AWS console under EC2. Also, make sure you have it available so it can be used to access the servers we will launch as part of this lab.

Step 3 – Launch our new environment

Objective: Adjust the Terraform files and launch our new environment.

1. Open the terraform.tfvars file and edit the aws_region and the key_name to match your desired settings. These will be used as part of the launch and determines which data center your resources will live in.

Make sure you have access to the SSH key that is listed so access to the servers will be possible!

2. Run the following command to initialize terraform. This will pull any provider that is required by terraform

```
terraform init
```

3. Now apply the configuration to aws. This will take a few minutes to execute. Run the following command -

```
terraform apply
```

4. Look through the AWS console and review the VPC/EC2 screens. Review the route tables and how they are applied.

5. SSH into the IP address that was provided at the end of the terraform apply. The user name for this user is ubuntu. This is our instance acting as our jump server to the environment and the router in the edge VPC to send traffic from the spokes to the internet. It is a simple Ubuntu instance with ip tables configured.

```
ssh ubuntu@(ip_address_from_terraform_output)
```

Make sure to use your SSH key that you specified as part of your configuration!!!

6. From your jump server make sure you can ping both vpc-web1 and vpc-web2 (These names are setup in your hosts file so they should resolve properly)

7. Now use your workstation and using a web browser access the public that was provided at the end of the terraform deployment and access this ip address and use port 8081 and 8082.
    a. Example http://<public_ip>:8081
    b. Port 8081 should NAT/Map to vpc1-web1
    c. Port 8082 should NAT/Map to vpc2-web1

8. This completes the basic setup and verification of our environment

9. Run the terraform apply again to see if any changes are required.

    a. What did you observe?
    b. How many changes?
    c. How many elements are part of our configuration?
    d. Review the terraform.tfstate file. What does it contain?

Step 4 – Terraform in use

Objective: Use Terraform to make changes to our environment and re-apply the configuration we expect to see based on our configuration.

1. Terminate one of the web server instance using either the console or the AWS CLI from your Cloud9 connection

2. Run terraform apply again and verify that the server is re-built and the instance is available from the internet. Once the instance starts, it should be available within a minute or so.

3. Now we will modify the vpc2.tf file and move the instance to the secondary availability zone. Make modifications to as needed to move the instance to the new subnet and use the ip range that is specified for that AZ as it is listed in the file.

4. Try the terraform apply

    a. What should happen is that 1 instance will be deleted and 1 will be created.
    b. Verify that you can ping the instance from the Ubuntu Jump host.

5. Next use vpc1.tf as an example and build a new spoke VPC based on this file.
    a. Use 172.23.0.0/16 as your VPC ip range.
    b. Use the same ip address format for the new vpc. The Ubuntu Jump Host already has a NAT for port 8083 to point to 172.23.5.20.

    c. Run the terraform apply and build the new vpc

    d. Verify that you can ping the new webserver from the Linux Jump server

    e. Verify you can access it via the web browser on port 8083

Step 5 – Now the fun part

1. Now run terraform destroy
    a. This will destroy everything that was created as part of the TGW example
    b. You may receive an error on the deletion, just run it once more (this I believe is a bug in the dependency logic in Terraform)
    c. Verify all of your resources are gone.

2. Destroy your Cloud9 environment from the AWS console, if you don't plan to use it any longer.

