Week 5 Lab

# Deployment of 2-Tiered Highly Available Web Application with Terraform

## The Goals

In week 5 hands on session, we will further expand our Terraform skills along with improving usability and robustness of our solution.

During our first session, we will create a folder structure that will support multi-environment deployment and generalize our code by creating Terraform module to deploy networking component of our infrastructure.

The created module will initially be stored locally. Next, we will make our module publicly available by using remote git repo. We will discuss the importance of unified naming convention and security considerations related to segregation of prod and non-prod environments.

In our second session, we will move away from the single instance deployment and explore cloud elasticity through introduction of application load balancers and auto-scaling groups.

## Target Architecture

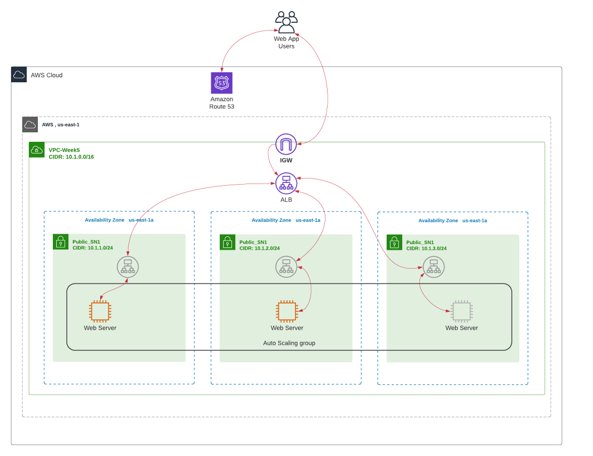


Figure 1 Target Architecture

## Task 1: Multi-environment deployment using Terraform modules

In this task we will introduce local and remote Terraform modules and multi-environment deployment support by further refactoring out solution for the fault tolerant website deployment.

Start with the code provided in BB, Learning Content => Week 5 => Lab =>week4-terraform\_code => Session 1 => week4\_task1\_start. This code is identical to the code we used in Week4, Session 2 to deploy Fault Tolerant web application on EC2 using elastic IP.

1. We will modify the folder structure the support the multi-environment module-based deployment. We will start with one module and 2 environments, dev, and prod. Firstly, we will create a folder structure.

cd Session\ 1/01\_task1\_start/

mkdir modules

mkdir modules/aws\_network

mkdir modules/aws\_webservers

mv 01-Networking/\* modules/aws\_network

ls modules/aws\_network/

mkdir -p PojectWeek5/dev

mkdir -p PojectWeek5/prod

mkdir ProjectWeek5/dev/network

mv 02-Webserver/\* modules/webservers/

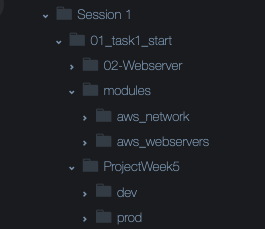


Figure 2 multi-environment folder structure and Terraform modules

1. Secondly, we will convert aws\_network into the module by removing default values and moving config.tf config.tf file to the deployment folder.

mv modules/aws\_network/config.tf ProjectWeek5/dev/network/

Text

Description automatically generated

Figure 3 Variables.tf i aws\_network module

1. Create main.tf, config.tf and variables.tf in **the ProjectWeek5/dev/network/** folder and add reference to the aws\_network module we created.

Text

Description automatically generated

Figure 4 config.tf in ProjectWeek5/dev/network/ folder

Text

Description automatically generated

Figure 5 Main.tf in in ProjectWeek5/dev/network/ folder

Text

Description automatically generated

Figure 6 Variables.tf in ProjectWeek5/dev/network/ folder

Text

Description automatically generated

Figure 7 Output. tf in ProjectWeek5/dev/network/ folder

1. Deploy the networking components by running the terraform init/plan/apply commands.
2. Deployment of the webserver.

Change folder to the webservers and create a new ssh key for this week.

cd ../webservers/

ssh-keygen -t rsa -f week5

Change the configuration to use your S3 bucket in config.tf and main.tf files.

Text

Description automatically generated

Figure 8 config.tf in ProjectWeek5/dev/webservers/ folder

Text

Description automatically generated

Figure 9 main.tf referencing network state in ProjectWeek5/dev/webservers/ folder

Update variables.tf to use the new prefix “week5”.

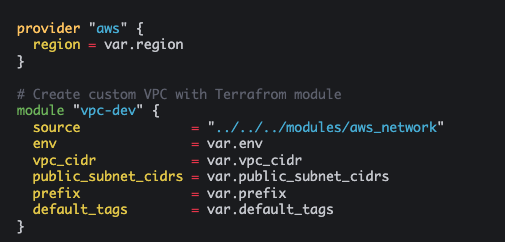


Figure 10 variables.tf in ProjectWeek5/dev/webservers/ folder

Deploy webserver by running terraform init/plan/apply from the **ProjectWeek5/dev/webservers** folder.

1. Destroy the successfully created infrastructure with terraform destroy first from the webservers folder and then from the network folder.

terraform destroy –auto-approve

cd ../network

terraform destroy –auto-approve

1. Create and use remote modules

cd ../../../modules

git init

vi .gitignore # add .git and .terraform to this file

git add.

git commit -m "Adding my module"

1. Create repository in Github and add a remote origin

git remote add origin git@github.com:igeiman/aws\_network.git

# Add public key to the Github settings

git push -u origin master

1. Update the maint.tf of the network deployment to point to the remote Terraform module and deploy the network component again.

cd ../../ProjectWeek5/dev/network/

tf init

tf plan

tf apply

Text

Description automatically generated

Figure 11 main.tf in ProjectWeek5/dev/network/ folder

## Task 2: Refactor the code, use naming conventions, extract global variables, use local variables

In this task we will further refactor our code to introduce naming convention. Naming convention is extremely important in multi-environment cloud deployments and is a corner stone of successful automation.

One of the potential solutions is implemented [in this Terraform module](https://github.com/cloudposse/terraform-null-label). It recommends the naming convention of **{namespace}-{environment}-{stage}-{name}-{attributes}.**

* **namespace**: A short (3-4 letters) abbreviation of the company name, to ensure globally unique IDs for things like S3 buckets
* **tenant**: (Rarely needed) When a company creates a dedicated resource per customer, tenant can be used to identify the customer the resource is dedicated to
* **environment**: A [short abbreviation](https://github.com/cloudposse/terraform-aws-utils/#introduction) for the AWS region hosting the resource, or gbl for resources like IAM roles that have no region
* **stage**: The name or role of the account the resource is for, such as prod or dev
* name: The name of the component that owns the resources, such as eks or rds

We will adopt naming convention of **{namespace}-{stage}-{name}**. For example, VPC deployed into prod environment should have a name tag: **week5-dev-vpc**.

1. Deploy prod and dev environments into different VPCs (Virtua Private Cloud), update resources’ Name tag to reflect our naming convention and verify that the webservers are responding in both environments.

Refactor the code of our Project and our module to adhere to the naming convention.

Deploy network and webserver in “dev” and “prod “environments; verify that the webservers are working

cd ../webservers/

tf plan

tf apply --auto-approve

A picture containing graphical user interface

Description automatically generated

Figure 12 Deployment of webserver in Dev environment

1. Improve readability, use of “templatefile” function in main.tf and make the message dynamic.

cp install\_httpd.sh install\_httpd.sh.tpl

Text

Description automatically generated

Figure 13 install\_httpd.sh.tpl – dynamic resolution of env variable

Text

Description automatically generated

Figure 14 Use of tempfile in main.tf

1. Deploy the updated version of the website

tf plan

tf apply --auto-approve

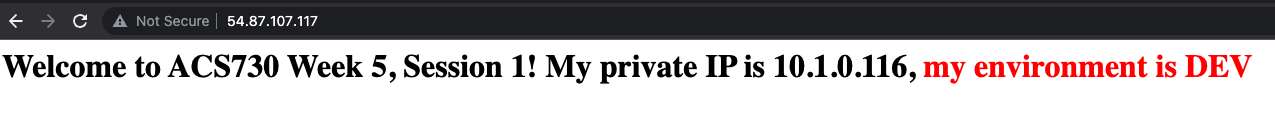


Figure 15 Deployment in Dev using templatefile function of Terraform

1. Deployment of prod environment

We want to deploy our web server into prod environment. We will simulate the prod environment using another VPC (Virtua Private Cloud) in our AWS Academy account.

Delete all the .terraform folders and then duplicate your dev deployment into prod folder.

cd ../..

cp -R dev/\* prod/

cd prod

in the prod subfolders. Create a new bucket to store terraform state. Remember, this is a prod environment, we want a full separation from the dev environment!

Update your network deployment.

Text

Description automatically generated

Figure 16 config.tf in ProjectWeek5/prod/network

Text

Description automatically generated

Figure 17 update input parameters in variables.tf in ProjectWeek5/prod/network

Deploy the network for the production environment.

tf init

tf plan

tf apply --auto-approve

1. Deploy the webserver in the production environment. You might want to create new ssh keys to make sure no one with dev ssh key can access prod servers.

cd ../webservers

ssh-keygen -t rsa -f week5

Update Terraform configurations.

Text

Description automatically generated

Figure 18 config.tf in ProjectWeek5/prod/webservers

Text

Description automatically generated

Figure 19 variables.tf in ProjectWeek5/prod/webservers

Text

Description automatically generated

Figure 20 main.tf in ProjectWeek5/prod/webservers

tf init

tf validate

tf plan

tf apply --auto-approve

The deployment errors out due to repeated Key pair name. The naming convention that we did not define from the start, and it came to affect us now!

1. Update the infrastructure tags to reflect environment name and re-run terraform commands to deploy the code.



Figure 21 webserver deployed into prod environment

1. Abstract the common variables into “globalvars” module, remove repeated code from the “dev” and “prod” variables.
2. Use local variables to improve readability
3. Destroy all the deployed infrastructure. First destroy web servers and proceed to destroying networking.

# Destroy prod environment

tf destroy --auto-approve

cd ../network

tf destroy --auto-approve

# Destroy dev environment

cd ../../dev/webservers

tf destroy --auto-approve

cd ../network

tf destroy --auto-approve

Congratulations! You created a **flexible** deployment of fault tolerant web application that supports multiple environments.

