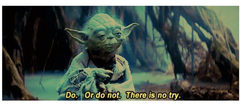
Challenge Instructions

Time to begin your first Challenge of this course! These challenges will help you solidify your skills and flex your brain muscles. It's very important that you take the time to do the Challenges, as they are what will ensure you are able to use Terraform when you're in the real world at your job.

If you get stuck, do not fret. They are supposed to be *challenging*. That's why I called them Challenges!

So go forth and challenge yourself to become better today!



How to solve these challenges:

- Create a **Free Tier AWS Account** and an **IAM use**r with administrative privileges and programmatic access. Use that user in your Terraform configuration to provision the AWS Infrastructure.
- Create a directory for each challenge. Write your solution to one or more Terraform configuration files (.tf files), and then **plan** and/or **apply** the configuration.
- If your solution is not correct, then try to understand the error messages, watch the video again, rewrite the solution, and test it again. Repeat these steps until you get the correct solution.
- **At the end of each exercise destroy the infrastructure to avoid any possible charges that you may incur.**

Challenge #1

- Create a Terraform configuration file with an **AWS provider block**. Set the **region** where the AWS operations will take place to **us-west-1**
- Set the credentials (**access** and **secret** keys) of the IAM user to authenticate to AWS.
- Initialize the provider plugin.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #2

- Provision a new resource of type AWS VPC called **production**
- Set the IPv4 CIDR block for the VPC to **192.168.0.0/24**
- Tag the new resource and use the key-value map **Name = "Production VPC"**
- Run **terraform init**, **terraform plan** and **terraform apply** to provision the VPC.
- Go to the **AWS Management Console -> Select "us-west-1" Region -> VPC Dashboard** and notice that the new VPC was created.
- Change the AWS region to another one and notice that the VPC does not exist in the new region.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #3

- Provision a new **subnet** for the **production** VPC with the local name **webapps**
- Set the IPv4 CIDR block for the subnet to **192.168.0.32/27**
- Set the availability zone for the subnet to **us-west-1b**
- **Tag the new resource and use the key-value map Name = "Web Applications Subnet"**
- Apply the configuration to provision the infrastructure on AWS.
- Go to the **AWS Management Console -> VPC Dashboard** and notice that both the VPC and the subnet were provisioned.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #4

- Consider the following Terraform configuration file which provisions a resource of type AWS VPC and a subnet in the VPC. However, when you want to apply the configuration you get some errors.
- Your job is to identify any errors and change the configuration so that it applies successfully.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #5

- Consider the following Terraform configuration file which provisions a resource of type AWS VPC and a subnet in the VPC. However, when you want to apply the configuration you get some errors.
- Your job is to identify any errors and change the configuration so that it applies successfully.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #6

- Consider the following Terraform configuration file used to provision a resource of type AWS EC2 instance.
- Declare **output values** to print out both the public and the private IP addresses.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #7

- Consider the following Terraform configuration file used to provision **two** identical resources of type AWS EC2 instance.
- Declare **output values** to print out both the public and the private IP addresses of both instances.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #8

- Consider the following partial Terraform configuration.
- Make the necessary changes to **open ports 22, 25, and 80** for the incoming traffic.
- Allow outgoing traffic to all protocols and all destination IP addresses.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #9

- Consider the following partial Terraform configuration.
- Generate an **SSH key pair** for authentication.
- Add the Terraform code that will spin up an EC2 instance in the VPC. Run the latest version of **Amazon Linux 2** or use **amid="ami-04a50faf2a2ec1901"**
- Set the instance type to **t2.micro**
- Associate the **Default Security Group** to the instance.
- Configure the instance to use the **SSH key pair** you've just created for authentication.
- Associate a **public IP** address to the instance.
- Apply the configuration and test that you can connect to the instance using SSH and the key pair. Find the public IP address of the instance on AWS Web Console.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #10

- Consider the solution to the last challenge.
- Add the Terraform code that prints out at the terminal both the public and the private IP addresses of the instance.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #11

- Consider the solution to the last challenge.
- Change the code to start an **Ubuntu Server** instead of Amazon Linux 2. Take the Ubuntu AMI from the AWS Management Console.
- Connect using SSH to the Ubuntu Server using the username called **ubuntu**
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #12

- Change the solution to the last challenge so that Terraform spins up the latest version of the Ubuntu Server. **Use a data source** to fetch the latest version of Ubuntu.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #13

- Consider the solution to the last challenge.
- Add the Terraform code that initializes the server at boot as follows:
- Install the latest **Apache2** web server
- add a new group called **terraform**
- add a new username called **backup-user**. It should belong to **admin** and **terraform** groups.
- Write all the initialization commands in a script.
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.

Challenge #14

- Change the solution to the previous challenge so that only your public IP address is allowed to connect to SSH (port 22). Find out your own public IP address on https://ident.me/
- Test your solution and then destroy the entire infrastructure.

Are you stuck? Do you want to see the solution to this challenge? Click here.