QATIPv3 AWS Lab4
Variables and Workspaces

Contents

Lab Objectives

In this lab you will:

- Deploy resources using a hard-coded terraform file
- Utilize variables to make your code reuseable
- Implement overrides using the command line and tfvars files
- Utilize terraform workspaces to allow multiple simultaneous deployments

Teaching Points

When writing terraform code, it is instinctive to supply explicit values when they are required, for example **name = "michael"**. This can make the code very static though, requiring manual changes of these values each time a modification is needed. A better practice is to use 'placeholders', variables, to which values can be passed at runtime, for example **name = var.username**. Passing values to these variables can be achieved using variable file default values, tfvars files, command line supplied or by being prompted. By default, all changes made to your configuration will be tracked by a single state file. Workspaces allow multiple state files to exist simultaneously so that the same base code, provided with unique variable values, can be deployed. This makes your code flexible and reuseable.

Before you begin

1. Ensure you have completed Lab0 before attempting this lab.
2. In the IDE terminal pane, enter the following commands…

   ```
   cd ~/environment/awslabs/04
   ```

3. This shifts your current working directory to awslabs/labs/04. Ensure all commands are executed in this directory
4. Close any open files and use the Explorer pane to navigate to and open the pre-configured main.tf file in awslabs/04.

Solution

There is no solution code for this lab as it involves multiple deployment phases. Reach out to your instructor if you encounter issues.

Task 1- Review provisioned terraform files

1. Review the provisioned files in awslabs/04..

   main.tf Hard coded deployment of an t2.micro EC2 instances in "**us-west-2**". This represents the type of resource we will create in this lab, in production there would be many other resources defined here. There is also an output block (lines 28-37) which displays information about the deployment, including the workspace.

   variables.tf All content currently commented out

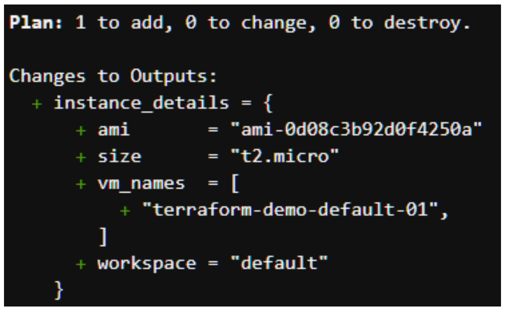
   terraform.tfvars All content currently commented out

   dev.tfvars Variable values that will be used in a new workspace

   prod.tfvars Variable values that will be used in a new workspace

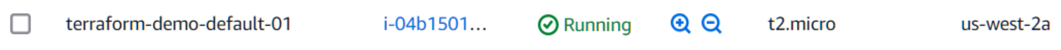
Task 2- Run terraform plan and apply with hardcoded parameter values

1. Ensure you have navigated to the **awslabs/04** folder
2. Run **terraform init**
3. Run **terraform plan**
4. Review the plan output..



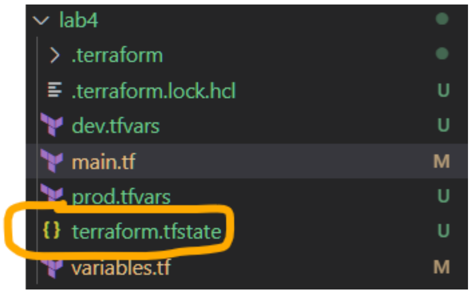
```
Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami       = "ami-0d08c3b92d0f4250a"
      + size      = "t2.micro"
      + vm_names  = [
          + "terraform-demo-default-01",
        ]
      + workspace = "default"
    }
```

5. Run **terraform apply** followed by **yes**
6. Switch to the Console and verify the creation of the EC2 instance in us-west-2 (your availability-zone may differ)..



☐	terraform-demo-default-01	i-04b1501...	Running		t2.micro	us-west-2a

7. Note the creation of a state file in the root folder..



```
lab4
  > .terraform
  .terraform.lock.hcl    U
  dev.tfvars             U
  main.tf                M
  prod.tfvars            U
  terraform.tfstate      U
  variables.tf           M
```

8. **Takeaway**: Hardcoded parameter values will always be used if they exist. This can make the code inflexible and static

Task 3- Substitute hardcoded parameter values with variables

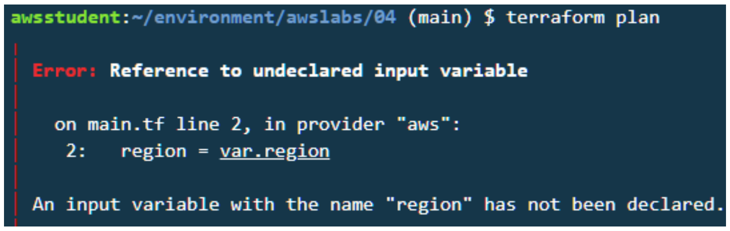
1. In **main.tf**; Replace hard-coded region "us-west-2" with **var.region** (no quotes)
2. Replace hard-coded ami "ami-0d08c3b92d0f4250a" with **var.inst_ami**

3. Replace hard-coded instance_type "t2.micro" with **var.inst_size**

4. Replace hard-coded count 1 with **var.inst_count**

5. Save the changes.

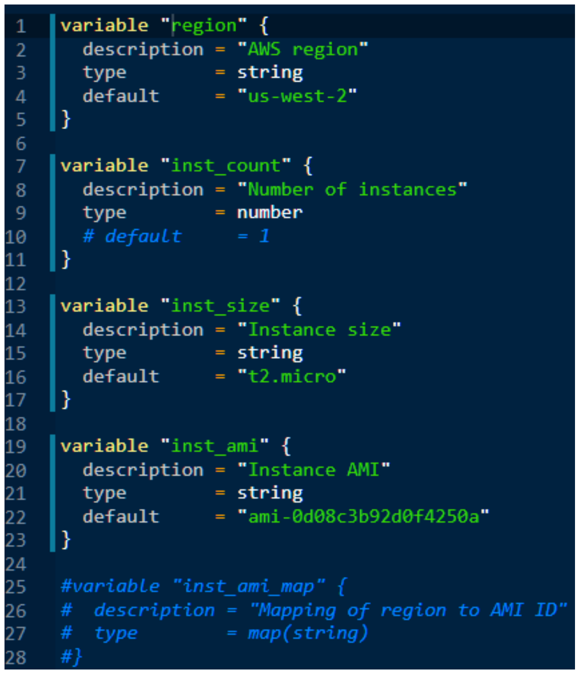
6. Run **terraform plan** and review the error…



```
awsstudent:~/environment/awslabs/04 (main) $ terraform plan
│
│ Error: Reference to undeclared input variable
│
│   on main.tf line 2, in provider "aws":
│    2:   region = var.region
│
│ An input variable with the name "region" has not been declared.
```

7. **Takeaway**: If variables are referenced in your code, then they **must** be declared.

8. In **variables.tf**; uncomment lines 1 through 23 **except** line 10 which provides default values for the inst_count variable. Leave lines 25 to 28 commented out…



```
1  variable "region" {
2    description = "AWS region"
3    type        = string
4    default     = "us-west-2"
5  }
6
7  variable "inst_count" {
8    description = "Number of instances"
9    type        = number
10   # default     = 1
11 }
12
13 variable "inst_size" {
14   description = "Instance size"
15   type        = string
16   default     = "t2.micro"
17 }
18
19 variable "inst_ami" {
20   description = "Instance AMI"
21   type        = string
22   default     = "ami-0d08c3b92d0f4250a"
23 }
24
25 #variable "inst_ami_map" {
26 #  description = "Mapping of region to AMI ID"
27 #  type        = map(string)
28 #}
```

9. Save the changes.

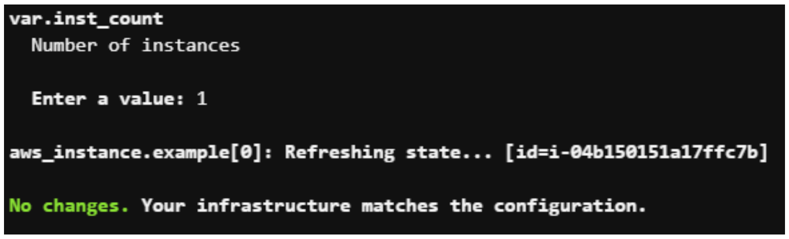
10. Switch to main.tf, uncomment line 32 and save the change

11. Run **terraform plan**

12. When prompted, enter **1** as the number of instances.

13. The first plan run will indicate a change to the proposed output. This is because we have asked for the region to be displayed (line 32).

14. Re-run **terraform plan** and again enter **1** as the number of instances.



```
var.inst_count
  Number of instances

  Enter a value: 1

aws_instance.example[0]: Refreshing state... [id=i-04b150151a17ffc7b]

No changes. Your infrastructure matches the configuration.
```

15. The planning completes using the variable default values if they exist and prompted for when there is no default value. Given that these values are the same as the old static values, the planning phase shows that there are no changes needed

16. In **variables.tf**; uncomment line 10 and save the file

17. Run **terraform plan**

18. The planning completes using all variable values drawn from the variables file. Given that these values are the same as the old static values, the planning phase shows that there are no changes needed.

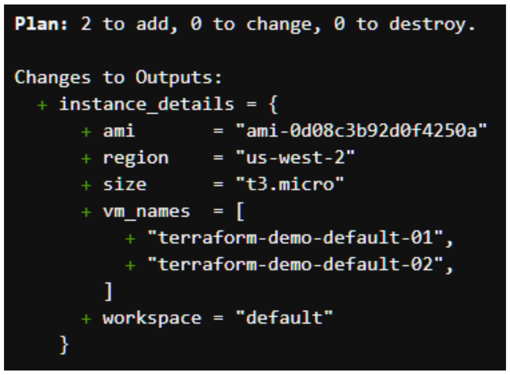
19. Destroy your deployment using **terraform destroy** followed by **yes**

20. Switch to the console and confirm the deletion of the default instance in us-west-2

21. **Takeaway**: If a variable is declared but no value has been assigned then you are prompted for values. If a variable is declared with a default value then this value will be used unless overridden.

Task 4- Overriding variable values at the command prompt

1. Enter the following command..
   terraform plan -var="inst_size=t3.micro" -var="inst_count=2"



```
Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami       = "ami-0d08c3b92d0f4250a"
      + region    = "us-west-2"
      + size      = "t3.micro"
      + vm_names  = [
          + "terraform-demo-default-01",
          + "terraform-demo-default-02",
        ]
      + workspace = "default"
    }
```

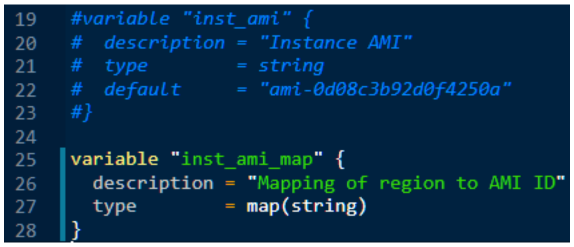
2. **Takeaway**: Supplying variable values at the command prompt using **-var=" "** has the highest priority and overrides values supplied ***anywhere*** else. In this case **t3.micro** is used as the instance type value and **2** instances would be created. The ami and region values are drawn from the variables file default values. Do not apply this deployment.

Task 5- Override variable values using terraform.tfvars

1. Modify line **19** of main.tf from **ami=var.inst_ami** to **ami=var.inst_ami_map[var.region]** and save the changes.

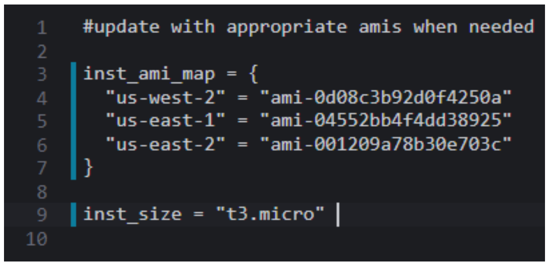
2. This references a new variable **var.inst_ami_map** and removes the use of the variable **var.inst_ami**

3. In **variables.tf**; comment out lines 19 to 23 and uncomment out lines 25 to 28. This removes the **inst_ami** variable and declares the **inst_ami_map** variable. Save the changes.



```
19  #variable "inst_ami" {
20  #  description = "Instance AMI"
21  #  type        = string
22  #  default     = "ami-0d08c3b92d0f4250a"
23  #}
24
25  variable "inst_ami_map" {
26    description = "Mapping of region to AMI ID"
27    type        = map(string)
28  }
```

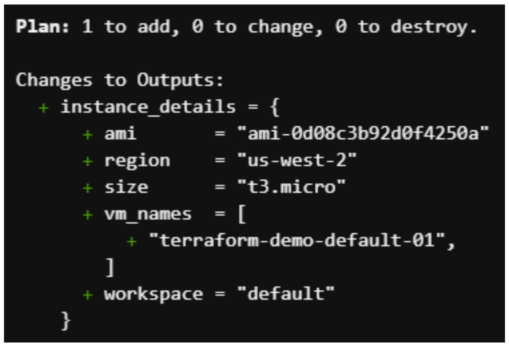
4. Uncomment lines 3 to 9 in **terraform.tfvars**. This supplies values to the **inst_ami_map** variable, allowing the ami relevant to the region to be returned. It also sets the size variable value to t3.micro...



```
 1  #update with appropriate amis when needed
 2
 3  inst_ami_map = {
 4    "us-west-2" = "ami-0d08c3b92d0f4250a"
 5    "us-east-1" = "ami-04552bb4f4dd38925"
 6    "us-east-2" = "ami-001209a78b30e703c"
 7  }
 8
 9  inst_size = "t3.micro"
10
```

Save the changes

5. Run **terraform plan**

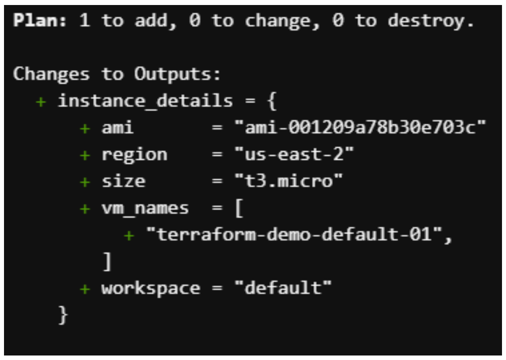


```
Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami       = "ami-0d08c3b92d0f4250a"
      + region    = "us-west-2"
      + size      = "t3.micro"
      + vm_names  = [
          + "terraform-demo-default-01",
        ]
      + workspace = "default"
    }
```

Notice that the size value t3.micro is drawn from terraform.tfvars, overriding the default size in the variables file. When us-west-2 is the region value, the ami ami-0d08c3b92d0f4250a is used.

6. In variables.tf, change the region default to **us-east-2** and save the changes

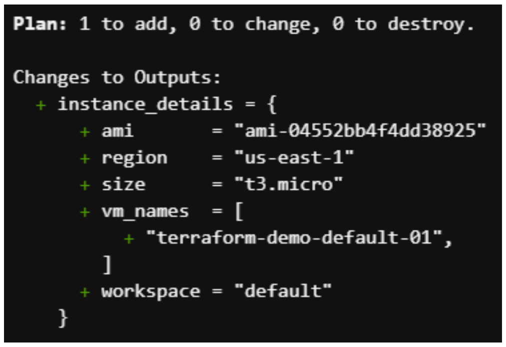
7. Run **terraform plan**



```
Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami       = "ami-001209a78b30e703c"
      + region    = "us-east-2"
      + size      = "t3.micro"
      + vm_names  = [
          + "terraform-demo-default-01",
        ]
      + workspace = "default"
    }
```

Notice that when us-east-2 is the region, ami-001209a78b30e703c is used

8. In variables.tf, change the region default to **us-east-1** and save the changes

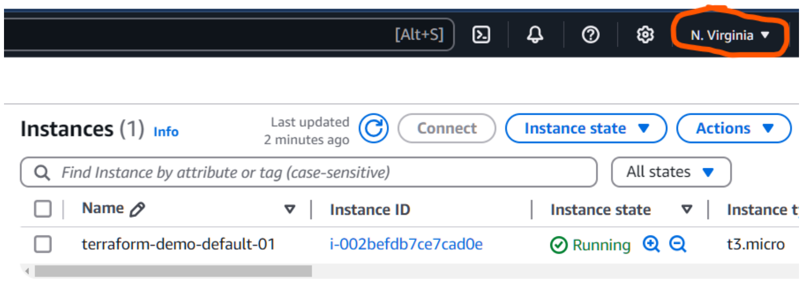


```
Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami        = "ami-04552bb4f4dd38925"
      + region     = "us-east-1"
      + size       = "t3.micro"
      + vm_names   = [
          + "terraform-demo-default-01",
        ]
      + workspace = "default"
    }
```

Notice that when us-east-1 is the region, ami-04552bb4f4dd38925 is used

9. Run **terraform apply** followed by **yes**

10. Switch to the console and verify the creation of the t3.micro instance in us-east-1 (N. Virginia)…



[Alt+S]
N. Virginia
Instances (1) Info
Last updated 2 minutes ago
Connect
Instance state
Actions
Find Instance by attribute or tag (case-sensitive)
All states

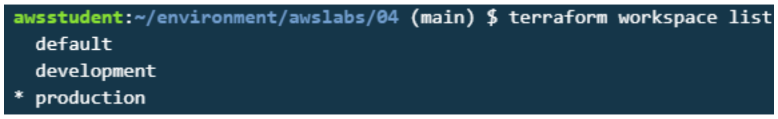
Name	Instance ID	Instance state	Instance t
terraform-demo-default-01	i-002befdb7ce7cad0e	Running	t3.micro

Task 6- Implement Terraform Workspaces

1. Changing deployment parameters will affect the current deployment. Workspaces allow multiple deployments, using the same base code but with different parameters, to exist simultaneously, each with its own state file. If no new workspaces are created then your deployment is in the **default** workspace which always exists and cannot be deleted.

2. Create two workspaces; "**development**" and "**production**"…
   terraform workspace new development
   terraform workspace new production

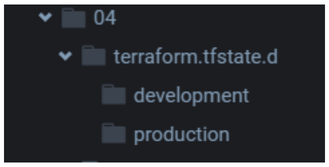
terraform workspace list



```
awsstudent:~/environment/awslabs/04 (main) $ terraform workspace list
  default
  development
* production
```

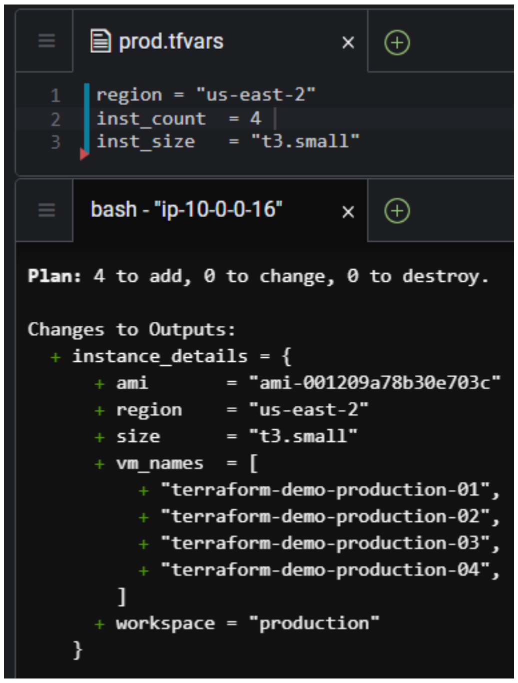
3. The * indicates your current workspace is now **production**. The **default** workspace always exists, and this is where your current deployment of an EC2 instance in us-east-1 is tracked by the state file **terraform.tfstate** in the root folder.

4. Note the creation of a new folder "**terraform.tfstate.d**" with an empty subfolder for each of the new workspaces..



```
04
  terraform.tfstate.d
    development
    production
```

5. Run **terraform plan --var-file=prod.tfvars**



prod.tfvars

```
region = "us-east-2"
inst_count  = 4
inst_size   = "t3.small"
```

bash - "ip-10-0-0-16"

```
Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami         = "ami-001209a78b30e703c"
      + region      = "us-east-2"
      + size        = "t3.small"
      + vm_names    = [
          + "terraform-demo-production-01",
          + "terraform-demo-production-02",
          + "terraform-demo-production-03",
          + "terraform-demo-production-04",
        ]
      + workspace = "production"
    }
```

6. tfvar files other than terraform.tfvars are not referenced unless specified at the command line. Here we have specified inclusion of prod.tfvars which contains region value **"us-east-2",** count value **4** and size value **"t3.small"**

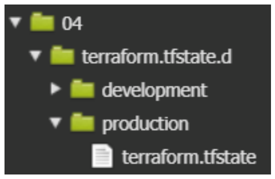
7. If there are conflicts, values from **prod.tfvars** override default variable values and values in the **terraform.tfvars** file. The plan shows that the region, size, and count value are therefore drawn from the **prod.tfvars** file. Using us-east-2 as the region, the ami value is drawn from the inst_ami_map variable values defined in the **terraform.tfvars** file.

8. Notice that the plan will not destroy any resources. Recall that we currently have an EC2 instance deployed in us-east-1. This is in the default workspace and will not be affected as we are now in the production workspace.

9. Run **terraform apply --var-file=prod.tfvars** followed by **yes**

10. Switch to the console and verify the creation of the 4 t3.small instances in us-east-2 (Ohio). Switch to us-east-1 (N. Virginia) to confirm the t3.micro instance created in the default workspace still exists

11. In the IDE, expand the terraform.tfstate.d folder and verify the creation of a new state file for the production workspace..

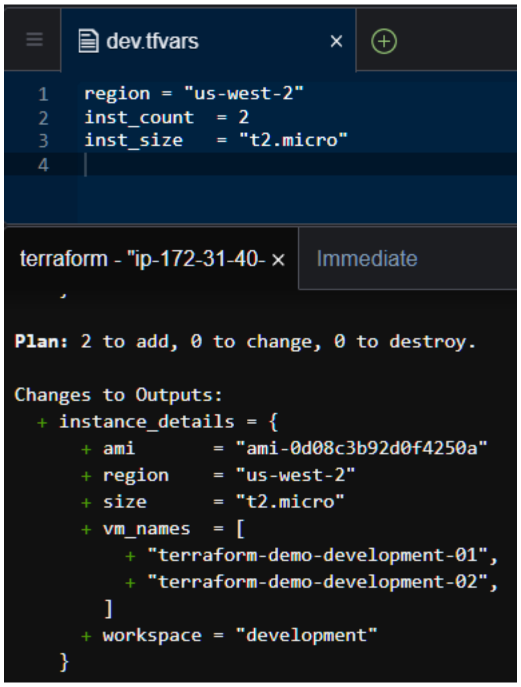


```
▼ 04
  ▼ terraform.tfstate.d
    ▶ development
    ▼ production
        terraform.tfstate
```

12. Move to the development workspace..

    terraform workspace select development

13. Run **terraform plan --var-file=dev.tfvars**



```
region = "us-west-2"
inst_count  = 2
inst_size   = "t2.micro"

```

```
Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_details = {
      + ami       = "ami-0d08c3b92d0f4250a"
      + region    = "us-west-2"
      + size      = "t2.micro"
      + vm_names  = [
          + "terraform-demo-development-01",
          + "terraform-demo-development-02",
        ]
      + workspace = "development"
    }
```

14. tfvar files other than terraform.tfvars are not referenced unless specified at the command line. Here we have specified inclusion of **dev.tfvars** which contains region value **"us-west-2",** count value **2** and size value **"t2.micro"**

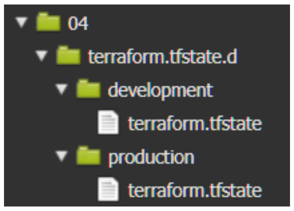
15. If there are conflicts, values from **dev.tfvars** override default variable values and values in the **terraform.tfvars** file. The plan shows that the region, size, and count value are therefore drawn from the **prod.tfvars** file. Using us-west-2 as the region, the ami value is drawn from the inst_ami_map variable values defined in the terraform.tfvars file.

16. Notice that the plan will not destroy any resources. Recall that we now have EC2 instances deployed in us-east-1 and us-east-2. These are in the default and production workspaces respectively and will not be affected as we are now in the **development** workspace.

17. Run **terraform apply --var-file=dev.tfvars** followed by **yes**

18. Switch to the console and verify the creation of the 2 t2.micro development instances in us-west-2 (Oregon). Switch to us-east-1 (N. Virginia) to confirm the t3.micro default instance created in the default workspace still exists. Switch to us-east-2 (Ohio) to confirm the 4 t3.small production instances created in the default workspace still exists

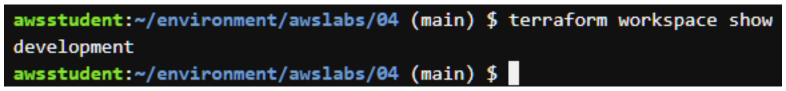
19. In the IDE, expand the terraform.tfstate.d folder and verify the creation of a new state file for the development workspace..



```
▼ 04
  ▼ terraform.tfstate.d
    ▼ development
        terraform.tfstate
    ▼ production
        terraform.tfstate
```

Task 7- Lab Clean-up

1. When deleting resources in workspaces, always pay close attention to the workspace you are currently working in. Use **terraform workspace show** to determine your current workspace..



```
awsstudent:~/environment/awslabs/04 (main) $ terraform workspace show
development
awsstudent:~/environment/awslabs/04 (main) $ 
```

2. Destroy the resources in the current workspace (dev) using..
   terraform destroy --var-file=dev.tfvars

   Note that the tfvars file **must** be specified when performing both apply and destroy actions

3. Enter **yes** when prompted

4. Switch to the Console to confirm the deletion of the 2 development instances in us-west-2. The Cloud9 instance will remain as this is not managed by terraform.

5. The current workspace cannot be deleted. Move to the **production** workspace and delete the **development** workspace…

   terraform workspace select production
   terraform workspace delete development

6. Destroy the resources in the current workspace (production) using..

terraform destroy --var-file=prod.tfvars

Note that the tfvars file **must** be specified when performing both apply and destroy actions

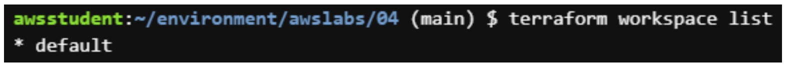
7. Enter **yes** when prompted

8. Switch to the console to confirm the deletion of the 4 production instances in us-east-2

9. The current workspace cannot be deleted. Move to the **default** workspace and delete the **production** workspace…

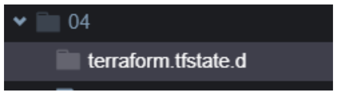
   terraform workspace select default
   terraform workspace delete production

10. Verify the deletion of the workspaces, leaving just the default workspace. This cannot be deleted..
    terraform workspace list



```
awsstudent:~/environment/awslabs/04 (main) $ terraform workspace list
* default
```

11. Note that the workspace directories are deleted along with the workspaces themselves…



```
04
  terraform.tfstate.d
```

12. Destroy the resources in the current workspace (default) using **terraform destroy** followed by **yes**

13. Enter **yes** when prompted

14. Use the console to confirm the deletion of the default instance in us-east-1

## Congratulations, you have completed this lab ##