**Work space**

In **Terraform**, a **workspace** is an isolated environment within the same Terraform configuration, allowing you to manage multiple states for different environments (like development, staging, and production) without duplicating configuration files.

**Key features of Terraform Workspace:**

* **Separate Environments:** Terraform workspaces allow you to create multiple, isolated environments within the *same* Terraform configuration. This is great for managing development, staging, and production infrastructures without having to duplicate your code.
* **Useful for managing multiple environments** – Instead of creating separate directories or repositories, you can use workspaces to manage different environments (e.g., dev, staging, prod).
* **State File Isolation:** Each workspace has its own dedicated state file. This ensures that changes in one environment don't accidentally affect another.
* **Simplified Management:** You can easily switch between workspaces to deploy, modify, or destroy resources in different environments using the same set of Terraform files.
* **Default Workspace:** When you initialize a Terraform project, it starts with a default workspace.
* **Create New Workspaces:** You can create additional workspaces using the “terraform workspace new <workspace\_name>” command.
* **Switch between Workspaces:** Use “terraform workspace select <workspace\_name>” to switch to a different workspace.
* **Apply Changes:** When you run terraform apply, the changes will only affect the currently selected workspace.

**Common Terraform Workspace Commands:**

1. **List all workspaces:**

terraform workspace list

1. **Create a new workspace:**

terraform workspace new <workspace\_name>

1. **Select an existing workspace:**

terraform workspace select <workspace\_name>

1. **Show the current workspace:**

terraform workspace show

1. **Delete a workspace** (except default):

terraform workspace delete <workspace\_name>

**Note:**

✅ Suitable for managing **small-scale** multi-environment deployments.  
❌ Not recommended for **large-scale** production environments—use separate backend configurations instead.

Imagine you are deploying same infrastructure for both **dev** and **prod** environments. Instead of managing separate Terraform configurations file in different directory or folder, we create the workspace, so that in single folder we can create the isolated environment for both **dev** and **prod** environment with separate state file.

Let’s do it practically

#provider Block

provider "azurerm" {

  features {}

  client\_id       = "1f79e427-2ac4-4eb6-9ca0-f4dd4b3f31ee"

  client\_secret   = "Vb18Q~Zj4DMUeKgAiQ-Lpm~pj92y.7s7SLBaAcyG"

  tenant\_id       = "4a623a04-9917-4ee2-8f59-02586964c992"

  subscription\_id = "51c6d184-6756-4a9a-ade4-cd0f3d57cded"

}

**Fig:** main.tf file

#resource group

resource "azurerm\_resource\_group" "rg" {

    name = var.rg-name

    location = var.rg-location

}

#virtual network creation

resource "azurerm\_virtual\_network" "TFVnet" {

    name = var.Vnet-name

    location = azurerm\_resource\_group.rg.location

    resource\_group\_name = azurerm\_resource\_group.rg.name

    address\_space = var.Vnet-address\_space

}

**Fig:** resource block.

variable "rg-name" {

}

variable "rg-location" {

}

variable "Vnet-name" {

}

variable "Vnet-address\_space" {

}

**Fig:** variable block file.

rg-name = "dev-RG01"

rg-location = "eastus"

Vnet-name = "dev-Vnet01"

Vnet-address\_space = ["10.0.0.0/16"]

**Fig:** dev.tfvars.

rg-name = "prod-RG02"

rg-location = "westus"

Vnet-name = "prod-Vnet02"

Vnet-address\_space = ["10.1.0.0/16"]

**Fig:** prod.tfvars

The below figure shows the block diagram of our task:

dev.tfvars

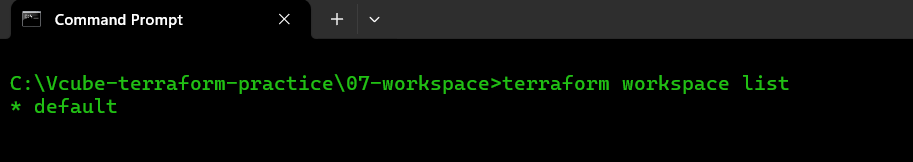
prod.tfvars

main.tf

resource.tf

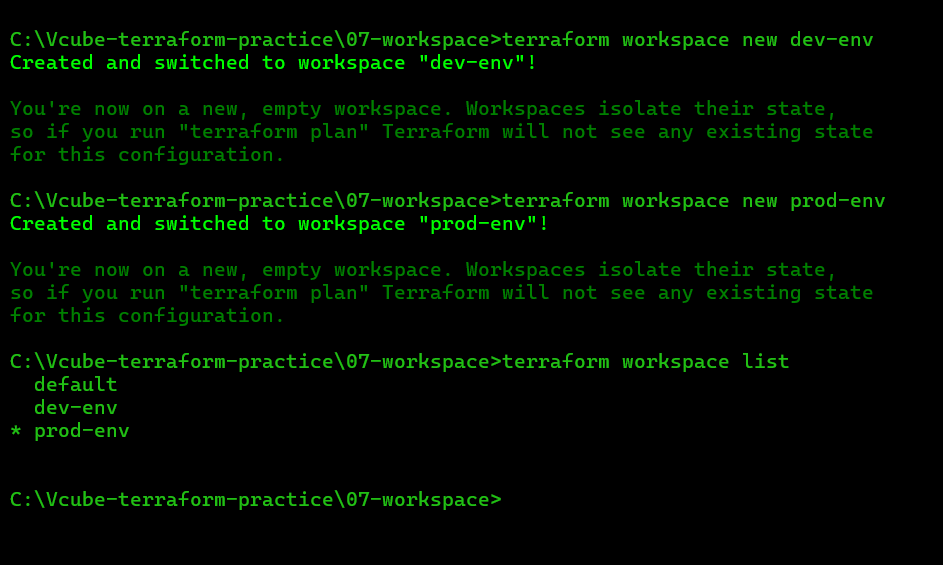
variable.tf

By default we only have single default workspace as shown below figure.

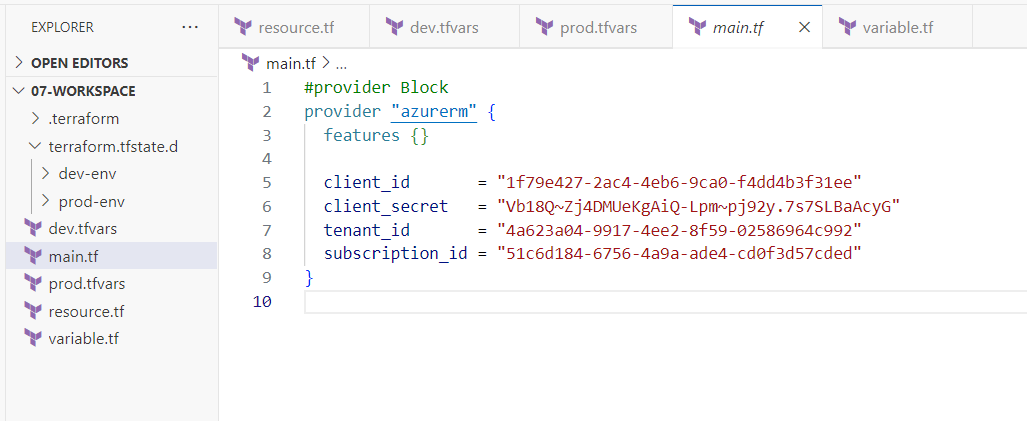


**Step1:** Now create the workspace for both development environment (dev-env) and

product environment (prod-env).

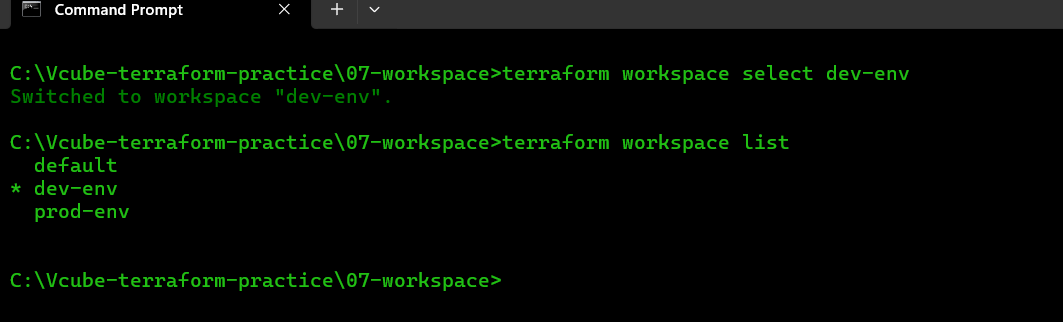


When we create the two workspaces (dev & prod) then a new folder or directory is created with name **“terraform.tfstate.d”,** within this folder we have to directories with name of **dev-env & prod-evn** as show in the below figure.



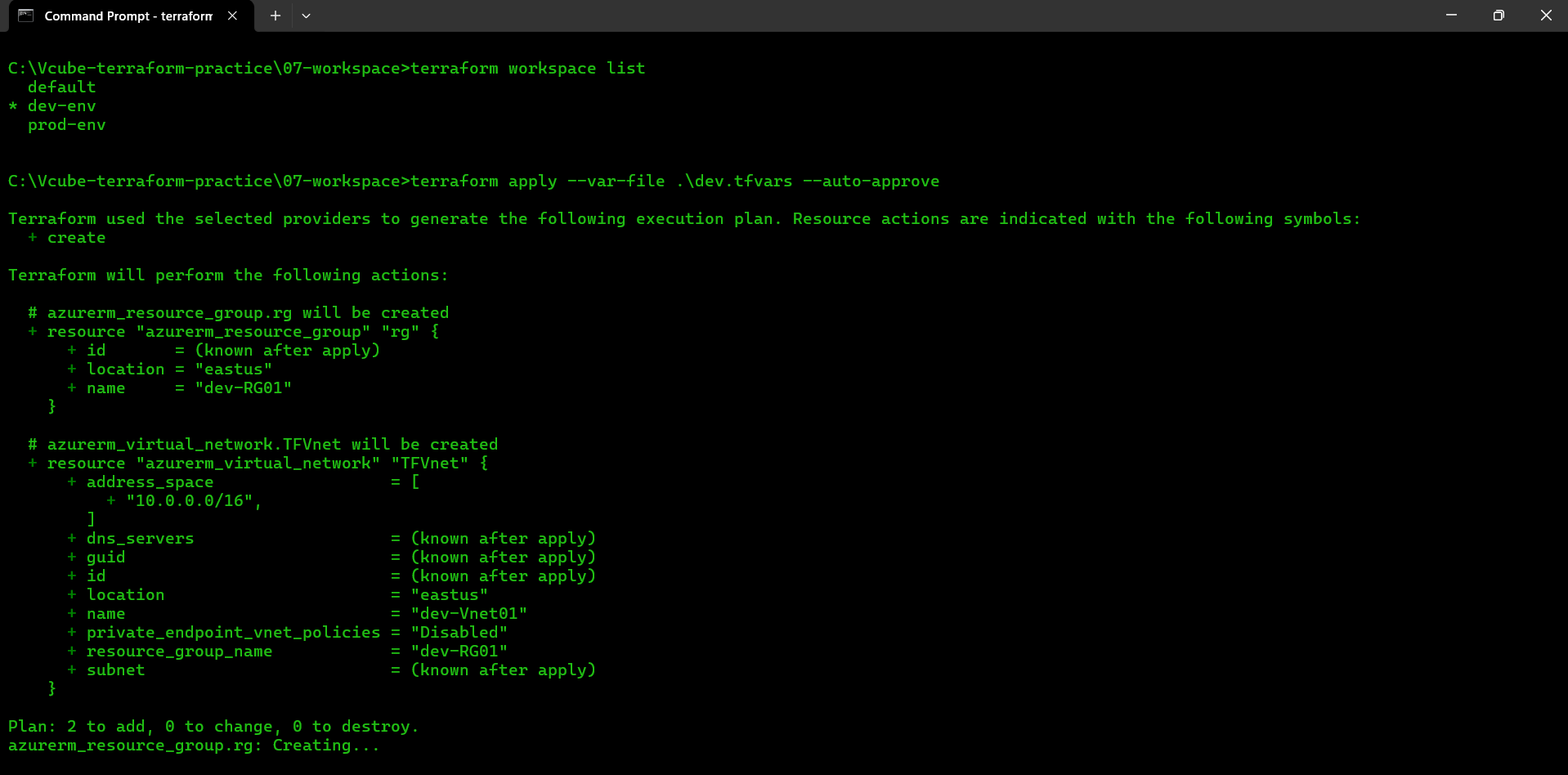
**Step2:** Switch to the development environment (dev-env) and do terraform apply.

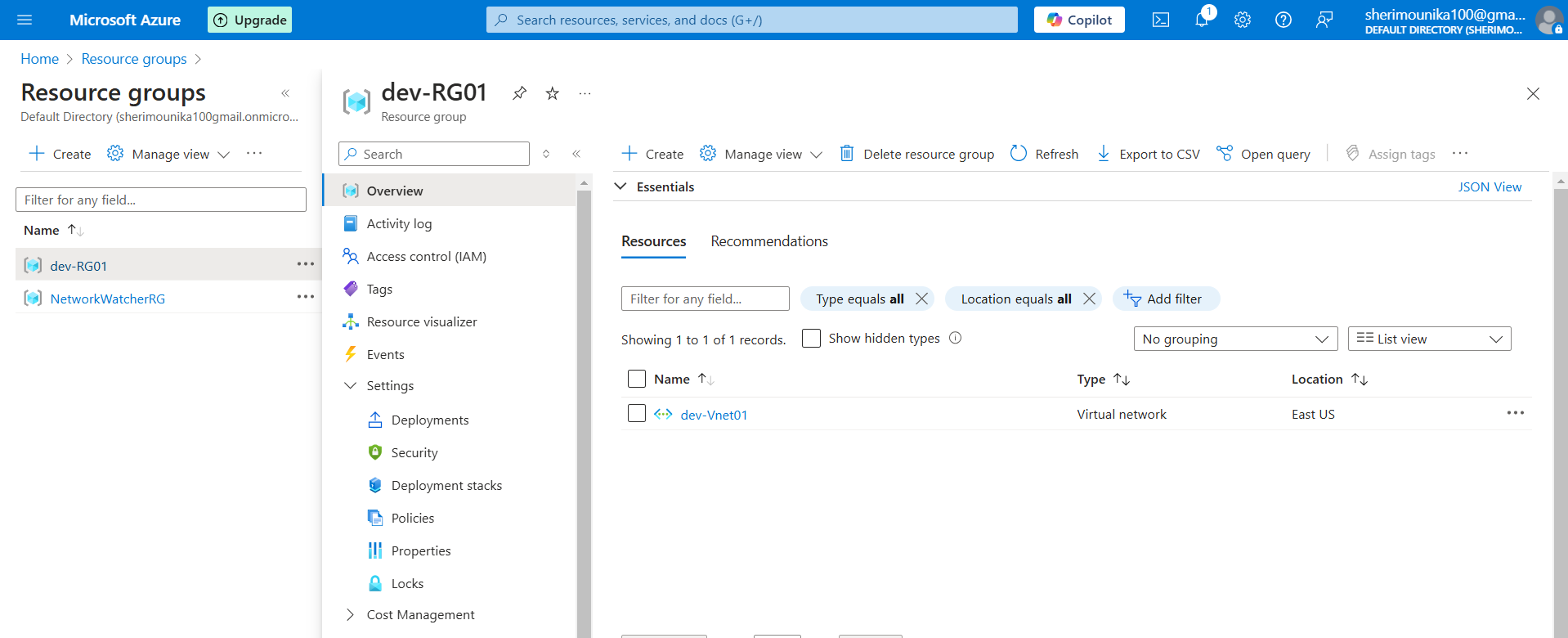
By using terraform workspace select <workspace-name> we can switch between the two workspaces (dev-env & prod-env). In below figure **‘\*’** indicates the current workspace on which we are working.



Now let’s build the infrastructure from standing on dev-env workspace.

The command is terraform destroy --var-file .\dev.tfvars --auto-approve

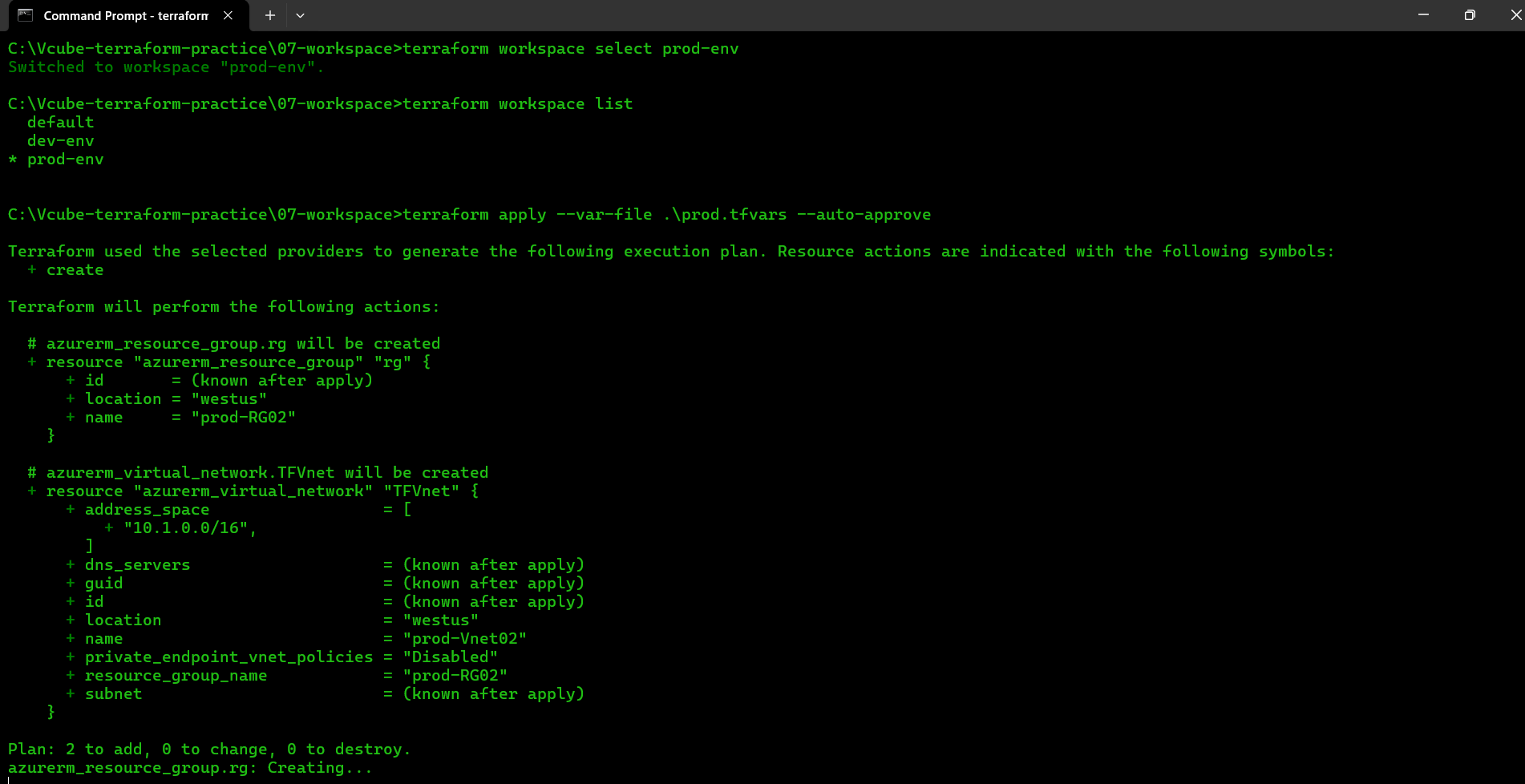




**Fig:** resource group and virtual network of **“dev-env”** is created using work space.

**Step3:** Now switch to the production environment and do terraform apply.

The command is terraform apply --var-file .\prod.tfvars --auto-approve.



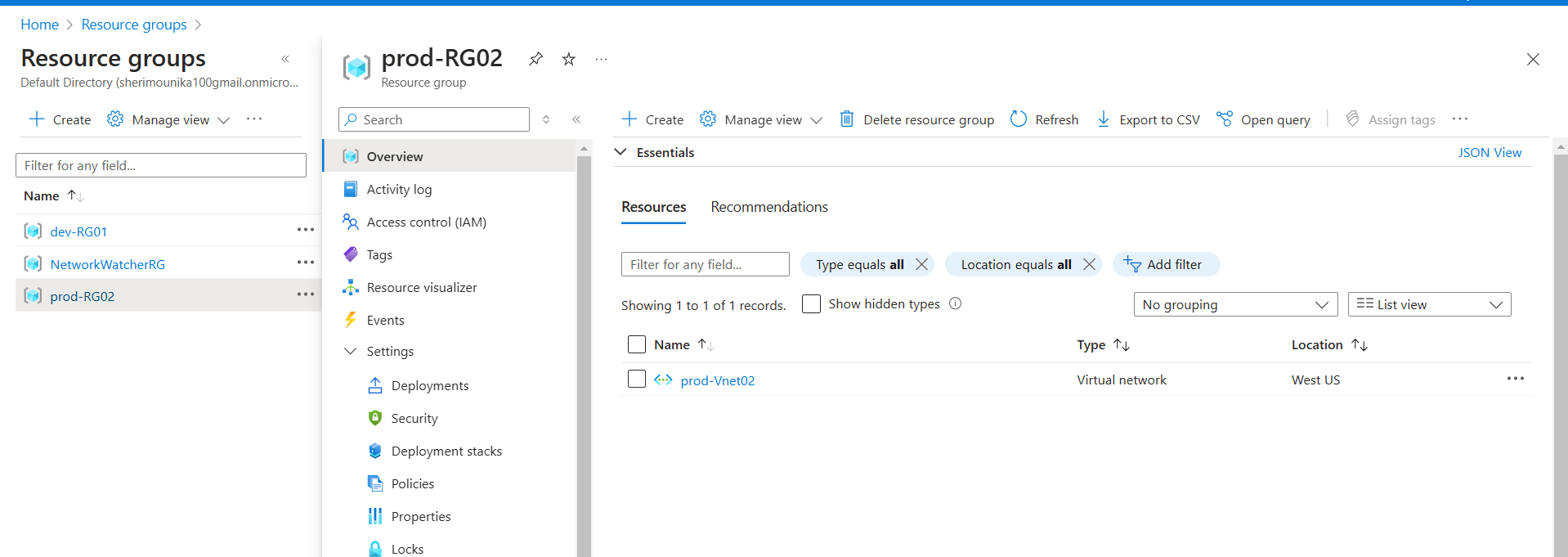
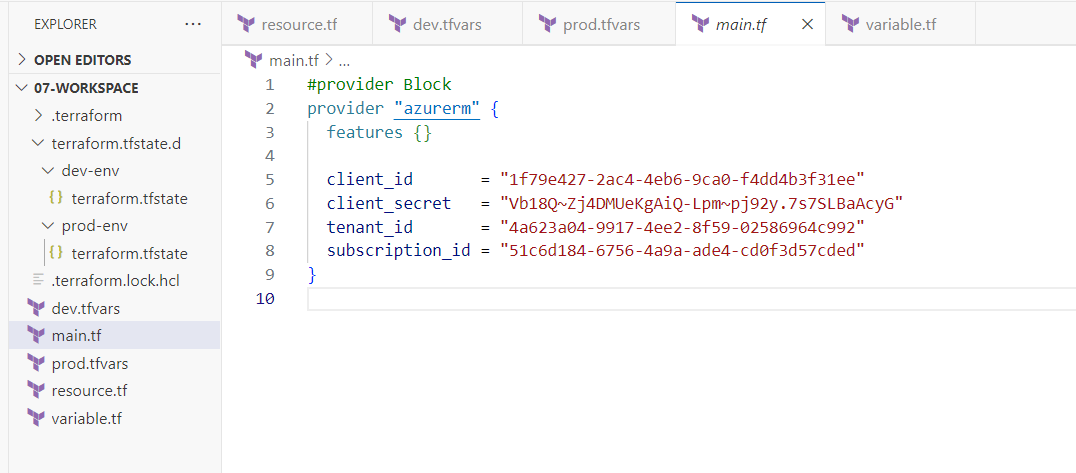


Fig: Resource group and virtual network for **“prod-env”** is created using workspace.

After creating the resources in both the dev-env & prod-env using workspace their state files are present in their corresponding directories as shown in below figure.



The command to destroy or delete the resource from corresponding workspace (dev-env & prod-env) are:

terraform destroy --var-file .\dev.tfvars --auto-approve

terraform destroy --var-file .\prod.tfvars --auto-approve

Deletion can also be done by switching among the workspaces (dev-env & prod-env).