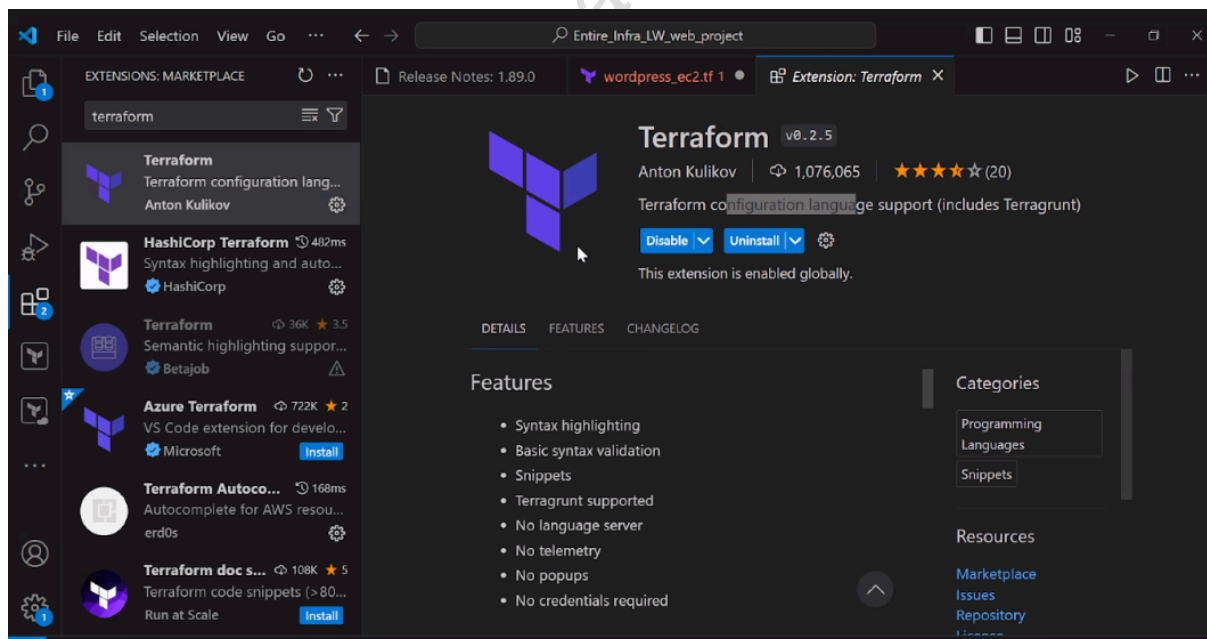




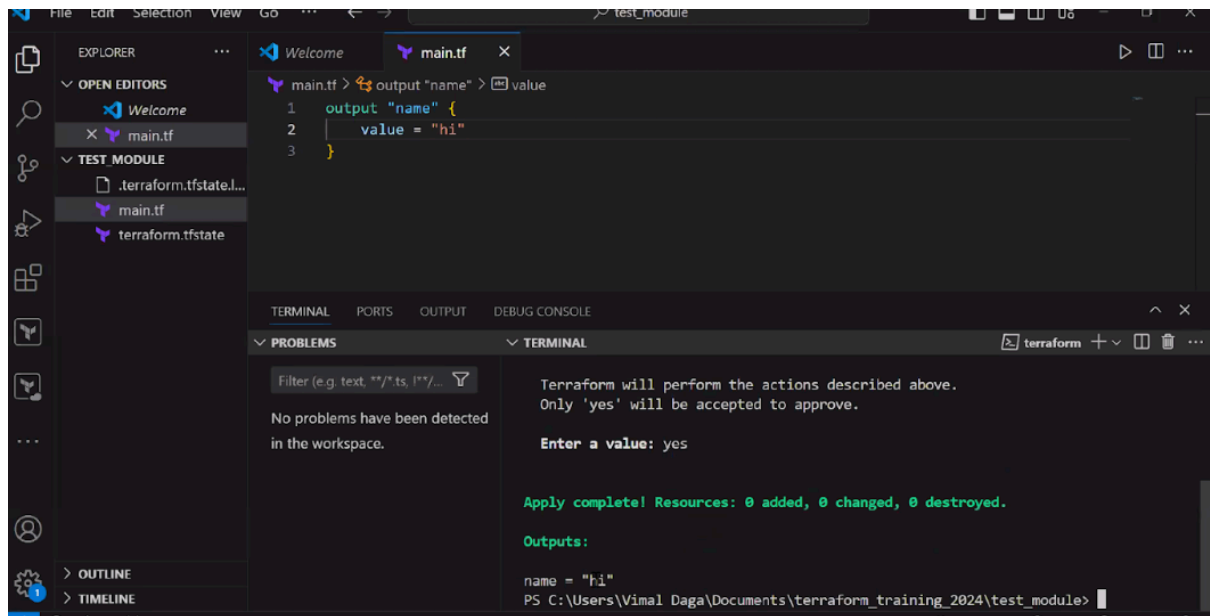
Terraform Training Session 6

Summary 12-05-24

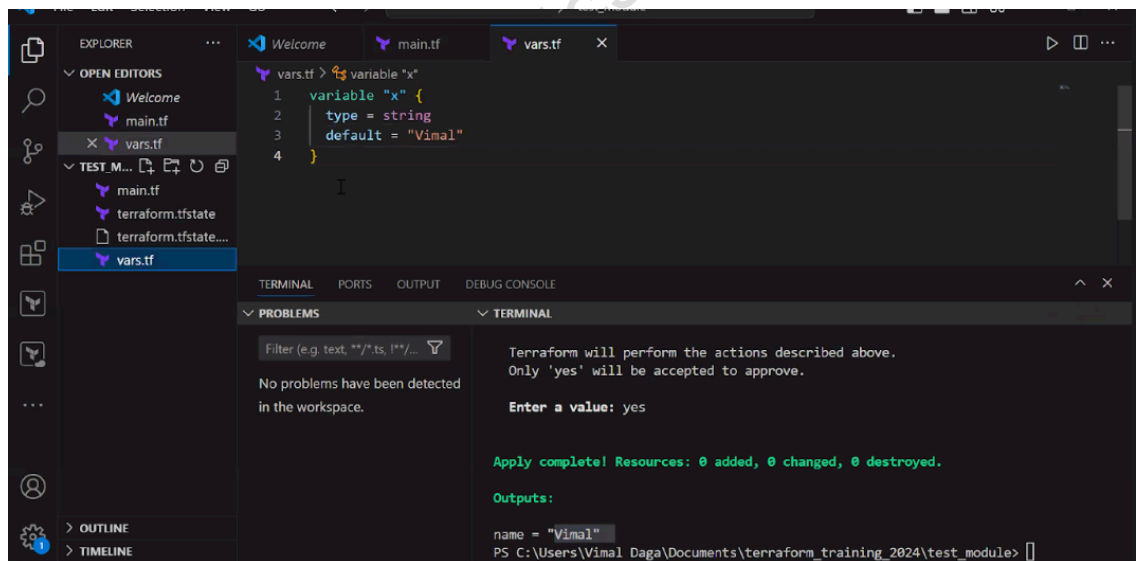
- In Terraform, modules are containers for multiple resources used together, allowing for reusable configurations. Variables in Terraform are placeholders for values that can be assigned to resources, making configurations more dynamic and reusable. Modules can include input variables, resources, and outputs, and variables can be defined with types, defaults, and descriptions. This enables flexible and maintainable infrastructure setups.
- The Visual Studio Code extension marketplace with the "Terraform" extension. This extension provides Terraform configuration language support, including features like syntax highlighting, basic syntax validation.



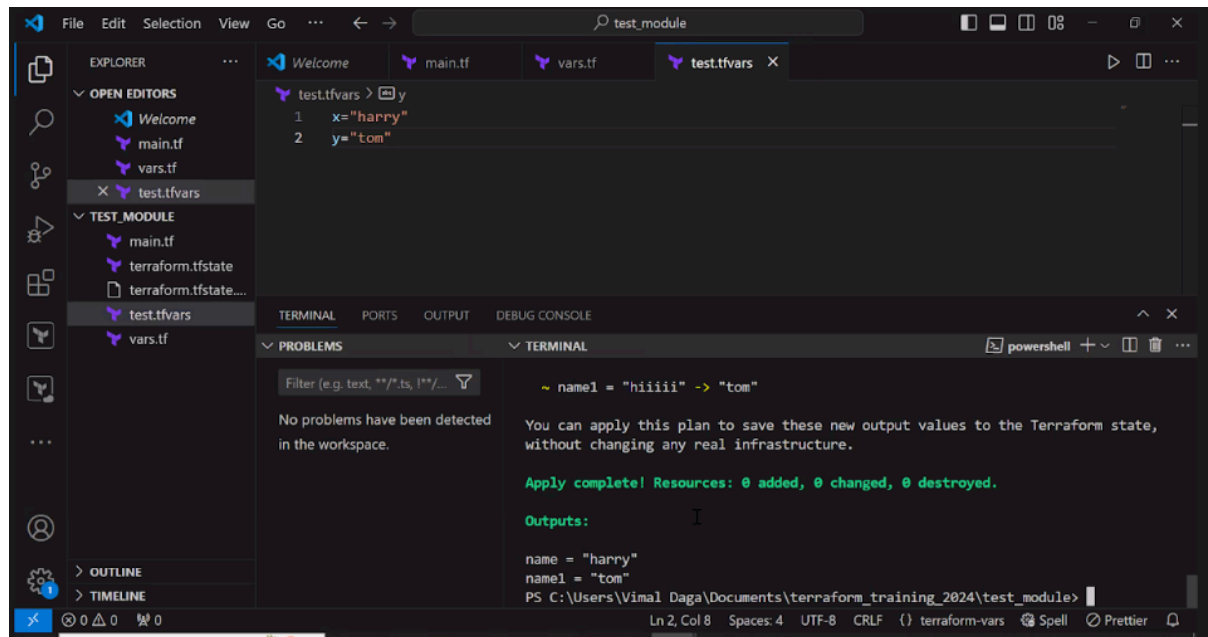
- The **main.tf** file contains a simple output block that assigns the value "hi" to the output variable "name." The terminal indicates that the Terraform apply command was successful, with no resources added, changed, or destroyed, and it outputs the value "name = 'hi'".



- A variable `x` is defined in `vars.tf` with a type of `string` and a default value of `"Vimal"`. The terminal indicates that the Terraform apply command was executed, resulting in no changes to resources, and the output confirms the variable's value as `"Vimal"`.



- The `test.tfvars` file contains variables `x` and `y` with values `"harry"` and `"tom"`. The terminal output indicates that a plan was applied with no changes to resources, and the outputs show the variables' values.



- The Terraform plan was applied with the **prod.tfvars** file, changing the output values from **dev1** to **prod** and **t2.micro** to **t8.large**. No resources were added, changed, or destroyed, and the new output values were saved to the Terraform state

```
pplly -auto-approve --var-file=prod.tfvars

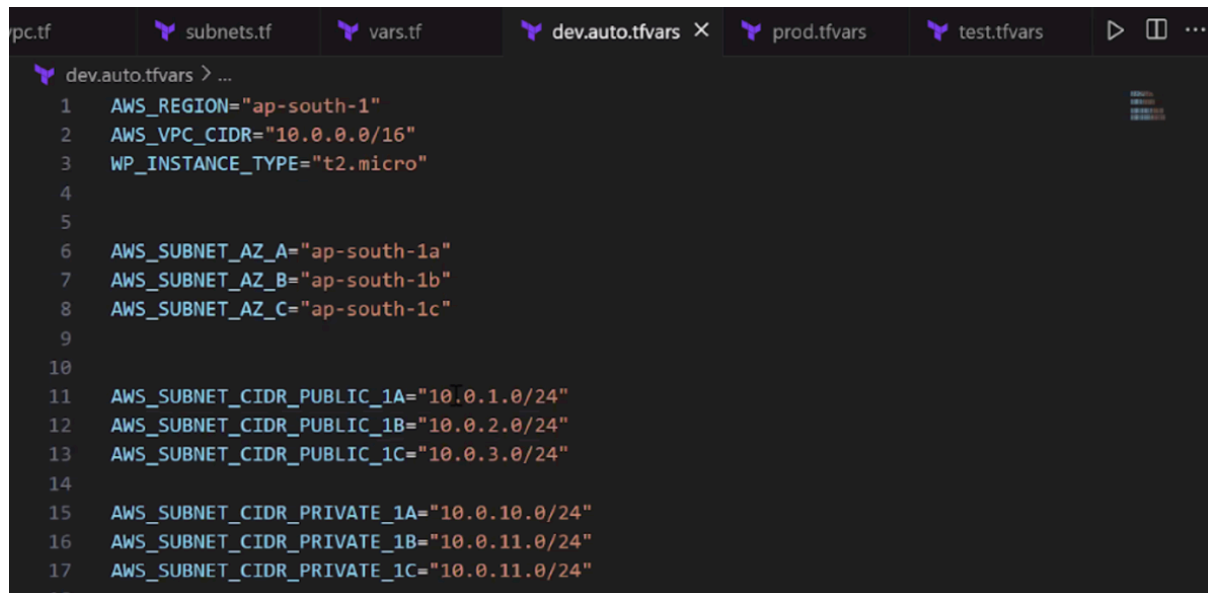
Changes to Outputs:
  ~ name  = "dev1" -> "prod"
  ~ name1 = "t2.micro" -> "t8.large"

You can apply this plan to save these new output values to the Terraform state,
without changing any real infrastructure.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:
name = "prod"
name1 = "t8.large"
```

- A Terraform variable file (**dev.auto.tfvars**) for configuring an AWS environment. It specifies the AWS region as **ap-south-1**, the VPC CIDR block as **10.0.0.0/16**, and the instance type as **t2.micro**. It also defines availability zones and CIDR blocks for public and private subnets within the VPC.



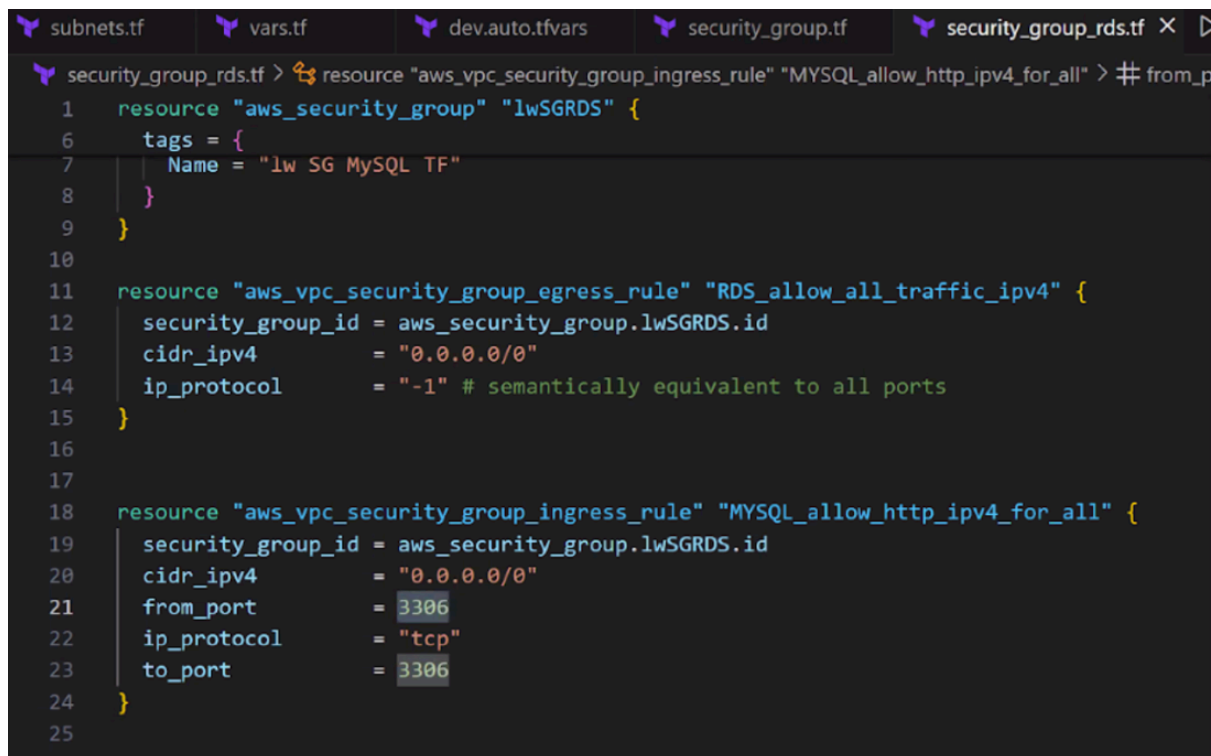
```
pc.tf  subnets.tf  vars.tf  dev.auto.tfvars x  prod.tfvars  test.tfvars  ▶  ▢  ...
dev.auto.tfvars > ...
1  AWS_REGION="ap-south-1"
2  AWS_VPC_CIDR="10.0.0.0/16"
3  WP_INSTANCE_TYPE="t2.micro"
4
5
6  AWS_SUBNET_AZ_A="ap-south-1a"
7  AWS_SUBNET_AZ_B="ap-south-1b"
8  AWS_SUBNET_AZ_C="ap-south-1c"
9
10
11  AWS_SUBNET_CIDR_PUBLIC_1A="10.0.1.0/24"
12  AWS_SUBNET_CIDR_PUBLIC_1B="10.0.2.0/24"
13  AWS_SUBNET_CIDR_PUBLIC_1C="10.0.3.0/24"
14
15  AWS_SUBNET_CIDR_PRIVATE_1A="10.0.10.0/24"
16  AWS_SUBNET_CIDR_PRIVATE_1B="10.0.11.0/24"
17  AWS_SUBNET_CIDR_PRIVATE_1C="10.0.11.0/24"
18
```

- This code snippet defines AWS route table associations for three public subnets in Terraform. Each resource block associates a specific subnet (1a, 1b, and 1c) with a common internet routing table. The code uses the "aws_route_table_association" resource type to create these associations, linking each subnet's ID to the ID of the main internet routing table.



```
.tf  subnets.tf  vars.tf  dev.auto.tfvars  subnet_routing_table_association.tf x  ▶  ▢  ...
subnet_routing_table_association.tf
1  resource "aws_route_table_association" "RTAssociationSubnet1a" {
2      subnet_id = aws_subnet.lw_public_subnet_1a.id
3      route_table_id = aws_route_table.LWRoutingTableInternet.id
4  }
5
6  resource "aws_route_table_association" "RTAssociationSubnet1b" {
7      subnet_id = aws_subnet.lw_public_subnet_1b.id
8      route_table_id = aws_route_table.LWRoutingTableInternet.id
9  }
10
11  resource "aws_route_table_association" "RTAssociationSubnet1c" {
12      subnet_id = aws_subnet.lw_public_subnet_1c.id
13      route_table_id = aws_route_table.LWRoutingTableInternet.id
14  }
```

- The Terraform configuration defines an AWS security group named "lwSGRDS" with tags for identification. It includes an egress rule allowing all outbound traffic and an ingress rule permitting TCP traffic on port 3306 from any IPv4 address (0.0.0.0/0), which is typically used for MySQL. This setup poses a security risk as it allows unrestricted access to the database.



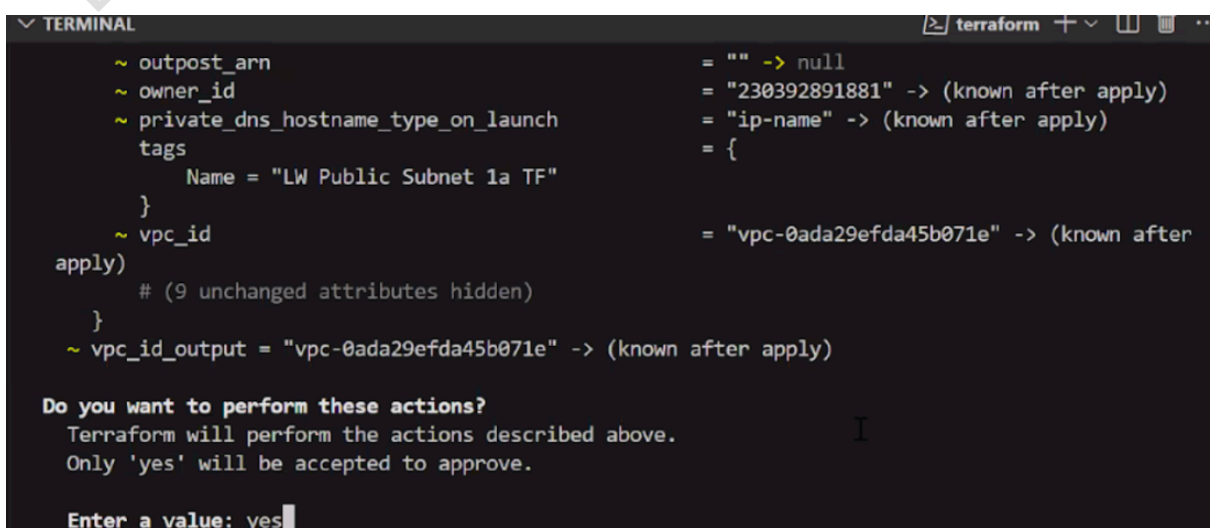
```
subnets.tf  vars.tf  dev.auto.tfvars  security_group.tf  security_group_rds.tf X
security_group_rds.tf > resource "aws_vpc_security_group_ingress_rule" "MYSQL_allow_http_ipv4_for_all" > # from_p
1  resource "aws_security_group" "lwSGRDS" {
6      tags = {
7          Name = "lw SG MySQL TF"
8      }
9  }
10
11 resource "aws_vpc_security_group_egress_rule" "RDS_allow_all_traffic_ipv4" {
12     security_group_id = aws_security_group.lwSGRDS.id
13     cidr_ipv4         = "0.0.0.0/0"
14     ip_protocol       = "-1" # semantically equivalent to all ports
15 }
16
17
18 resource "aws_vpc_security_group_ingress_rule" "MYSQL_allow_http_ipv4_for_all" {
19     security_group_id = aws_security_group.lwSGRDS.id
20     cidr_ipv4         = "0.0.0.0/0"
21     from_port         = 3306
22     ip_protocol       = "tcp"
23     to_port           = 3306
24 }
25
26
```

- The Terraform configuration in the image defines an AWS Internet Gateway resource named "lwIGW." It is associated with a VPC identified by `aws_vpc.lwvpc.id`. Additionally, the resource is tagged with the name "LW Internet Gateway TF."



```
ity_group.tf  security_group_rds.tf  routing_table.tf  rds.tf  internet_gateway.tf X
internet_gateway.tf > resource "aws_internet_gateway" "lwIGW"
1  resource "aws_internet_gateway" "lwIGW" {
2      vpc_id = aws_vpc.lwvpc.id
3
4      tags = {
5          Name = "LW Internet Gateway TF"
6      }
7  }
```

- A Terraform plan output in a terminal. It lists changes to be made to a configuration, including updates to attributes like `outpost_arn`, `owner_id`,



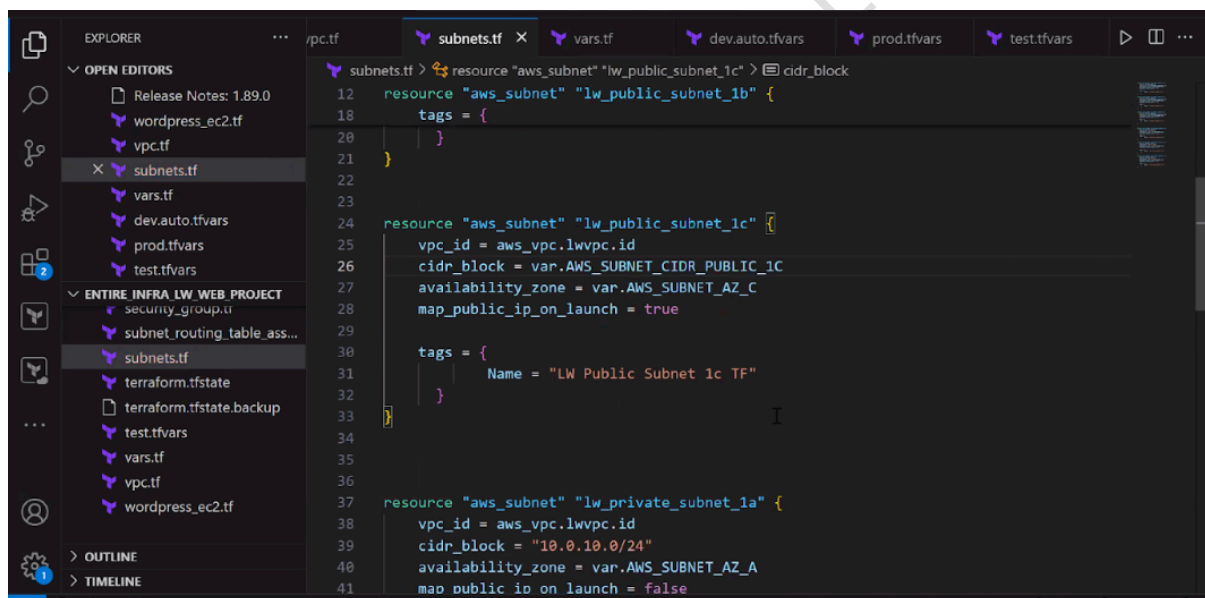
```
▼ TERMINAL  terraform + - □ ▢
~ outpost_arn = "" -> null
~ owner_id = "230392891881" -> (known after apply)
~ private_dns_hostname_type_on_launch = "ip-name" -> (known after apply)
tags = {
  Name = "LW Public Subnet 1a TF"
}
~ vpc_id = "vpc-0ada29efda45b071e" -> (known after apply)
# (9 unchanged attributes hidden)
}
~ vpc_id_output = "vpc-0ada29efda45b071e" -> (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
```

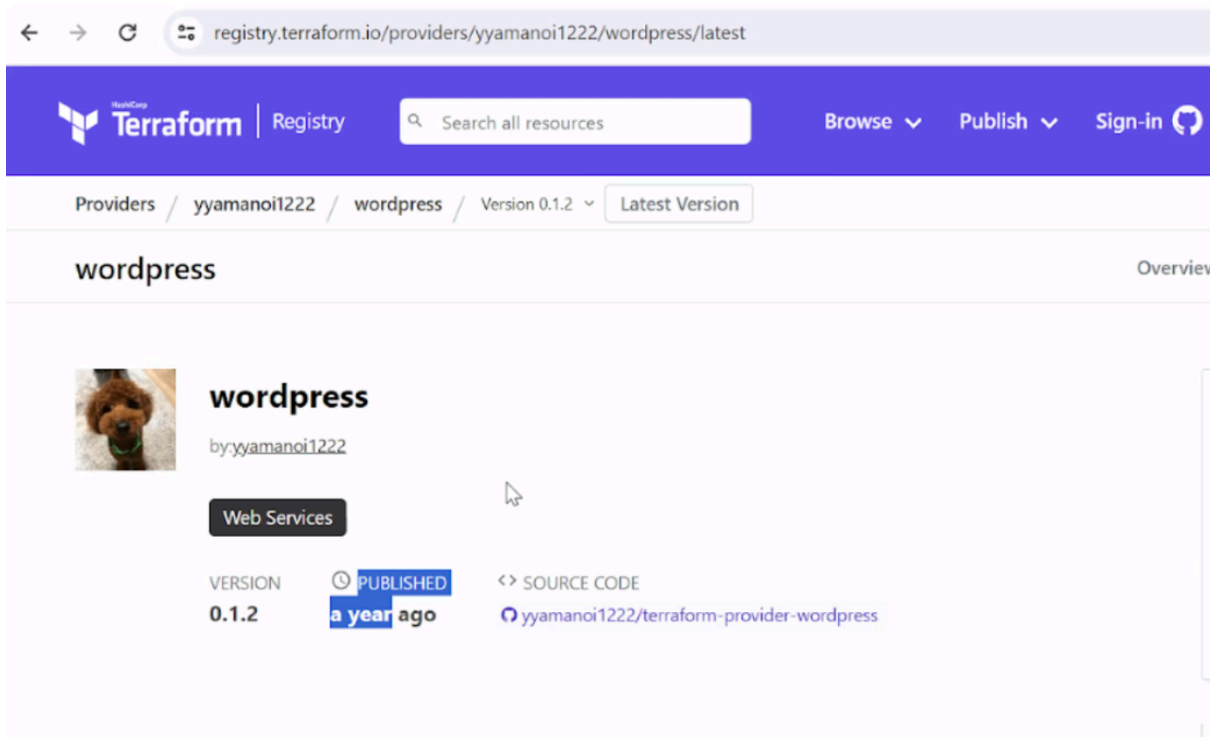
and **tags**. The user is prompted to confirm these actions by typing "yes" to proceed with applying the changes.

- The Terraform configuration in the image defines three AWS subnets:
 - **Public Subnet lw_public_subnet_1b**: An empty resource block with no specific configurations.
 - **Public Subnet lw_public_subnet_1c**: Configured with a VPC ID, a CIDR block, an availability zone, and maps public IPs on launch. It includes a tag for identification.
 - **Private Subnet lw_private_subnet_1a**: Configured with a VPC ID, a CIDR block for a private subnet, an availability zone, and does not map public IPs on launch.

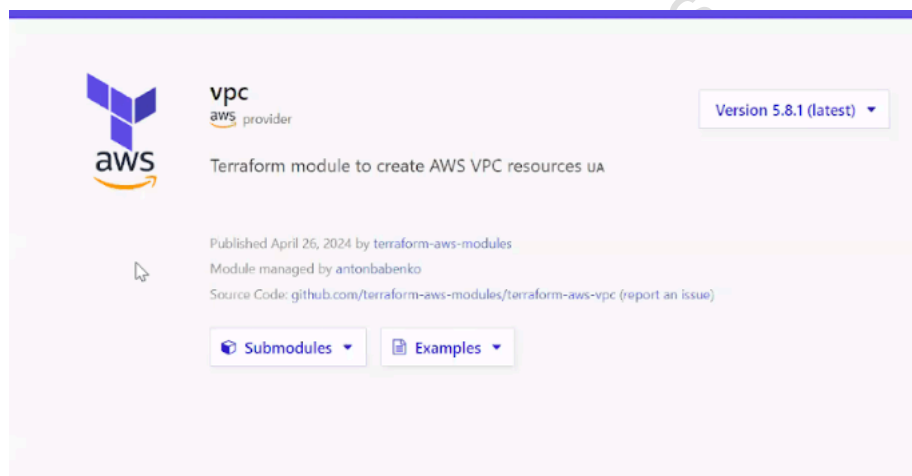


```
12 resource "aws_subnet" "lw_public_subnet_1c" {
13   cidr_block = var.AWS_SUBNET_CIDR_PUBLIC_1C
14   vpc_id     = aws_vpc.lwvpc.id
15   availability_zone = var.AWS_SUBNET_AZ_C
16   map_public_ip_on_launch = true
17   tags = {
18     Name = "LW Public Subnet 1c TF"
19   }
20 }
21
22 resource "aws_subnet" "lw_public_subnet_1b" {
23   cidr_block = var.AWS_SUBNET_CIDR_PUBLIC_1B
24   vpc_id     = aws_vpc.lwvpc.id
25   availability_zone = var.AWS_SUBNET_AZ_C
26   map_public_ip_on_launch = true
27   tags = {
28     Name = "LW Public Subnet 1b TF"
29   }
30 }
31
32 resource "aws_subnet" "lw_private_subnet_1a" {
33   cidr_block = "10.0.10.0/24"
34   vpc_id     = aws_vpc.lwvpc.id
35   availability_zone = var.AWS_SUBNET_AZ_A
36   map_public_ip_on_launch = false
37   tags = {
38     Name = "LW Private Subnet 1a TF"
39   }
40 }
```

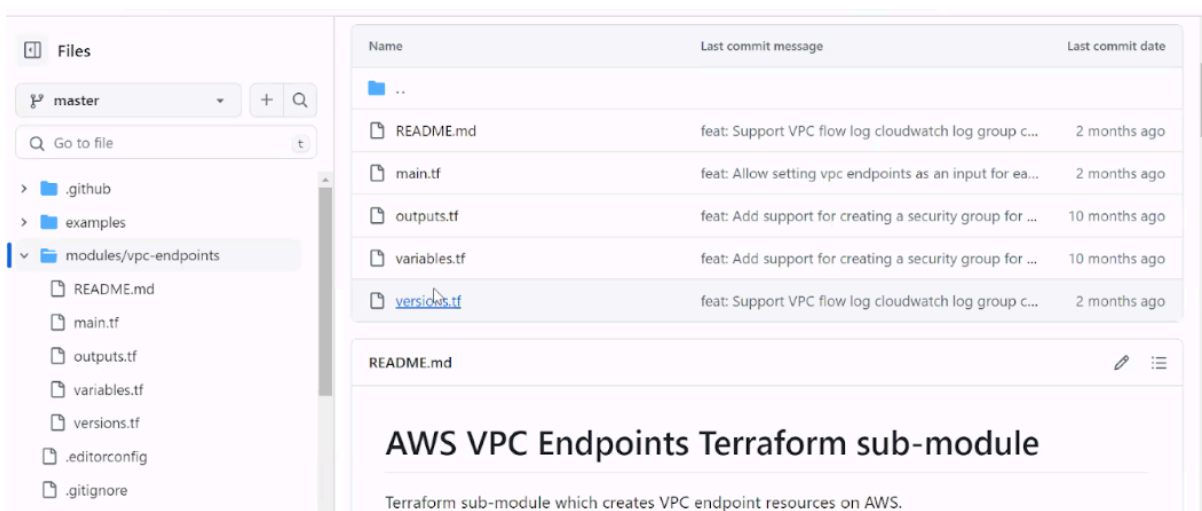
- The Terraform Registry showing a provider named "wordpress" by the user "yyamanoi1222." The provider is categorized under "Web Services" and the latest version is 0.1.2, which was published a year ago. The source code is available on GitHub



- A Terraform module for creating AWS VPC resources.

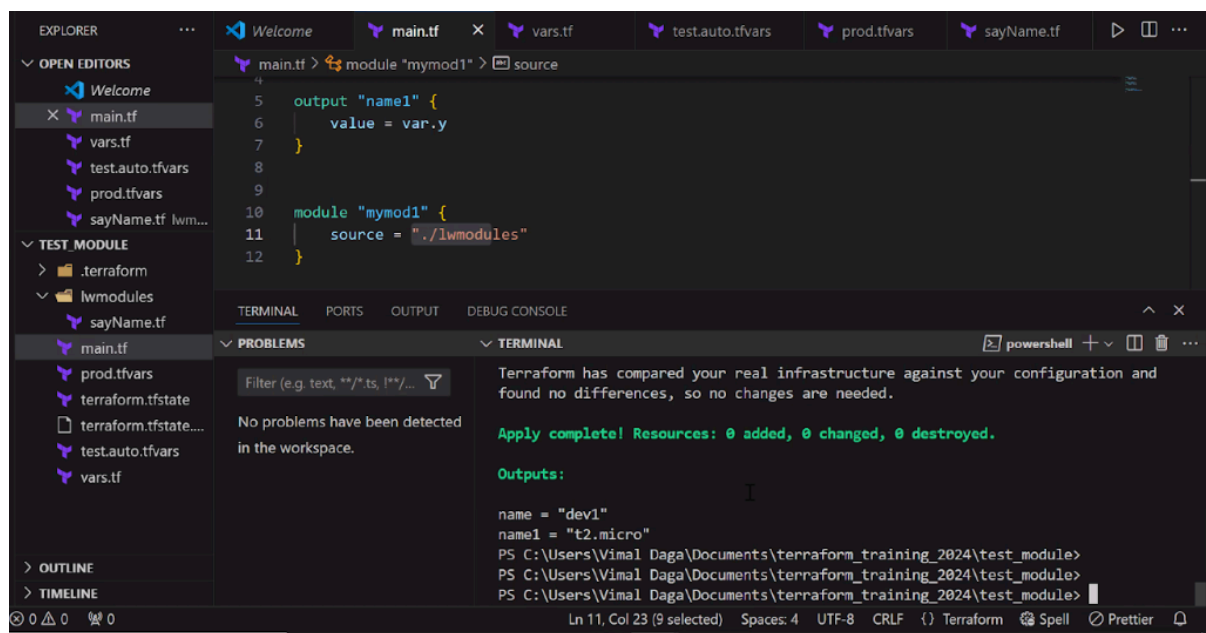


- A GitHub repository for a Terraform sub-module named "AWS VPC Endpoints." It includes files such as **main.tf**, **outputs.tf**, **variables.tf**,



and **versions.tf**, with recent commits focused on supporting VPC flow log CloudWatch log group configurations and setting VPC endpoints as input.

- The **main.tf** file defines an output variable **name1** and a module **mymod1** sourced from **./lwmodules**. The terminal indicates that Terraform applied the configuration with no changes, displaying outputs **name = "dev1"** and **name1 = "t2.micro"**.



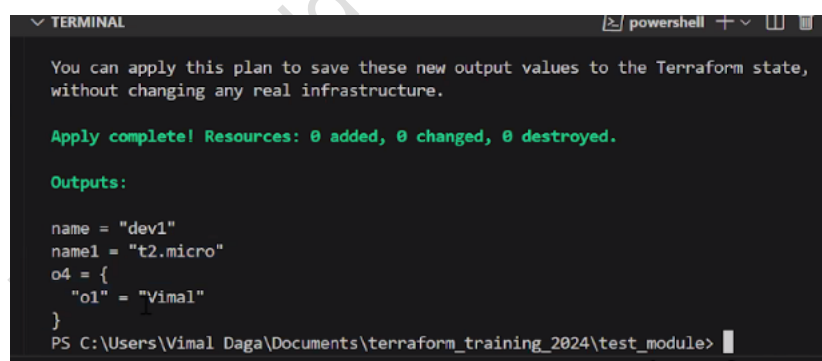
The screenshot shows the Visual Studio Code interface with the Terraform Explorer on the left. The main editor displays the **main.tf** file, which contains the following code:

```
4
5  output "name1" {
6    value = var.y
7  }
8
9
10 module "mymod1" {
11   source = "../lwmodules"
12 }
```

The bottom panel shows the **TERMINAL** output, which indicates that Terraform has compared the real infrastructure against the configuration and found no differences, so no changes are needed. The output also shows the following variables:

```
name = "dev1"
name1 = "t2.micro"
PS C:\Users\Vimal Daga\Documents\terraform_training_2024\test_module>
PS C:\Users\Vimal Daga\Documents\terraform_training_2024\test_module>
PS C:\Users\Vimal Daga\Documents\terraform_training_2024\test_module>
```

- A Terraform output indicating that no resources were added, changed, or destroyed. The outputs include variables: **name** set to "dev1", **name1** set to "t2.micro", and **o4** as a map with "O1" set to "Vimal".



The screenshot shows the **TERMINAL** output, which indicates that Terraform has compared the real infrastructure against the configuration and found no differences, so no changes are needed. The output also shows the following variables:






```
You can apply this plan to save these new output values to the Terraform state,
without changing any real infrastructure.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:


name = "dev1"
name1 = "t2.micro"
o4 = {
  "o1" = "Vimal"
}
PS C:\Users\Vimal Daga\Documents\terraform_training_2024\test_module>
```

- Terraform modules are collections of configuration files used to manage infrastructure resources. They help organize and reuse code by encapsulating groups of related resources, making infrastructure management more efficient and scalable.

 terraform module example    


[All](#) [Images](#) [Videos](#) [Shopping](#) [News](#) [More](#)


[GitHub](#) [Azure](#) [Import](#) [VPC](#) [Output](#)

 **Spacelift**
[https://spacelift.io > blog > what-are-terraform-modules...](https://spacelift.io/blog/what-are-terraform-modules...)

What Are Terraform Modules and How to Use Them


A **Terraform module** is a collection of standard configuration files in a dedicated directory. **Terraform modules** encapsulate groups of resources dedicated to one ...



 **Gruntwork**
[https://blog.gruntwork.io > how-to-create-reusable-infra...](https://blog.gruntwork.io/how-to-create-reusable-infra...)

How to create reusable infrastructure with Terraform ...

As a first step, run **terraform destroy** in the `stage/services/webserver-cluster` to clean up any resources that you created earlier. Next, create a new top-level ...



Linux World Informatics Pvt Ltd