Terraform Practical

By Gaurav Sir

1. Create a file in terraform which takes 2 inputs from the user i.e. name and age and print it in following ways - taking input during execution, taking default value, taking command line arguments.

Firstly taking input during execution

```
# inside the var-list.tf

variable username {}
variable userage {}
# inside var-file.tf

output lineone {
    value = "hello ${var.username}, how was your day"
    }

output linetwo {
    value = "Sorry ${var.username}, you are ${var.userage} years old, not eligible for game"
    }
```

Now, we are considering the default values

```
# inside the var-list.tf

variable username {
    default = "John Nathan"
}
variable userage {
    default = 45
}


# inside var-file.tf

output lineone {
    value = "hello ${var.username}, how was your day"
    }

output linetwo {
    value = "Sorry ${var.username}, you are ${var.userage} years old, not eligible for game"
    }
```

Now providing the command line argument

2. Tell the variable types in Terraform

Here are some common variable types in Terraform with examples for each:

1. String: Represents a sequence of text characters.

- **Example:**

Terraform

```
variable "region" {
  type = string
  description = "The region where resources will be created."
  default = "us-east-1"
}

variable "api_gateway_name" {
  type = string
  description = "The name for your API Gateway"
}
```

2. Number: Represents numeric values, including whole numbers and decimals.

- **Example:**

Terraform

```
variable "instance_count" {
  type = number
  description = "The number of instances to launch."
  default = 2
}

variable "memory_size" {
  type = number
  description = "The amount of memory (GiB) for each instance."
}
```

3. Bool: Represents a boolean value, either `true` or `false`.

- **Example:**

Terraform

```
variable "enable_monitoring" {
  type = bool
  description = "Enable CloudWatch monitoring for the resources."
  default = true
}

variable "create_public_subnet" {
  type = bool
  description = "Create a public subnet for internet access (if needed)."
  default = false
}
```

4. List: Represents an ordered sequence of values. Elements in a list can be of any valid Terraform type, including other lists.

- **Example:**

Terraform

.

```
# putting this code in variable file

variable "security_group_ids" {
  type = list(string)
  description = "List of security group IDs to attach to the instances."
  default = ["sg-0123456789abcdef0"]
}


# putting following code in execution file (implementing the concept of file destructuring - Gaurab sir)
output mydata {
    value = "Your 1st security group id is ${var.security_group_ids[0]} "
    }


# agar hum execution time input de to fir following format mei denge
["sg-0123456789abcdef0","sg-4576789abcdef0","sg-9777756789abcdef0"]

# agar hum command line se input de rahe hai to
terraform plan -var 'security_group_ids=["sg-0123456789abcdef0","sg-4576789abcdef0","sg-9777756789abcdef0"]'
```

- **Note:** In the `tags` example, the list elements are actually key-value pairs represented as a map within the list. Not understand, may be baad me samajh aaye. But list ki indexing 0 se start hoti hai, it is important

5. Object: Represents a collection of key-value pairs, where keys are strings and values can be any valid Terraform type.

- **Example:**

Terraform

```
variable "vpc_config" {
  type = object({
    cidr_block = string
    availability_zones = list(string)
  })
  description = "Configuration for the VPC (Virtual Private Cloud)."
}
```

Here's a breakdown of the variable types you requested in Terraform, including descriptions and examples:

6. Set: Represents an unordered collection of unique values. Elements in a set must be of the same type (string, number, etc.). Terraform does not guarantee the order in which elements appear when referencing the set.

- **Example:**

Terraform

```
variable "allowed_ip_addresses" {
  type = set(string)
  description = "A set of IP addresses allowed to access the resource."
  default = {
    "10.0.0.1",
    "10.0.0.2",
  }
}
```

7. Map: Represents a collection of key-value pairs. Keys must be strings, while values can be any valid Terraform type (including other maps or lists). Maps provide a way to store structured data associated with variables.

- **Example:**

Terraform

```
variable "web_server_config" {
  type = map(object({
    instance_type = string
    ami_id = string
  }))
  description = "Configuration for web server instances in different environments."
  default = {
    "production" = {
      instance_type = "t2.micro"
      ami_id = "ami-0123456789abcdef0"
    },
    "staging" = {
      instance_type = "t2.small"
      ami_id = "ami-fedcba0987654321"
    },
  }
}
```

8. Null: Represents the absence of a value. It's useful for indicating that a variable has no assigned value or can be left empty intentionally.

- **Example:**

Terraform

```
variable "optional_message" {
  type = string
  description = "An optional message to include in the resource creation."
  default = null
}
```

9. Object: Similar to maps, objects represent collections of key-value pairs. However, objects enforce a specific structure by defining the required keys and their expected types. This provides better type safety and helps prevent errors during configuration. Object v ariable repeat iss liye ho gaya because ye mera favourite hai.

- **Example:**

Terraform

```
variable "network_config" {
  type = object({
    name = string
    cidr_block = string
    subnet_cidr_blocks = list(string)
  })
  description = "Configuration for creating a virtual network."
  default = {
    name = "my-vpc"
    cidr_block = "10.0.0.0/16"
    subnet_cidr_blocks = ["10.0.1.0/24", "10.0.2.0/24"]
  }
}
```

3. What is the difference between map and object variable type in terraform

- **Map:** More flexible structure. Keys must be strings, but values can be any valid Terraform type (including other maps or lists). You can add or remove key-value pairs dynamically without affecting the overall structure.

- **Object:** Enforces a predefined structure. You define the required keys and their expected types upfront. This provides better type safety and prevents errors if you try to use an incorrect type for a value.

Hum object create time object ka schema define karte hai, agar koi data schema me fit nhi hota hai to vo data partially lose ho jata hai. Hence Terrafrom mei map to object conversion possible hai but data loss ho jata hai.

Example

```
# Map Example (Flexible)
variable "tags" {
  type = map(string)
  default = {
    Name = "My Resource"
    Environment = "Production"
    # You can add more key-value pairs dynamically
  }
}

# Object Example (Structured)
variable "network_config" {
  type = object({
    name = string
    cidr_block = string
    # Keys and their types are predefined
  })
  default = {
    name = "my-vpc"
    cidr_block = "10.0.0.0/16"
  }
}
```

4. Tell some function in Terraform

Terraform offers a rich set of functions to manipulate data, perform calculations, and enhance the readability and reusability of your configurations. Here are some popular functions categorized by their functionalities:

String Functions:

- **trim(string, chars):** Removes leading and trailing characters from a string. Optionally, you can specify characters to remove.

Terraform

```
variable "username" {
  type = string
  default = "  johndoe  "
}

output "trimmed_username" {
  value = trim(var.username, " ")
}

# Output: johndoe
```

- **replace(string, old, new):** Replaces all occurrences of a substring with another substring within a string.

Terraform

```
variable "hostname" {
  type = string
  default = "web-server-prod"
}

output "dev_hostname" {
  value = replace(var.hostname, "-prod", "-dev")
}

  # Output: web-server-dev
```

Numeric Functions:

- **abs(number):** Returns the absolute value of a number.

Terraform

```
variable "temperature" {
  type = number
  default = -10
}
```

```
output "absolute_temperature" {
  value = abs(var.temperature)
}

  # Output: 10
```

- **ceil(number):** Rounds a number up to the nearest integer.

Terraform

```
variable "disk_size" {
  type = number
  default = 10.5
}

output "rounded_disk_size" {
  value = ceil(var.disk_size)
}

  # Output: 11
```

Collection Functions:

- **length(list or map):** Returns the number of elements in a list or map.

Terraform

```
variable "security_groups" {
  type = list(string)
  default = ["sg-0123456", "sg-abcdefg"]
}

output "number_of_security_groups" {
  value = length(var.security_groups)
}

  # Output: 2
```

- **element(list, index):** Accesses a specific element within a list based on its index (starting from 1).

Terraform

```
variable "web_servers" {
  type = list(object({
    name = string
    instance_type = string
  }))
  default = [
    { name = "web-server-1", instance_type = "t2.micro" },
    { name = "web-server-2", instance_type = "t2.small" },
  ]
}

output "first_webserver_name" {
  value = element(var.web_servers, 1).name
}
```

Date and Time Functions:

- **timestamp():** Returns the current timestamp as a RFC 3339 formatted string.

Terraform

```
output "current_timestamp" {
  value = timestamp()
}

  # Output: The current timestamp in RFC 3339 format (e.g., "2024-07-06T21:16:00Z")
```

- **formatdate(format, timestamp):** Formats a timestamp according to a specified format string.

Terraform

```
variable "creation_time" {
  type = string
  default = timestamp()
}

output "formatted_creation_time" {
  value = formatdate("YYYY-MM-DD", var.creation_time)
```

```
}

  # Output: The formatted creation time in YYYY-MM-DD format (e.g., "2024-07-06")
```

Encoding Functions:

- **base64encode(string):** Encodes a string to base64 format.

Terraform

```
variable "api_key" {
  type = string
  sensitive = true
}

output "encoded_api_key" {
  value = base64encode(var.api_key)
}

  # Output: The base64 encoded API key (obfuscated due to sensitive variable)
```

Type Conversion Functions:

- **tonumber(value):** Attempts to convert a value to a number.

Terraform

```
variable "disk_size_string" {
  type = string
  default = "10 GB"
}

output "disk_size_in_gb" {
  value = tonumber(split(var.disk_size_string, " ")[0])
}

  # Output: 10 (Extracts the numeric value from the string)
```

Resource list

https://learning-ocean.com/tutorials/terraform/terraform-variable/

https://www.zero2devops.com/blog/ultimate-guide-to-terraform

https://zeet.co/blog/terraform-tutorial

https://developer.hashicorp.com/terraform/tutorials

https://k21academy.com/ansible/terraform-vs-ansible/

```
# Use this following line while while executing your terraform script

terraform plan -var "username=Anish" -var "userage=75"
```

2. Tell the variable types in Terraform

Here are some common variable types in Terraform with examples for each:

1. String: Represents a sequence of text characters.

- **Example:**

Terraform

```
variable "region" {
  type = string
  description = "The region where resources will be created."
  default = "us-east-1"
}

variable "api_gateway_name" {
  type = string
  description = "The name for your API Gateway"
}
```

2. Number: Represents numeric values, including whole numbers and decimals.

- **Example:**

Terraform

```
variable "instance_count" {
  type = number
  description = "The number of instances to launch."
  default = 2
}

variable "memory_size" {
  type = number
  description = "The amount of memory (GiB) for each instance."
}
```

3. Bool: Represents a boolean value, either `true` or `false`.

- **Example:**

Terraform

```
variable "enable_monitoring" {
  type = bool
  description = "Enable CloudWatch monitoring for the resources."
  default = true
}

variable "create_public_subnet" {
  type = bool
  description = "Create a public subnet for internet access (if needed)."
  default = false
}
```

4. List: Represents an ordered sequence of values. Elements in a list can be of any valid Terraform type, including other lists.

- **Example:**

Terraform

.

```
# putting this code in variable file

variable "security_group_ids" {
  type = list(string)
  description = "List of security group IDs to attach to the instances."
  default = ["sg-0123456789abcdef0"]
}


# putting following code in execution file (implementing the concept of file destructuring - Gaurab sir)
output mydata {
    value = "Your 1st security group id is ${var.security_group_ids[0]} "
    }


# agar hum execution time input de to fir following format mei denge
["sg-0123456789abcdef0","sg-4576789abcdef0","sg-9777756789abcdef0"]

# agar hum command line se input de rahe hai to
terraform plan -var 'security_group_ids=["sg-0123456789abcdef0","sg-4576789abcdef0","sg-9777756789abcdef0"]'
```

- **Note:** In the `tags` example, the list elements are actually key-value pairs represented as a map within the list. Not understand, may be baad me samajh aaye. But list ki indexing 0 se start hoti hai, it is important

5. Object: Represents a collection of key-value pairs, where keys are strings and values can be any valid Terraform type.

- **Example:**

Terraform

```
variable "vpc_config" {
  type = object({
    cidr_block = string
    availability_zones = list(string)
  })
  description = "Configuration for the VPC (Virtual Private Cloud)."
}
```

Here's a breakdown of the variable types you requested in Terraform, including descriptions and examples:

6. Set: Represents an unordered collection of unique values. Elements in a set must be of the same type (string, number, etc.). Terraform does not guarantee the order in which elements appear when referencing the set.

- **Example:**

Terraform

```
variable "allowed_ip_addresses" {
  type = set(string)
  description = "A set of IP addresses allowed to access the resource."
  default = {
    "10.0.0.1",
    "10.0.0.2",
  }
}
```

7. Map: Represents a collection of key-value pairs. Keys must be strings, while values can be any valid Terraform type (including other maps or lists). Maps provide a way to store structured data associated with variables.

- **Example:**

Terraform

```
variable "web_server_config" {
  type = map(object({
    instance_type = string
    ami_id = string
  }))
  description = "Configuration for web server instances in different environments."
  default = {
    "production" = {
      instance_type = "t2.micro"
      ami_id = "ami-0123456789abcdef0"
    },
    "staging" = {
      instance_type = "t2.small"
      ami_id = "ami-fedcba0987654321"
    },
  }
}
```

8. Null: Represents the absence of a value. It's useful for indicating that a variable has no assigned value or can be left empty intentionally.

- **Example:**

Terraform

```
variable "optional_message" {
  type = string
  description = "An optional message to include in the resource creation."
  default = null
}
```

9. Object: Similar to maps, objects represent collections of key-value pairs. However, objects enforce a specific structure by defining the required keys and their expected types. This provides better type safety and helps prevent errors during configuration. Object v ariable repeat iss liye ho gaya because ye mera favourite hai.

- **Example:**

Terraform

```
variable "network_config" {
  type = object({
    name = string
    cidr_block = string
    subnet_cidr_blocks = list(string)
  })
  description = "Configuration for creating a virtual network."
  default = {
    name = "my-vpc"
    cidr_block = "10.0.0.0/16"
    subnet_cidr_blocks = ["10.0.1.0/24", "10.0.2.0/24"]
  }
}
```

3. What is the difference between map and object variable type in terraform

- **Map:** More flexible structure. Keys must be strings, but values can be any valid Terraform type (including other maps or lists). You can add or remove key-value pairs dynamically without affecting the overall structure.
- **Object:** Enforces a predefined structure. You define the required keys and their expected types upfront. This provides better type safety and prevents errors if you try to use an incorrect type for a value.

Hum object create time object ka schema define karte hai, agar koi data schema me fit nhi hota hai to vo data partially lose ho jata hai. Hence Terrafrom mei map to object conversion possible hai but data loss ho jata hai.

Example

```
# Map Example (Flexible)
variable "tags" {
  type = map(string)
  default = {
    Name = "My Resource"
    Environment = "Production"
    # You can add more key-value pairs dynamically
  }
}

# Object Example (Structured)
variable "network_config" {
  type = object({
    name = string
    cidr_block = string
    # Keys and their types are predefined
  })
  default = {
    name = "my-vpc"
    cidr_block = "10.0.0.0/16"
  }
}
```

4. Tell some function in Terraform

Terraform offers a rich set of functions to manipulate data, perform calculations, and enhance the readability and reusability of your configurations. Here are some popular functions categorized by their functionalities:

String Functions:

- **trim(string, chars):** Removes leading and trailing characters from a string. Optionally, you can specify characters to remove.

Terraform

```
variable "username" {
  type = string
  default = "  johndoe  "
}

output "trimmed_username" {
  value = trim(var.username, " ")
}

# Output: johndoe
```

- **replace(string, old, new):** Replaces all occurrences of a substring with another substring within a string.

Terraform

```
variable "hostname" {
  type = string
  default = "web-server-prod"
}

output "dev_hostname" {
  value = replace(var.hostname, "-prod", "-dev")
}

  # Output: web-server-dev
```

Numeric Functions:

- **abs(number):** Returns the absolute value of a number.

Terraform

```
variable "temperature" {
  type = number
  default = -10
}

output "absolute_temperature" {
  value = abs(var.temperature)
}

  # Output: 10
```

- **ceil(number):** Rounds a number up to the nearest integer.

Terraform

```
variable "disk_size" {
  type = number
  default = 10.5
}

output "rounded_disk_size" {
  value = ceil(var.disk_size)
}

  # Output: 11
```

Collection Functions:

- **length(list or map):** Returns the number of elements in a list or map.

Terraform

```
variable "security_groups" {
  type = list(string)
  default = ["sg-0123456", "sg-abcdefg"]
}

output "number_of_security_groups" {
  value = length(var.security_groups)
}

  # Output: 2
```

- **element(list, index):** Accesses a specific element within a list based on its index (starting from 1).

Terraform

```
variable "web_servers" {
  type = list(object({
    name = string
    instance_type = string
  }))
  default = [
    { name = "web-server-1", instance_type = "t2.micro" },
    { name = "web-server-2", instance_type = "t2.small" },
  ]
}

output "first_webserver_name" {
  value = element(var.web_servers, 1).name
}
```

Date and Time Functions:

- **timestamp():** Returns the current timestamp as a RFC 3339 formatted string.

Terraform

```
output "current_timestamp" {
  value = timestamp()
}

  # Output: The current timestamp in RFC 3339 format (e.g., "2024-07-06T21:16:00Z")
```

- **formatdate(format, timestamp):** Formats a timestamp according to a specified format string.

Terraform

```
variable "creation_time" {
  type = string
  default = timestamp()
}

output "formatted_creation_time" {
  value = formatdate("YYYY-MM-DD", var.creation_time)
}

  # Output: The formatted creation time in YYYY-MM-DD format (e.g., "2024-07-06")
```

Encoding Functions:

- **base64encode(string):** Encodes a string to base64 format.

Terraform

```
variable "api_key" {
  type = string
  sensitive = true
}

output "encoded_api_key" {
  value = base64encode(var.api_key)
}

  # Output: The base64 encoded API key (obfuscated due to sensitive variable)
```

Type Conversion Functions:

- **tonumber(value):** Attempts to convert a value to a number.

Terraform

```
variable "disk_size_string" {
  type = string
  default = "10 GB"
}

output "disk_size_in_gb" {
  value = tonumber(split(var.disk_size_string, " ")[0])
}

  # Output: 10 (Extracts the numeric value from the string)
```

5. How to use custom tfvars file in terraform

we use the option `-var-file=<your-filenamwe>.tfvars`

6. How to pass system variable in Terraform

Suppose the following code

```
# defining the varialbe
variable username {
    type = string
    }

# using the variable
output firstline {
    value = "hello ${var.username}, how are your ? "
    }

# We need to create the system variable such that it have prefix "TF_VAR_"
export TF_VAR_username="Heena"


# executing command - terraform plan we get following result
hello Heena, how are you ?
```

7. How to create EC2 instance in aws using terraform

Step 1:

Create a provider for aws, file - provider.tf

```
provider "aws" {
  region     = "us-east-1"
  secret_key = "GQiNUlYgmvXeGlwcZXVIhZJkQFqSsCS2VmsQ8TVh"
  access_key = "AKIA6ODU7FGKAGA4MLXO"
}
```

Step 2:

Create an EC2 instance, file - instance.tf

```
# Creating a resource - ec2 instnce
resource "aws_instance" "app_server" {
  ami                    = "ami-06c68f701d8090592" # Select correct  AMI by considering the region properly
  instance_type          = "t2.micro"
  key_name               = aws_key_pair.my_key_pair.key_name  # Taking key from key pair configuration
  vpc_security_group_ids = ["${aws_security_group.my-sg-tf.id}"] # Taking id from security group configuration
  tags = {
    Name = "Instance from terraform"
  }
}
```

Step 3:

Create a key-pair, file - keys.tf in outside directory. where .terraform is there at beside

```
# creating a resource type - key pair
# Create a key pair resource with the public key content
resource "aws_key_pair" "my_key_pair" {
  key_name   = "my-key-pair" # this key pair name can be different as name at local
  public_key = file("${path.module}/key-pair/id_rsa.pub")

}
```

Step 4:

Creating a key. Use command ssh-keygen. Give the absolute path to your id_rsa file. see your private and public keys inside the directory - key-pair/

Step 5:

Creating a security group adding ports.

```
# Creating a security group
resource "aws_security_group" "my-sg-tf" {
  name        = "my-sg-tf"
  description = "Security group for SSH, web, and HTTPS access"

  # we can also create a dynamic block, inside the security group block
  dynamic "ingress" {
    for_each = [22, 80, 443, 3306, 27017]
    iterator = port
    content {
      description = "TLS from VPC"
      from_port   = port.value
      to_port     = port.value
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}

#  ingress {
#     from_port = 22
#     to_port   = 22
#     protocol = "tcp"
#     cidr_blocks = ["0.0.0.0/0"]  # Update with specific IP range if needed
#   }

#   ingress {
#     from_port = 80
#     to_port   = 80
#     protocol = "tcp"
#     cidr_blocks = ["0.0.0.0/0"]  # Update with specific IP range if needed
#   }

#   ingress {
#     from_port = 443
#     to_port   = 443
#     protocol = "tcp"
#     cidr_blocks = ["0.0.0.0/0"]  # Update with specific IP range if needed
#   }

#   egress {
#     from_port = 0
#     to_port   = 0
```

```
#     protocol = "-1"
#     cidr_blocks = ["0.0.0.0/0"]
#   }
```

Step 6:

Execute the commands

```
# Validating thi code
terraform validate

# Formatting the code
terraform fmt

# Checking the plan
terraform plan

# Execution the configuration file
terraform apply --auto-approve

# Destroy the infrastructure
terraform destroy --auto-approve
```

8. Perform a practical to implement the concept of modules

Step 1 : Creating directory structure

```
.
├── main.tf
├── modules
│   └── webserver
│       ├── ec2.tf
│       ├── key.tf
│       ├── output.tf
│       └── variable.tf
├── outputs.tf
├── private-key
├── providers.tf
├── public-key.pub
├── security-group.tf
├── terraform.tfstate
├── terraform.tfstate.backup
├── terraform.tfvars
└── variables.tf
```

Step 2: Write the files for parent directory

file - providers.tf

```
provider "aws" {
  region     = "us-east-1"
  secret_key = "GQiNUlYgmvXeGlwcZXVIhZJkQFqSsCS2VmsQ8TVh"
  access_key = "AKIA6ODU7FGKAGA4MLXO"
}
```

file - variables.tf

```
variable "parent-ami-id" {
  type = string
}

variable "parent-instance-type" {
  type = string
}

variable "parent-instance1-key" {
  type = string
}

variable "parent-instance2-key" {
  type = string
}
```

```
variable "parent-instance1-name" {
  type = string
}

variable "parent-instance2-name" {
  type = string
}

variable "parent-instance1-env" {
  type = string
}

variable "parent-instance2-env" {
  type = string
}

variable "key_name" {
  type = string
}

variable "ami" {
  type = string
}

variable "instance_type" {
  type = string
}

variable "name" {
  type = string
}

variable "environment" {
  type = string
}


variable "sg-id" {
  type = string
}
```

file - main.tf

```
module "web-server1" {
  # providing directory of child module
  source = "./modules/webserver"

  # providing values to variables as arguments
  # parent variable have "parent" keyword
  ami           = var.parent-ami-id
  instance_type = var.parent-instance-type
  key_name      = var.parent-instance1-key
  name          = var.parent-instance1-name
  environment   = var.parent-instance1-env
  sg-id         = aws_security_group.second-sg.id

} # closing module1 block


module "web-server2" {
  # providing directory of child module
  source = "./modules/webserver"

  # providing values to variables as arguments
  # parent variable have "parent" keyword
  ami           = var.parent-ami-id
  instance_type = var.parent-instance-type
```

```
  key_name      = var.parent-instance2-key
  name          = var.parent-instance2-name
  environment   = var.parent-instance2-env
  sg-id         = aws_security_group.second-sg.id

} # closing module1 block
```

file - security-group.tf

```
resource "aws_security_group" "second-sg" {
  # name can be different
  name        = "second-sg"
  description = "Eating chilli and be chill"

  # we are creating dynamic block for port
  # 3 magical points - array of values - blueprint of content - using values port.value (iterator = port)
  dynamic "ingress" {

    # we are using for loop, ports - ssh, http, https, mongodb, mysql
    for_each = [22, 80, 443, 3306, 27017]
    # wow we are assingning values from left to right
    iterator = port

    # our blueprint block
    content {
      description = "TLS from VPC"
      from_port   = port.value
      to_port     = port.value
      protocol    = "tcp"

      # by mistake I written - cird - wrong
      # this attribut takes array
      cidr_blocks = ["0.0.0.0/0"]
    } # closing content block
  }   # closing dynamic ingress block
}     # closing aws sg resource block
```

file - terraform.tfvars

```
parent-ami-id         = "ami-04a81a99f5ec58529"
parent-instance-type  = "t2.micro"
parent-instance1-key  = "my-key1"
parent-instance2-key  = "my-key2"
parent-instance1-name = "first-server"
parent-instance2-name = "second-server"
parent-instance1-env  = "development"
parent-instance2-env  = "production"
```

file - output.tf

```
output "instance-ip" {
    value = aws_instance.my-webs.public_ip
}
```

file - outputs.tf

```
output "my-instnce-ip" {
    value = <<-EOF
            To get the access use the following command -
            ssh -i private-key ubuntu@${module.web-server1.instance-ip} \n
            ssh -i private-key ubuntu@${module.web-server2.instance-ip} \n
            Thankyou \n
            EOF
}
```

file - ec2.tf

```
# creating instance resource
resource "aws_instance" "my-webs" {
  # this time ami is argument in this block & var.ami is child variable
  ami           = var.ami
  instance_type = var.instance_type
  # ____ instance key pair
```

```
  key_name = aws_key_pair.my-key-pair.key_name

  # setting the security group, it takes arry
  vpc_security_group_ids = [var.sg-id]

  tags = {
    Name        = var.name
    environment = var.environment
  }
} # closing instance resource block
```

file - key.tf

```
# creating key pair resource - module ka resource
resource "aws_key_pair" "my-key-pair" {
  # we will give distict keys uing different names
  key_name = var.key_name

  # testing - reading file of external directory
  public_key = file("${path.module}/../../public-key.pub")
} # closing key pair resource block
```

file - variable.tf

```


variable "key_name" {
  type = string
}

variable "ami" {
  type = string
}

variable "instance_type" {
  type = string
}

variable "name" {
  type        = string
  description = "It is the name of instance that will be assigned to name tag"
}

variable "environment" {
  type = string
}

variable "sg-id" {
  type = string
}
```

Step 3: Execute your terraform script

```
# run the command at root level

terraform init
terraform validate
terraform fmt ./*
terraform plan --aut-approve
terraform apply --auto-approve
terraform destroy --auto-approve

# sometime given command doesn't work , try without --auto-approve and press enter enter enter
```

Lessons

- Attributes of Resources, created at root level is not accessible in the module. However we can pass the resource attributes as arguments to modules (values in string format)
- We if we are using the variables inside the variables, we need to define it twice (at module level as well as root level)

- Some arguments takes array, be careful
- terraform.tfvars file ko read nhi karra hai, chutiya kahinka
- `terraform apply --auto-approve` likhne ke baad bhi sala inputs leta hai. It needs default values. Don't worry, your default values won't be used, the values of terraform.tfvars would be only used. I gave default values - "sorry" for all the variables.
-

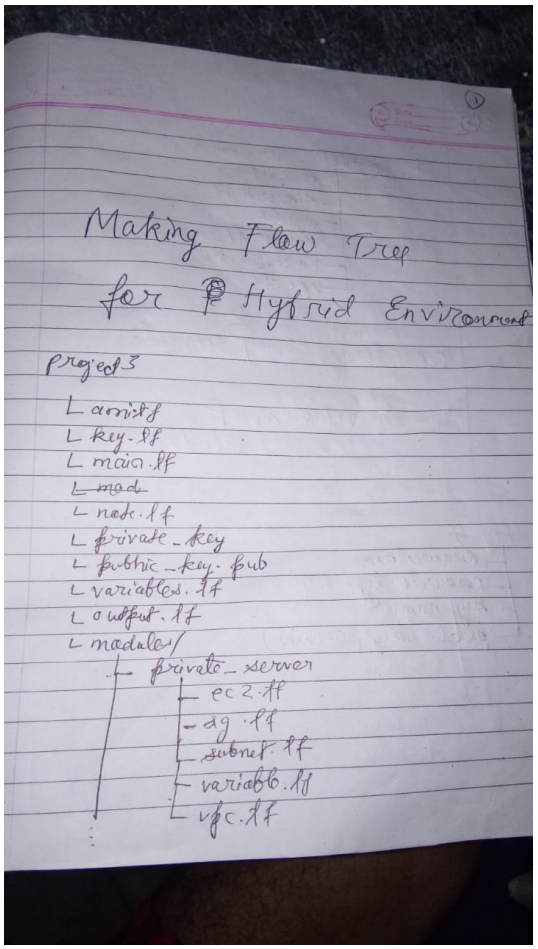
9. Practical Problem

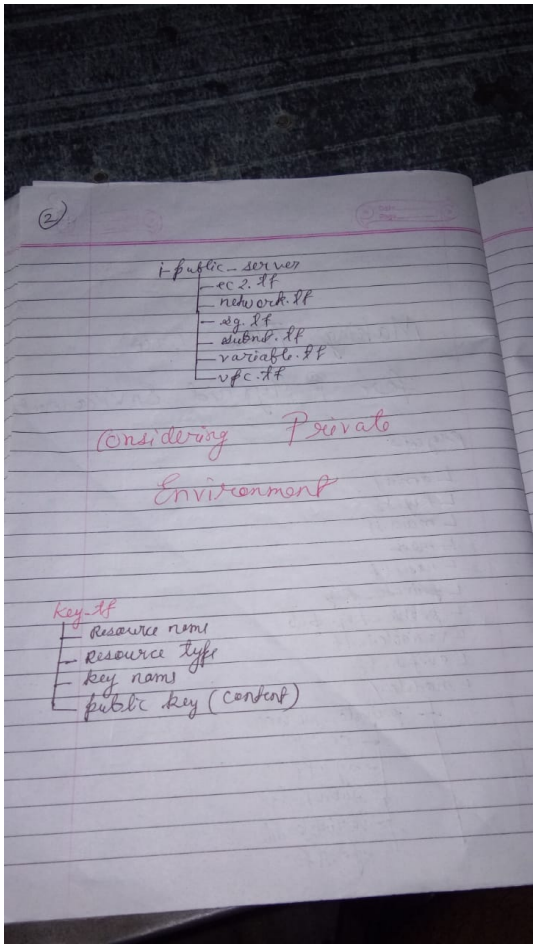
Problem Statement - Create 5 VPC, 4 Private subnets and 1 public subnet Create 1 EC2 instance in each of subnet. Do all of this using Terraform. Then create a Transit Gateway and attatch all the private subnets to that Transit Gateway (using Route table approach)

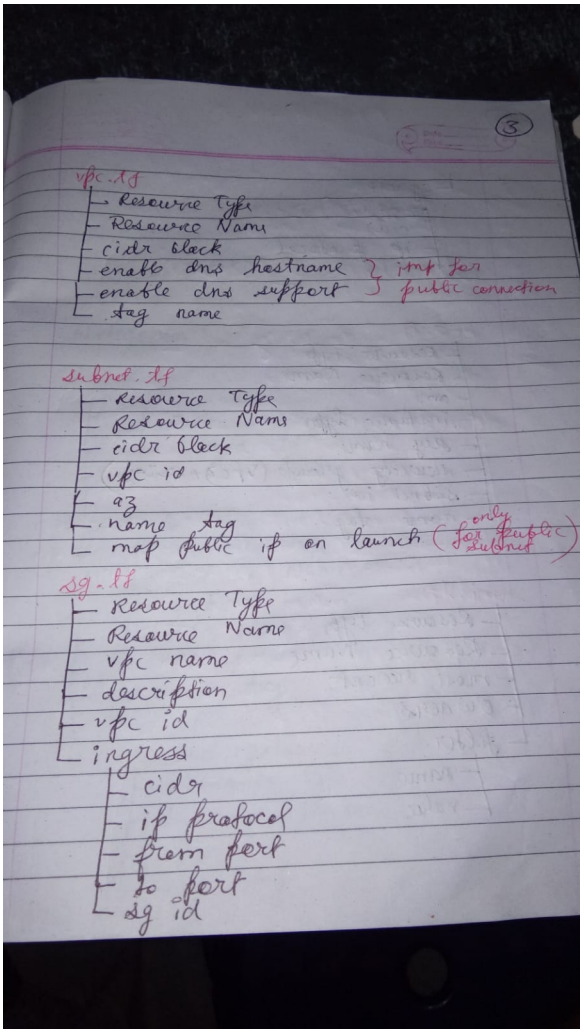
procedure

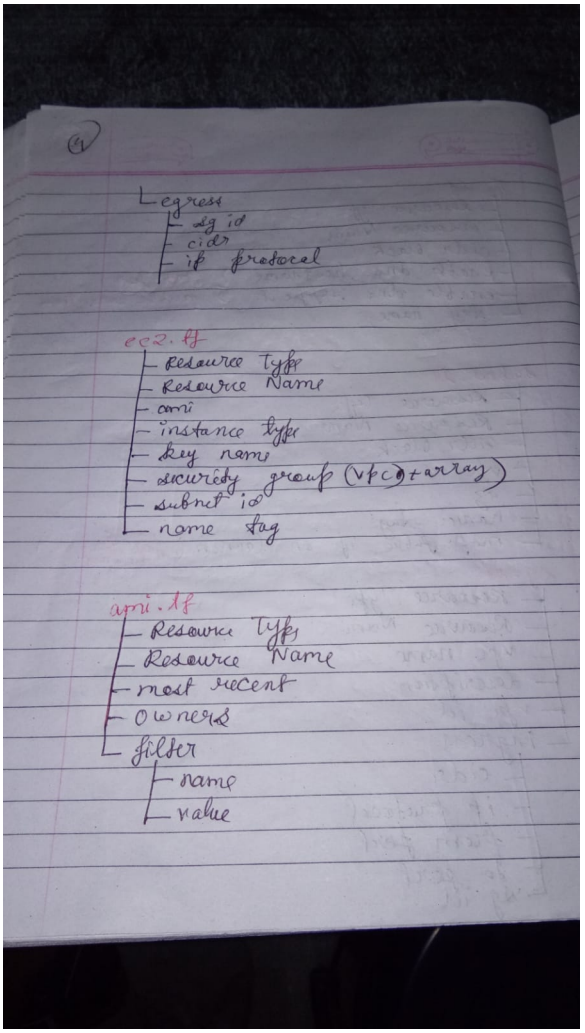
Step 1:

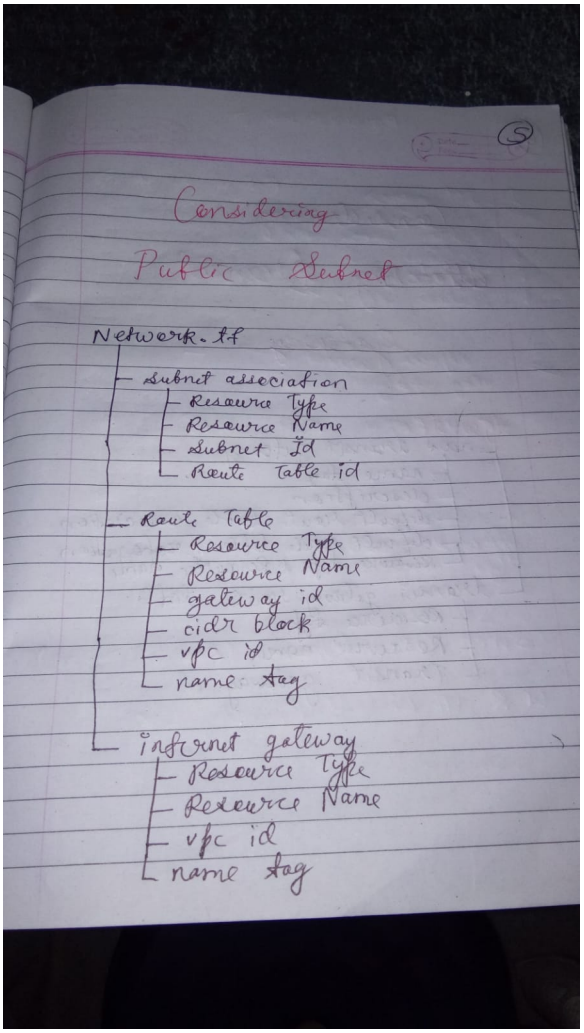
use the following structure to create the infrastructure

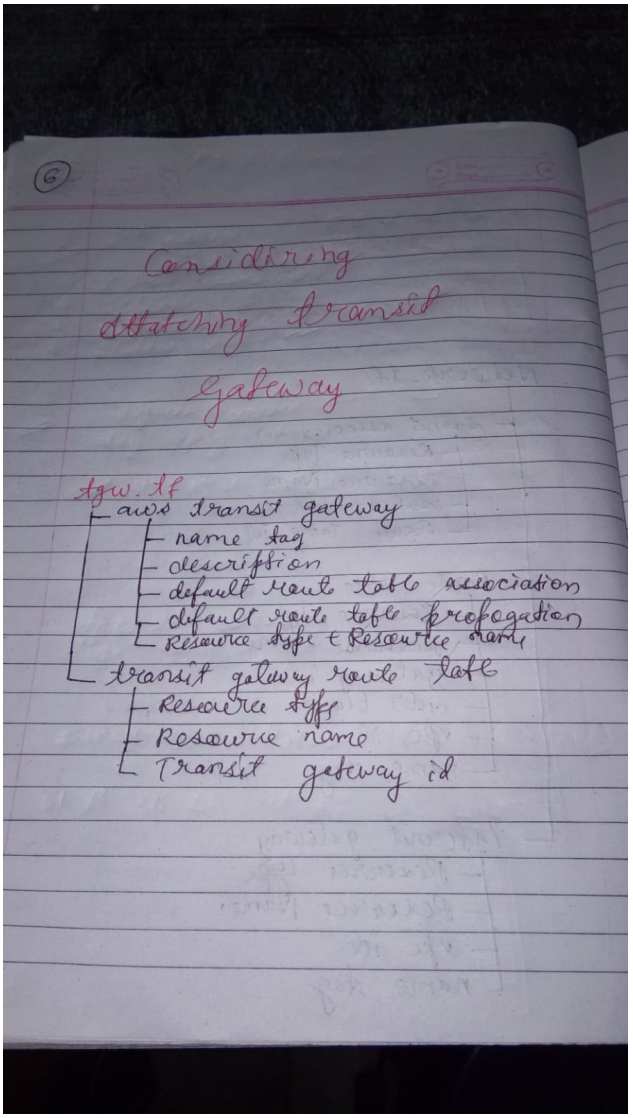


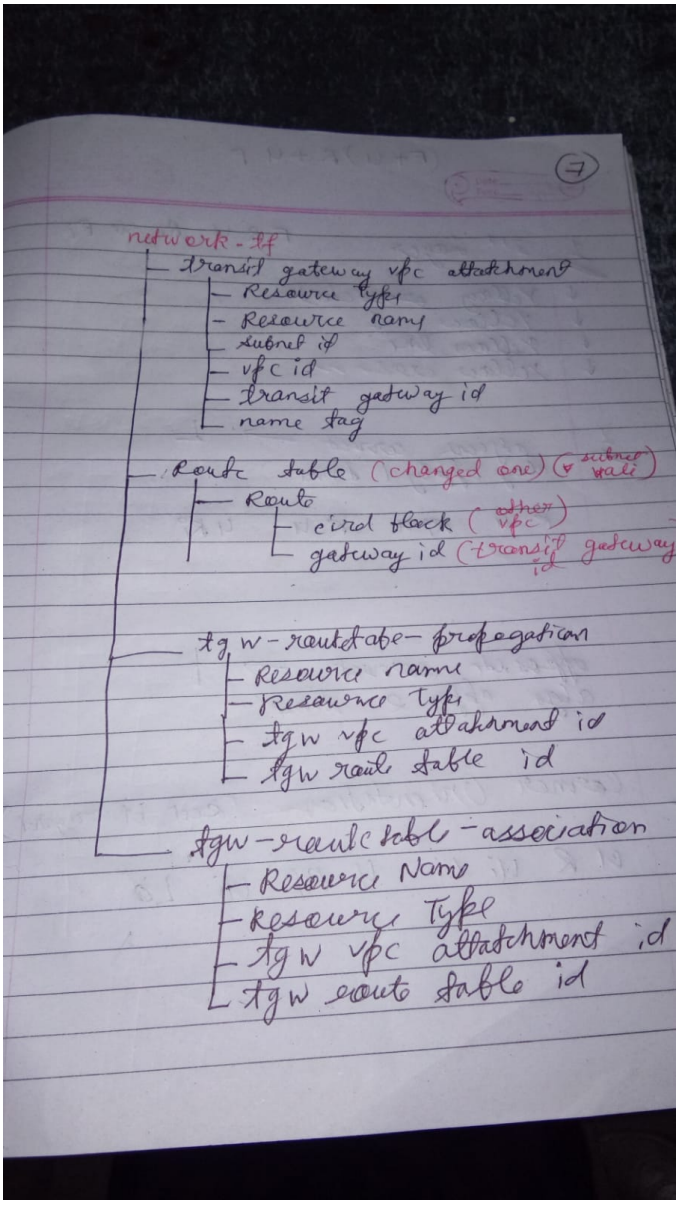

2
i-public-server
ec2.tf
network.tf
sg.tf
subnet.tf
variable.tf
vpc.tf
Considering Private
Environment
key.tf
Resource name
Resource type
key name
public key (content)


3
vpc.tf
Resource Type
Resource Name
cidr block
enable dns hostname } imp for
enable dns support } public connection
tag name
subnet.tf
Resource Type
Resource Name
cidr block
vpc id
az
name tag
map public ip on launch (only for public subnet)
sg.tf
Resource Type
Resource Name
vpc name
description
vpc id
ingress
cidr
ip protocol
from port
to port
sg id


egress
sg id
cidr
ip protocol
ec2.tf
Resource Type
Resource Name
ami
instance type
key name
security group (vpc)+array
subnet id
name tag
ami.tf
Resource Type
Resource Name
most recent
Owners
filter
name
value


Considering
Public Subnet
Network.tf
subnet association
Resource Type
Resource Name
Subnet Id
Route Table id
Route Table
Resource Type
Resource Name
gateway id
cidr block
vpc id
name tag
internet gateway
Resource Type
Resource Name
vpc id
name tag


Considering
Attaching transit
gateway
tgw.tf
aws transit gateway
name tag
description
default route table association
default route table propogation
Resource type & Resource name
transit galway route table
Resource type
Resource name
Transit gateway id



Step 2

Create the files as following

```
├── ami.tf
├── key.tf
├── main.tf
├── modules
│   ├── private_server
│   │   ├── ec2.tf
│   │   ├── network.tf
│   │   ├── output.tf
│   │   ├── sg.tf
│   │   ├── subnet.tf
│   │   ├── variable.tf
│   │   └── vpc.tf
│   └── public_server
│       ├── ec2.tf
│       ├── netowrk.tf
│       ├── output.tf
│       ├── sg.tf
│       ├── subnet.tf
│       ├── variable.tf
│       └── vpc.tf
├── note
```

```
├── outputs.tf
├── private-key
├── providers.tf
├── public-key.pub
├── tgw.tf
└── variables.tf
```

ami.tf

```
# Get latest Ubuntu Linux Trusty 14.04 AMI
data "aws_ami" "tf_ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical

  # only one filter is sufficient
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-*"]
  }
}
```

key.tf

```
# creating key pair resource - module ka resource
resource "aws_key_pair" "tf_key_pair" {
  # we will give distict keys uing different names
  key_name = var.p_key_name

  # testing - reading file of external directory - test passed
  public_key = file("${path.module}/public-key.pub")
} # closing key pair resource block
```

main.tf

```
module "public_web_1" {
  # providing directory of child module
  source = "./modules/public_server"
  # providing values to variables as arguments
  # parent variable have "parent" keyword
  ami               = data.aws_ami.tf_ubuntu.id
  instance_type     = "t2.micro"
  key_name          = var.p_key_name
  instance_name     = "public_instance_1_from_tf"
  env               = "prod"
  vpc_name          = "vpc_1_from_tf"
  sg_name           = "sg_1_from_tf"
  igw_name          = "igw_1_from_tf"
  route_table_name  = "route_table_1_from_tf"
  cidr_block        = "10.11.0.0/16"
  subnet_cird_block = "10.11.0.0/24"
  subnet_name       = "public_subnet_1_from_tf"


  # giving transit gateway id
  tgw_id = aws_ec2_transit_gateway.tf_tgw.id

  # giving cidrs for other vpc for tgw attatchments
  cidrs = ["10.12.0.0/16", "10.13.0.0/16", "10.14.0.0/16", "10.15.0.0/16"]

  # giving name to transit attatchment
  tgw_vpc_attatchment_name = "tgw_vpc_attatchment_to_vpc_1"

  # passing transit gateway route table id
  tgw_route_table_id = aws_ec2_transit_gateway_route_table.tf_tgw_route_table.id




} # closing module1 block



module "private_web_2" {
  source            = "./modules/private_server"
  ami               = data.aws_ami.tf_ubuntu.id
```

```
  key_name          = var.p_key_name
  instance_name     = "private_intance_2_from_tf"
  env               = "test"
  cidr_block        = "10.12.0.0/16"
  vpc_name          = "vpc_2_from_tf"
  subnet_cird_block = "10.12.0.0/24"
  subnet_name       = "private_subnet_2_from_tf"
  sg_name           = "sg_2_from_tf"

  # giving name to route table
  route_table_name = "rt_2_from_tf"

  # giving cidrs for other vpc for tgw attatchments
  cidrs = ["10.11.0.0/16", "10.13.0.0/16", "10.14.0.0/16", "10.15.0.0/16"]

  # giving transit gateway id
  tgw_id = aws_ec2_transit_gateway.tf_tgw.id


  # giving name to transit attatchment
  tgw_vpc_attatchment_name = "tgw_vpc_attatchment_to_vpc_2"

  # passing transit gateway route table id
  tgw_route_table_id = aws_ec2_transit_gateway_route_table.tf_tgw_route_table.id

}

module "private_web_3" {
  source            = "./modules/private_server"
  ami               = data.aws_ami.tf_ubuntu.id
  key_name          = var.p_key_name
  instance_name     = "private_intance_3_from_tf"
  env               = "test"
  cidr_block        = "10.13.0.0/16"
  vpc_name          = "vpc_3_from_tf"
  subnet_cird_block = "10.13.0.0/24"
  subnet_name       = "private_subnet_3_from_tf"
  sg_name           = "sg_3_from_tf"


  # giving name to route table
  route_table_name = "rt_3_from_tf"


  # giving cidrs for other vpc for tgw attatchments
  cidrs = ["10.11.0.0/16", "10.12.0.0/16", "10.14.0.0/16", "10.15.0.0/16"]

  # giving transit gateway id
  tgw_id = aws_ec2_transit_gateway.tf_tgw.id


  # giving name to transit attatchment
  tgw_vpc_attatchment_name = "tgw_vpc_attatchment_to_vpc_3"

  # passing transit gateway route table id
  tgw_route_table_id = aws_ec2_transit_gateway_route_table.tf_tgw_route_table.id

}

module "private_web_4" {
  source            = "./modules/private_server"
  ami               = data.aws_ami.tf_ubuntu.id
  key_name          = var.p_key_name
  instance_name     = "private_intance_4_from_tf"
  env               = "test"
  cidr_block        = "10.14.0.0/16"
  vpc_name          = "vpc_4_from_tf"
  subnet_cird_block = "10.14.0.0/24"
  subnet_name       = "private_subnet_4_from_tf"
  sg_name           = "sg_4_from_tf"

  # giving name to route table
  route_table_name = "rt_4_from_tf"


  # giving cidrs for other vpc for tgw attatchments
  cidrs = ["10.11.0.0/16", "10.12.0.0/16", "10.13.0.0/16", "10.15.0.0/16"]
```

```
  # giving transit gateway id
  tgw_id = aws_ec2_transit_gateway.tf_tgw.id


  # giving name to transit attatchment
  tgw_vpc_attatchment_name = "tgw_vpc_attatchment_to_vpc_4"

  # passing transit gateway route table id
  tgw_route_table_id = aws_ec2_transit_gateway_route_table.tf_tgw_route_table.id

}

module "private_web_5" {
  source            = "./modules/private_server"
  ami               = data.aws_ami.tf_ubuntu.id
  key_name          = var.p_key_name
  instance_name     = "private_intance_5_from_tf"
  env               = "test"
  cidr_block        = "10.15.0.0/16"
  vpc_name          = "vpc_5_from_tf"
  subnet_cird_block = "10.15.0.0/24"
  subnet_name       = "private_subnet_5_from_tf"
  sg_name           = "sg_5_from_tf"


  # giving name to route table
  route_table_name = "rt_5_from_tf"


  # giving cidrs for other vpc for tgw attatchments
  cidrs = ["10.11.0.0/16", "10.12.0.0/16", "10.13.0.0/16", "10.14.0.0/16"]

  # giving transit gateway id
  tgw_id = aws_ec2_transit_gateway.tf_tgw.id


  # giving name to transit attatchment
  tgw_vpc_attatchment_name = "tgw_vpc_attatchment_to_vpc_5"

  # passing transit gateway route table id
  tgw_route_table_id = aws_ec2_transit_gateway_route_table.tf_tgw_route_table.id


}
```

outputs.tf

```
output "get_public_ips" {
  value = <<-EOF

            use the following to get the ssh to ${module.public_web_1.intance_name_from_module}
            ssh -i private-key ubuntu@${module.public_web_1.public_ip_from_module}

            name: ${module.private_web_2.intance_name_from_module}
            ip: ${module.private_web_2.private_ip_from_module}

            name: ${module.private_web_3.intance_name_from_module}
            ip: ${module.private_web_3.private_ip_from_module}

            name: ${module.private_web_4.intance_name_from_module}
            ip: ${module.private_web_4.private_ip_from_module}

            name: ${module.private_web_5.intance_name_from_module}
            ip: ${module.private_web_5.private_ip_from_module}

  EOF
}
```

private-key

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAuCWOCV2beCP71zqjtF0PXZ7K0hO3uAnikL3xBHQ3Vwv98FlS+ltm
TPjYB5nof8ost1GBQ+oCq5gTj0WohdPhYlQIEQuZWizRChSeYiMNKWV40vw9tmbPGisGHH
6AHfji7xoJesmin//dKEQVZuL2q1myrczihuZvQLa0bmy/xx1zjZMY5rs6+z8hCEILbDRj
```

```
Pc/DAf2n3zJAlhpdsmZvjO2LTxZN+bLM9UH80H6DJWRZDpxGb+bRb3Neaqs65slLBhUEXC
JPyLOpMfp+tPk6gSLPXoNNrXYC6OeMUGluvkLYptRSvS4lOCAI3cEBw1vTOgfii3oNY9NR
8KdrmVJs+LMrTFF5zdo7Ofs2ooOBQX01iPkVm8rhRgslqsImOmhSXz9dZ4c5K7Jc4bwmyQ
A1vX9cfg0QA7cI8lYXCZnPrrjt0PSO6J6GWZQU2Qp9WcGDgQ5u8c7Qvs799bQhl1KmVqF1
uM5Jz3vDKUGU3rN2Od2o2FxMrPBxmZlIVqRxJ4+fAAAFiCUVSIclFUiHAAAAB3NzaC1yc2
EAAAGBALgljgldm3gj+9c6o7RdD12eytITt7gJ4pC98QR0N1cL/fBZUvpbZkz42AeZ6H/K
LLdRgUPqAquYE49FqIXT4WJUCBELmVos0QoUnmIjDSlleNL8PbZmzxorBhx+gB344u8aCX
rJop//3ShEFWbi9qtZsq3M4obmb0C2tG5sv8cdc42TGOa7Ovs/IQhCC2w0Yz3PwwH9p98y
QJYaXbJmb4zti08WTfmyzPVB/NB+gyVkWQ6cRm/m0W9zXmqrOubJSwYVBFwiT8izqTH6fr
T5OoEiz16DTa12AujnjFBpbr5C2KbUUr0uJTggCN3BAcNb0zoH4ot6DWPTUfCna5lSbPiz
K0xRec3aOzn7NqKDgUF9NYj5FZvK4UYLJarCJjpoUl8/XWeHOSuyXOG8JskANb1/XH4NEA
O3CPJWFwmZz6647dD0juiehlmUFNkKfVnBg4EObvHO0L7O/fW0IZdSplahdbjOSc97wylB
lN6zdjndqNhcTKzwcZmZSFakcSePnwAAAAMBAAEAAAGACtm57sJYDMSSmh2HpgVo47tmpe
mNqYxC/d+pBVWFu9P4kKbqwvtUvVPrgsoBxjVN3IDXUZEQKMnUbcpD5uUc3ereqM+ap7c8
wAaq2sI11UGvC2S3Hnmux6Uq6/J4ikJvJnm7ftXFdGx+GDTxFsDWLHxs7IHGTUfkAU6HSH
tIF5lAu71/Xnv3FhmRYxDS5sSHww2Q3dKqGdHtbVPzZCNWDuxq5KMHUVZXJTcG9n3KdcXw
JJcI5FWFhduWf9vWWryFuF98XbvDZo5u1WrWjH8IJFQNEG4XMwBI+pMSGMt8fDwRt5YK8m
Dxir+Q1dqvfdC+ua8TjMlvmd3vn3Z6mM5797tG43WU+kozk7nPAsHoQ2gp3NzCWj4GN+qB
gvnBreqxRlAguPCtO2HjTekIjXqlgnPcOOg8cWssYrJoXSg+Kzo1c1uAz+9jBiqTLZtFIm
zdsyubv7srW8WcnDcEY7hTtIpCxOFtm3N37bqyxWN48V0Y3OBWf8H80KAy/VfAtC2dAAAA
wC5IYW5Oxif4n00DsY2wz7Sk3Xs+Dy8PEuseiXE+t8WSTHBUM9aP7E41gQBubWifJAug5I
pSpmQllh6Vm6btszFwUc/PvvKe3mtKx9yiRRm39kwXtR3U6dMwXFJoA29g0hheqbgP4Sd5
xf9LKgUEUdMWQ83UevbS8vcyZp8FchmgDUP/6T2RHa6t1t4o/YAha8Pt1EdacQOVPYlum/
Z753/EpdeD7MGXRKa3dahcXIjiVizTHSLEvS1c6KjEDX9+LgAAAMEA2/YFIcmaiQUDyy9x
t2uM2hvL07YvWPkNGV/TuE+unbW5kJ9Wo55QdHmagkLsOWjiRZmKpXZ1y6G30r0M1WJn/e
7IdexIj+qUB7sgOBJZ8PBGRS/iMpFPU5zxR6PpttROk6bdvjH7Ek+Dvp6P5Cg8XDBVczPo
V6RPa6cCfduOnPOt/JL2zquLGZTB6/sth6Bq+WjT3FWniG+EFM33Wi9VGP1tnVtnMsFhz8
m0/0ibooNWlcMtiz49orbMEVTVhYStAAAAwQDWUVmBJfrvR6EZAX+eCe0NVBO/Us8JmNQJ
+AsSU0Fdbtyi+ilm8oEK2eW383J9uJ/UwFnsvS1uCJ6QsL5NmWFE0bUvqSpzXuMundV6Ng
JkbKHN7YPhlooJ1M15JIwxAl2mgHI36YwwPWPE7vRp4qJE7MsXTMZz4JnptgNo2Nb5YMeo
3h58dPTUi0O+CuRe3CDp3P6GS5QeV/zxHQ2t4OpmlDWyrujcel0G/v04d0fhpBcoLTDMD4
Pb4jGUU16NovsAAAAMc2hvYmhpdEBrYWxpAQIDBAUGBw==
-----END OPENSSH PRIVATE KEY-----
```

public-key.pub

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC4JY4JXZt4I/vXOqO0XQ9dnsrSE7e4CeKQvfEEdDdXC/3wwVL6W2ZM+NgHmeh/yiy3UYFD6gKrmBOPRaiF0+Fi
```

providers.tf

```
provider "aws" {
  region     = "us-east-1"
  secret_key = "GQiNUlYgmvXeGlwcZXVIhZJkQFqSsCS2VmsQ8TVh"
  access_key = "AKIA6ODU7FGKAGA4MLXO"
}
```

tgw.tf

```
# we are going to create a transit gateway
resource "aws_ec2_transit_gateway" "tf_tgw" {
  description                     = "I like alpenlibe lolipop"
  default_route_table_association = "disable"
  default_route_table_propagation = "disable"

  tags = {
    Name = "tgw_from_tf"
  }
}

# may by vpc to transit gateway attatchment is iski route table mei entry ho jati hogi
resource "aws_ec2_transit_gateway_route_table" "tf_tgw_route_table" {
  transit_gateway_id = aws_ec2_transit_gateway.tf_tgw.id
}
```

variables.tf

```
variable "p_key_name" {
  default = "key_pair_from_tf"
}

# we don't need to define the variable twice for our module
```

Step 3:

Now we will create private server module

ec2.tf

```
# copied from internet
resource "aws_instance" "tf_instance" {
  # this time ami is argument in this block & var.ami is child variable
  ami           = var.ami
  instance_type = var.instance_type
  # ____ instance key pair
  key_name = var.key_name
  # setting the security group, it takes arry
  vpc_security_group_ids = [aws_security_group.tf_sg.id]
  # providing subnet
  subnet_id = aws_subnet.tf_subnet.id
  # providing tags
  tags = {
    Name        = var.instance_name
    environment = var.env
  }
}
```

network.tf

```
# subnet association
resource "aws_route_table_association" "tf_route_table_association" {
  subnet_id      = aws_subnet.tf_subnet.id
  route_table_id = aws_route_table.tf_route_table.id
}

# creating resource -> route table
resource "aws_route_table" "tf_route_table" {
  vpc_id = aws_vpc.tf_vpc.id

  route {
    cidr_block = var.cidr_block # cidr for our vpc
    gateway_id = "local"        # allow traffic everywhere , inside our vpc boundry
  }

  # in this scenario, humko other vpc cidr pata hai, that's why hum usse hard code karre hai

  route {
    cidr_block         = var.cidrs[0]
    transit_gateway_id = var.tgw_id
  }
  route {
    cidr_block         = var.cidrs[1]
    transit_gateway_id = var.tgw_id
  }
  route {
    cidr_block         = var.cidrs[2]
    transit_gateway_id = var.tgw_id
  }
  route {
    cidr_block         = var.cidrs[3]
    transit_gateway_id = var.tgw_id
  }

  tags = {
    Name = var.route_table_name
  }
}

# VPC attatchment to transit gateway
resource "aws_ec2_transit_gateway_vpc_attachment" "tf_tgw_vpc_attatchment" {
  subnet_ids         = [aws_subnet.tf_subnet.id]
  transit_gateway_id = var.tgw_id
  vpc_id             = aws_vpc.tf_vpc.id

  tags = {
    Name = var.tgw_vpc_attatchment_name
  }
}

# Just like formality -> propagation and association

resource "aws_ec2_transit_gateway_route_table_propagation" "tf_tgw_propagation" {
```

```
  transit_gateway_attachment_id  = aws_ec2_transit_gateway_vpc_attachment.tf_tgw_vpc_attatchment.id
  transit_gateway_route_table_id = var.tgw_route_table_id
}

resource "aws_ec2_transit_gateway_route_table_association" "tf_tgw_association" {
  transit_gateway_attachment_id  = aws_ec2_transit_gateway_vpc_attachment.tf_tgw_vpc_attatchment.id
  transit_gateway_route_table_id = var.tgw_route_table_id
}
```

output.tf

```
# we can't access the values directly from outside the module, thus we have to throw the values from module
output "private_ip_from_module" {
  value = aws_instance.tf_instance.private_ip
}

# wow, tags in an array, we can provide any tag and also can access it
output "intance_name_from_module" {
  value = aws_instance.tf_instance.tags["Name"]
}
```

sg.tf

```
# creating security group resource at module level
resource "aws_security_group" "tf_sg" {
  name        = var.sg_name
  description = "Allow TLS inbound traffic and all outbound traffic"
  vpc_id      = aws_vpc.tf_vpc.id

}

# creating an ingress rule for getting ssh
resource "aws_vpc_security_group_ingress_rule" "ingress_for_ssh" {
  security_group_id = aws_security_group.tf_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}

# creating an ingress rule for icmp ping
resource "aws_vpc_security_group_ingress_rule" "ingress_for_icmp" {
  security_group_id = aws_security_group.tf_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = -1
  ip_protocol       = "icmp"
  to_port           = -1
}

# creating an egrss rule for all protocols
resource "aws_vpc_security_group_egress_rule" "egress_for_all" {
  security_group_id = aws_security_group.tf_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1" # semantically equivalent to all ports
}
```

subnet.tf

```
# creating subnet resource
resource "aws_subnet" "tf_subnet" {

  # privinding cidr block
  # mystry - this cidrsubnet() function takes an array
  # use - cidrsubnet(aws_vpc.test_env.cidr_block, 8)
  cidr_block = var.subnet_cird_block

  # providing vpc id
  vpc_id = aws_vpc.tf_vpc.id

  # providing az
  availability_zone = var.availability_zone

  # providing tag
```

```
  tags = {
    Name = var.subnet_name
  }
}
```

variable.tf

```
variable "tgw_route_table_id" {
  default = "sorry"
}

variable "tgw_vpc_attatchment_name" {
  default = "my_tgw_vpc_attatchment"
}
variable "cidrs" {
  default = ["0.0.0.0/0"]
}

variable "tgw_id" {
  default = "my-tgw"
}
variable "route_table_name" {
  default = "some"
}

variable "vpc_name" {
  default = "my-vpc"
}

variable "key_name" {
  default = "my-key"
}

variable "availability_zone" {
  default = "us-east-1a"
}

variable "subnet_name" {
  default = "my-subnet"
}

variable "sg_id" {
  default = "my-sg"
}

variable "instance_name" {
  default = "my-instance"
}

variable "env" {
  default = "my-env"
}

variable "instance_type" {
  default = "t2.micro"
}

variable "ami" {
  default = "some"
}


variable "cidr_block" {
  default = "some"
}

variable "subnet_cird_block" {
  default = "some"
}

variable "sg_name" {
  default = "my-sg"
}


# Types of variables
```

```
# 1. some from agrument
# 2. some taker form resource attribute
```

vpc.tf

```
# crating a vpc
resource "aws_vpc" "tf_vpc" {
  # providing cidr block
  cidr_block = var.cidr_block

  # it is important for transit gateway
  enable_dns_hostnames = true
  enable_dns_support   = true
  tags = {
    # providing name tag
    Name = var.vpc_name
  }
}
```

Step 4:

Now we will create public server module

ec2.tf

```
# copied from internet
resource "aws_instance" "tg_instance" {
  # this time ami is argument in this block & var.ami is child variable
  ami = var.ami

  instance_type = var.instance_type
  # ____ instance key pair
  key_name = var.key_name

  # setting the security group, it takes arry
  vpc_security_group_ids = [aws_security_group.tf_sg.id]

  # providing subnet
  subnet_id = aws_subnet.tf_subnet.id

  # providing tags
  tags = {
    Name        = var.instance_name
    environment = var.env
  }
}
```

network.tf

```
# subnet association
resource "aws_route_table_association" "tf_route_table_association" {
  subnet_id      = aws_subnet.tf_subnet.id
  route_table_id = aws_route_table.tf_route_table.id
}

# creating resource -> route table
resource "aws_route_table" "tf_route_table" {
  vpc_id = aws_vpc.tf_vpc.id

  route {
    cidr_block = var.cidr_block # cidr for our vpc
    gateway_id = "local"        # allow traffic everywhere , inside our vpc boundry
  }

  route {
    cidr_block = "0.0.0.0/0"                    # packet ka address nhi pata
    gateway_id = aws_internet_gateway.tf_igw.id # uss packet ko bahar fek do
  }

  # in this scenario, humko other vpc cidr pata hai, that's why hum usse hard code karre hai

  route {
    cidr_block         = var.cidrs[0]
    transit_gateway_id = var.tgw_id
  }
```

```
  route {
    cidr_block         = var.cidrs[1]
    transit_gateway_id = var.tgw_id
  }
  route {
    cidr_block         = var.cidrs[2]
    transit_gateway_id = var.tgw_id
  }
  route {
    cidr_block         = var.cidrs[3]
    transit_gateway_id = var.tgw_id
  }


  tags = {
    Name = var.route_table_name
  }
}

# creating resourece -> internet gateway
resource "aws_internet_gateway" "tf_igw" {
  # providing vpc id
  vpc_id = aws_vpc.tf_vpc.id

  tags = {
    Name = var.igw_name
  }
}

# VPC attatchment to transit gateway
resource "aws_ec2_transit_gateway_vpc_attachment" "tf_tgw_vpc_attatchment" {
  subnet_ids         = [aws_subnet.tf_subnet.id]
  transit_gateway_id = var.tgw_id
  vpc_id             = aws_vpc.tf_vpc.id

  tags = {
    Name = var.tgw_vpc_attatchment_name
  }
}


# Just like formality -> propagation and association

resource "aws_ec2_transit_gateway_route_table_propagation" "tf_tgw_propagation" {
  transit_gateway_attachment_id  = aws_ec2_transit_gateway_vpc_attachment.tf_tgw_vpc_attatchment.id
  transit_gateway_route_table_id = var.tgw_route_table_id
}

resource "aws_ec2_transit_gateway_route_table_association" "tf_tgw_association" {
  transit_gateway_attachment_id  = aws_ec2_transit_gateway_vpc_attachment.tf_tgw_vpc_attatchment.id
  transit_gateway_route_table_id = var.tgw_route_table_id
}
```

output.tf

```
output "public_ip_from_module" {
  value = aws_instance.tg_instance.public_ip
}

output "intance_name_from_module" {
  value = aws_instance.tg_instance.tags["Name"]
}
```

sg.tf

```
# creating security group resource at module level
resource "aws_security_group" "tf_sg" {
  name        = var.sg_name
  description = "Allow TLS inbound traffic and all outbound traffic"
  vpc_id      = aws_vpc.tf_vpc.id

}

# creating an ingress rule for getting ssh
resource "aws_vpc_security_group_ingress_rule" "ingress_for_ssh" {
  security_group_id = aws_security_group.tf_sg.id
  cidr_ipv4         = "0.0.0.0/0"
```

```
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}

# creating an ingress rule for icmp ping, by mistake i Provided ip_protocol = -1
resource "aws_vpc_security_group_ingress_rule" "ingress_for_icmp" {
  security_group_id = aws_security_group.tf_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = -1
  ip_protocol       = "icmp"
  to_port           = -1
}

# creating an egrss rule for all protocols

resource "aws_vpc_security_group_egress_rule" "egress_for_all" {
  security_group_id = aws_security_group.tf_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1" # semantically equivalent to all ports
}
```

subnet.tf

```
# creating subnet resource
resource "aws_subnet" "tf_subnet" {

  # prividing cidr block
  # mystry - this cidrsubnet() function gives an array
  # use - cidrsubnet(aws_vpc.test_env.cidr_block, 8)
  cidr_block = var.subnet_cird_block

  # providing vpc id
  vpc_id = aws_vpc.tf_vpc.id

  # it makes this a public subnet
  map_public_ip_on_launch = true

  # providing az
  availability_zone = var.availability_zone

  # providing tag
  tags = {
    Name = var.subnet_name
  }
}
```

variable.tf

```
variable "tgw_route_table_id" {
  default = "sorry"
}


variable "tgw_id" {
  default = "sorry"
}

variable "tgw_vpc_attatchment_name" {
  default = "sorry"
}

variable "cidrs" {
  default = ["0.0.0.0/0"]
}
variable "route_table_name" {
  default = "my-route-table"
}

variable "igw_name" {
  default = "my-igw"
}

variable "vpc_name" {
  default = "my-vpc"
```

```
}

variable "sg_name" {
  default = "my-sg"
}

variable "key_name" {
  default = "my-key"
}

variable "availability_zone" {
  default = "us-east-1a"
}

variable "subnet_name" {
  default = "my-subnet"
}

variable "sg_id" {
  default = "my-sg"
}

variable "instance_name" {
  default = "my-instance"
}

variable "env" {
  default = "my-env"
}

variable "instance_type" {
  default = "t2.micro"
}

variable "ami" {
  default = "some"
}


variable "cidr_block" {
  default = "some"
}

variable "subnet_cird_block" {
  default = "2.2.0.0/24"
}


# Types of variables
# 1. some from agrument
# 2. some taker form resource attribute
```

vpc.tf

```
# crating a vpc
resource "aws_vpc" "tf_vpc" {
  # providing cidr block
  cidr_block = var.cidr_block

  # it is important for transit gateway or connectivity among other services
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    # providing name tag
    Name = var.vpc_name
  }
}
```

Problems

```
| Error: creating VPC Security Group Rule
|
```

```
|   with module.web1.aws_vpc_security_group_ingress_rule.ingress_for_icmp,
|   on modules/webserver/sg.tf line 19, in resource "aws_vpc_security_group_ingress_rule" "ingress_for_icmp":
|   19: resource "aws_vpc_security_group_ingress_rule" "ingress_for_icmp" {
|
| InvalidParameterValue: TCP/UDP (from) port (-1) out of range
|       status code: 400, request id: 7f2fc278-6790-436c-8d84-637ced6b95cd
```

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-rules-reference.html#sg-rules-ping

my mistakes

1. I declared a variable 2 times
2. I did not write var before the variable
3. I have to privide the id of an resource but I did typo like "aws_security_group.tg-subnet" here I forgotten .id
4. For the ingress rule of ICMP, I did not provided the proper ports ⇒ -1 , last time it was 22, that's why it was showing duplicate
5. I had created a module resoruce (security goup). I have to give the sg id in module aspect (aws_security_group.tg_sg.id) but I was giving in root level aspect (sg_id).
6. I did not gave ip protocol = icmp, I had written tcp (for tcp, I can give the range)

Questions Section #2

How to install the terraform in linux/Debian

```
# Ensure that your system is up to date and you have installed the gnupg, software-properties-common, and curl packages
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common

# Install the HashiCorp GPG key.
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null


# Verify the key's fingerprint.
gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint

# Add the official HashiCorp repository to your system.
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list

# Download the package information from HashiCorp.
sudo apt update

# Install Terraform from the new repository.
sudo apt-get install terraform

# Verify the installation
terraform -help
```