Terraform Theory

Other Questions

1. Tell some popular IaC Tools

1. Terraform

An open-source declarative tool that offers pre-written modules to build and manage an infrastructure.

2. Chef:

A configuration management tool that uses
cookbooks and recipes to deploy the desired environment. Best used for
Deploying and configuring applications using a pull-based approach.

3. Puppet:

Popular tool for configuration management
that follows a Client-Server Model. Puppet needs agents to be deployed
on the target machines before the puppet can start managing them.

4. Ansible:

Ansible is used for building infrastructure
as well as deploying and configuring applications on top of them. Best
used for Ad hoc analysis.

5. Packer:

Unique tool that generates VM images (not running VMs) based on steps you provide. Best used for Baking compute images.

6. Vagrant:

Builds VMs using a workflow. Best used for Creating pre-configured developer VMs within VirtualBox.

2. What are the benefits of using the terraform

- Does orchestration, not just configuration management
- Supports multiple providers such as AWS, Azure, Oracle, GCP, and many more
- Provide immutable infrastructure where configuration changes smoothly
- Uses easy to understand language, HCL (HashiCorp configuration language)
- Easily portable to any other provider

3. Describe Terraform Lifecycle

Terraform lifecycle consists of – **init**, **plan**, **apply**, and **destroy**.

1. **Terraform init** initializes the (local) Terraform environment. Usually executed only once per session.
2. **Terraform plan** compares the Terraform state with the as-is state in the cloud, build and display an execution plan. This does not change the deployment (read-only).
3. **Terraform apply** executes the plan. This potentially changes the deployment.
4. **Terraform destroy** deletes all resources that are governed by this specific terraform environment.

4. Describe the core concepts of Terraform

1. Variables: Terraform has input and output
variables, it is a key-value pair. Input variables are used as
parameters to input values at run time to customize our deployments.
Output variables are return values of a terraform module that can be
used by other configurations.

Read our blog on **Terraform Variables**

2. Provider: Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, OCI, and others. A *provider* is a plugin that interacts with the various APIs required to create, update, and delete various resources.

Read our blog to know more about **Terraform Providers**

3. Module: Any set of Terraform configuration files in a folder is a *module*. Every Terraform configuration has at least one module, known as its ***root module.***

4. State: Terraform records information about what infrastructure is created in a Terraform *state* file. With the state file, Terraform is able to find the resources it created previously, supposed to manage and update them accordingly.

5. Resources: Cloud Providers provides various services in their offerings, they are referenced as Resources in Terraform. Terraform resources can be anything from compute instances, virtual networks to higher-level components such as DNS records. Each resource has its own attributes to define that resource.

6. Data Source: Data source performs a read-only operation. It allows data to be fetched or computed from resources/entities that are not defined or managed by Terraform or the current Terraform configuration.

7. Plan: It is one of the stages in the Terraform lifecycle where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.

8. Apply: It is one of the stages in the Terraform lifecycle where it applies the changes real/current state of the infrastructure in order to achieve the desired state.

Important Questions

1. Difference between Terraform and Ansible

Terraform is a tool designed to help with the provisioning and deprovisioning of cloud infrastructure using an infrastructure as code approach. It is highly specialized for this purpose. On the other hand, Ansible is a more general tool that can be used for automation across various domains. Both Terraform and ansible have strong open-source communities and commercial products that are well supported.

Terraform vs. Ansible: Highlighting the Differences. Terraform sets up and manages your IT infrastructure, using an infrastructure as code approach. Ansible, on the other hand, focuses on automating IT tasks like provisioning and deployment. In short: Use Terraform for infrastructure setup and Ansible for configuration. Both have strong open-source support and commercial options.

Difference between Terraform and Ansible Provisioning (Terraform vs. Ansible)

Let's see how the Terraform vs. Ansible battle differentiates from each other:

Terraform	Ansible
Terraform is a provisioning tool.	Ansible is a configuration management tool.
It follows a declarative Infrastructure as a Code approach.	It follows both declarative & procedural approaches.
It is the best fit for orchestrating cloud services and setting up cloud infrastructure from scratch.	It is mainly used for configuring servers with the right software and updating already configured resources.
Terraform does not support bare metal provisioning by default.	Ansible supports the provisioning of bare metal servers.
It does not provide better support in terms of packaging and templating.	It provides full support for packaging and templating.
It highly depends on lifecycle or state management.	It does not have lifecycle management at all.

2.

Lecture Questions - section #1

The following questions are been sourced from Terraform Series by Gaurav Sir. These questions are taken from videos no - #22 to #39.

1. Why it is suggested to try the process manually before doing the automation. How does it improves understanding of the concept

- **Clarity:** Humko concept ki clearity mil jati hai, ki actually me behind the scene kya ho raha hai. We come to know the dependencies and various terms jo humari process me involved hote hai.
- **Problem-solving:** Process ke during kon kon si problems aa sakti hai, humko idea mil jata hai. Isse agar humare automation process fail ho jata hai to humko idea hota hai ki background me kya problem ho sakti hai.
- **Efficiency:** Kisi bhi kaam ko karne ke multiple ways hote hai, hum sabse efficient way ko choose kar sakte hai and process ko aur bhi jada concise kar sakte hai.

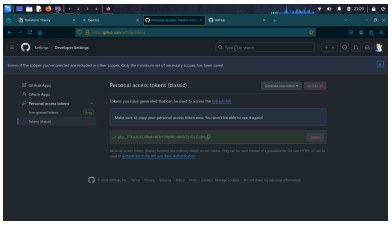
2. Describe the detailed steps to create the repository in GitHub via terraform

Step 1:

Prepare the environment, install the terraform, use the installation guidance https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

Step 2:

Generate GitHub Personal Access Token: You'll need a personal access token with appropriate permissions to interact with GitHub repositories. Go to your GitHub settings, navigate to "Developer settings" → "Personal access tokens" and create a new token. Grant it "repo" or "admin:repo" access depending on your needs. I had granted all the permissions.



Step 3:

Creating a terraform file for listing the provider - provider.tf

```
provider "github" {
  token = var.github_token
}
```

Step 4:

Creating a terraform file creating a repo resource-github-repo.tf

```
# Here Resource type is github_repository
# Here Name of resource ( used in terraform internal environment) is my_repo


resource "github_repository" "my_repo" {
  name        = var.repo_name        # For setting the name of repository
  description = var.repo_description # For setting the description of repo
  visibility     = var.repo-visibility     # For deciding whether your repo should be public or private

}
```

Step 5:

Defining the variables, which are being used in the terraform environment - variables.tf

```
variable "github_token" {}
variable "repo_name" {}
variable "repo_description" {}
variable "repo-visibility" {
    default = "public"
}
```

Step 6:

Setting the values of variables - terraform.tfvars # name of .tfvars file should be this one only

```
github_token = "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"
repo_name = "my-repo-from-terraform"
repo_description = "This repository shows successufl learning of terraform, congratualtions"
```

Step 7:

Format the terrafrom code

```
# run the followng command in command line
terraform fmt
```

Initialize the terraform providers

```
# Run the following code to install the plugins associate with the providers
terrafrom init
```

Step 8:

Run the configuration files

```
# validation the syntax of all the configuration
terraform validate

# See the plan, which resources are going to be created
terraform plan

# Now finally create the resources at providers side
terraform apply --auto-approve
```

3. Describe some Terraform commands

- **terraform plan:** Ye terraform ki current state ko current configuration ke saath compare karta hai and infrastructure mei jo changes hone wale hote hai, unko preview karta hai
- **terraform providers:** ye vo saare plugin ko list karta hai jo ki terraform ko various platforms se connect hone mei help karenge.
- **terraform init:** ye command all required plugins ko install karta hai jo ki humare configuration me defined hai. Also it initialize the terraform state file.
- **terraform destroy:** This powerful command **destroys the infrastructure** managed by your Terraform configuration. Hum kisi particular resource ko bhi destroy kar sakte hai `destroy --target <resource-name-at-local-environment>`
- **terraform fmt:** This command **formats your Terraform code** , ensures consistent formatting and readability for your Terraform configuration files
- **terraform validate:** This command **validates your Terraform configuration** for syntax errors and potential issues.
- **Terraform show** - This command is used to display information about your Terraform configuration or state in a human-readable format. You can store the output of this command in a particular file `terraform show > your_file_name`
- **Terraform refresh** - This command refresh the Terraform state to updated state of infrasturcture. It updates the values and id's which has been changed.
- **Terraform graph**
- **Terraform output**
- **Terrafrom console**
- **Terraform apply**
-

4. How to recreate the destroyed resource in Terraform

Run `terraform apply --auto-approve` just after `terraform destroy`

5. Assuming the scenario of Repository creation, give suggestions for file destructuring

- Structure the .tf file into many other files and folders according to resource type and its objects.
- Make use of variables instead of hard coding the values in it
- Make use of Variable Interpolation and Conditional Expressions
- Don't repeat the code, use loop or use code reusability for making the code more clear and understandable
- Provide the comments for better understanding of code

6. What is the use of following commands

8. Why it is suggested to use the command terraform plan before terraform apply

Here's why it's highly recommended to use `terraform plan` before running `terraform apply` in Terraform:

Safety and Transparency:

- **Preview Changes:** `terraform plan` provides a detailed preview of the changes isse humko idea mil jata hai ki infrasturcture me exactly kya kya resources create/modiy/delete hone wale hai and hum apne work ko lekar confident ho jate hai jo expected hai, vahi hone wala hai.
- **Error Detection:** `terraform plan` can often detect potential errors or misconfigurations in your Terraform code. Plan ke output ke through we can catch typos, resource conflicts, or missing dependencies before they execute.
- **Resource Cost Estimation:** Some cloud providers, like AWS, offer cost estimation during the `terraform plan` phase. This allows you to get an approximate idea of the potential costs associated with the planned infrastructure
- **Review by Others:** The `terraform plan` output can be shared with colleagues or stakeholders for review before applying the changes.
- **Prevent Accidental Destruction:** Human error or unexpected behavior can sometimes lead to unintended resource deletion during the `terraform apply` phase. By reviewing the plan first, you can verify that no resources are being unintentionally destroyed and avoids accidental data loss.

9. Where does the output comes after executing `terraform apply` command

Our output will be seen at tfstate file. Further we can see the output agar humko output block ka naam pata ho. We can use the command `terraform output <output-block-name>`

10. How to print the values of variables (in current state) from our current working directory

We can use the command terraform console . Here we will enter to some other terraform env shell, which can provide details directly to us. If we used a variable - github_token. Then we can see its value -

```
# accessing our terraform terminal
terraform console

# getting the value of our variable
> var.github_token
"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"

# above line will look like output.
```

11. How to add aws provider to Terraform. Also tell how does AWS would authenticate the Terraform user using keys.

some codes

```
# Configure the AWS provider
provider "aws" {
  region = "us-east-1"  # Replace with your desired region
}

# Define the EC2 instance resource
resource "aws_instance" "my_instance" {
  ami           = "ami-0e8217805e34b49e0"  # Replace with a valid AMI ID
  instance_type = "t2.micro"
  # Add other configuration options as needed (e.g., security groups, tags)
}


# Configure the AWS provider
provider "aws" {
  region = "us-east-1" # Replace with your desired region
}

# Read the public key content from a file
data "file" "public_key" {
  filename = "/my-key/public_key.pem" # Replace with your public key path
```

```
}

# Create a key pair resource with the public key content
resource "aws_key_pair" "my_key_pair" {
  key_name   = "my-key-pair"
  public_key = data.file.public_key.content
}

# Create a security group
resource "aws_security_group" "allow_ssh_web_https" {
  name = "allow-ssh-web-https"
  description = "Security group for SSH, web, and HTTPS access"

  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
  }
Lecture Questions - section #1
  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"]
  }
}
```

Lecture Questions - section #2

1. How to give values to terraform variable from environment variables of our system.

making environment variable using prefix - IF_VAR_<your-variable>

2. What is the use of terraform taint command

`terraform taint` command ke through hum resource ko jo ki unexpected state acqure kar liya hai, usko corrupted mark kar sakte hai. Later jab hum `terraform apply` command chalayenge then terraform humare particular resource ko destroy karke uska new create kar dega.

The `terraform taint` command has been deprecated by HashiCorp in Terraform versions 0.15.2 and later. The recommended approach for forcing resource recreation is to use the `-replace` option with the `terraform apply` command. Here's the syntax:

`terraform apply -replace <resource_identifier>`

3. What is Argment reference and Attribut Reference in Terraform documentation

Argment reference

Ye humko various parameters ke baare me batata hai jo ki hum kisi rescource ko create karne ke during banate hai.

Attribut Reference

Kisi resource ko create karne ke baad usser related details jin variables mei automatically store hoti hai, those are Attribute References.

4. Where does “user data” is found in Terraform documentation for creating EC2 instance.

Under Argument Reference

5. How to write multi line data in Terraform

Multi line data likhne ke liye jo approach use hoti hai, that is known as Heredoc Syntax. Here hum 2 less than sign ka use karte hai (`<<`) , followed by an End of File identifier (EOF). Optionally, you can add a hyphen (`-`) after the double less-than signs for indented heredocs.

```
# We can use both EOT or EOF. We can use this for storing linux commands.
variable "multiline_data_with_indent" {
  type = string
  default = <<-EOT
    This is the first line of my multi-line data,
    with indentation.
    This is the second line, still indented.
    EOT
}
```

6. See following questions from picture notes

1. Where does html file is located in Nginx
2. What 2 mistakes sir did , during providing user data to create EC2 instance
3. How to check how many ports are open. Linux machine is listning which ports.
4. What is the use of configuration management tool over terraform
5. What is the meaning of error - missing connection configuration for provisioner .
6. Which resources are created first in terraform
7. Agar hum miltiple provisioner de dete hai to provisioners kis order mei execute hote hai.
8. Write local-exec provisioner such that jab tum resource create karo to perticular bash command run hoye and jab tum resource delete karo to (`terraform destroy --auto-approve`) then some other bash command run hoye.
9. What happens if `terraform apply` command run karne ke baad local-exec provisioner fail ho jata hai
10. How to do, local-exec provisioner fail hone ke baad bhi `terraform apply` command run hoye
11. Sir explained why not to use provisioners.
12.

7. How to read a file in terraform

```
# (Approach 1) By implementing the concept of data source

# Creating a data source
data "file" "my_config_file" {
  filename = "path/to/my/config.txt"
}

# Using data source and storing content into a variable
variable "config_data_value" {
  type = string
  default = data.file.my_config_file.content
}

# (Approach 2) , By reading the file using directly using file function. We are providing user data to EC2 instance
user_data = file ("${path.module}/path/from/current/directory/script.sh")
```

8. What are terraform provisioners.

Teffaform provisioners vo scripts or commands hoti hai jo ki local machine or remote resources par tab execute hoti hai jab vo create ya update hoti hai. There are three main types of built-in provisioners in Terraform:

local-exec:

Executes a shell command on the Terraform machine (where Terraform is running). Ye provisioner Local environment par dyanmic configuration ko manage karne ke liye use hota hai.

```
# Waah , hum null resoruce create karre hai
# For running the script in our local environmnent
resource "null_resource" "prepare_config" {

  provisioner "local-exec" {
    command = "scripts/generate_config.sh"
  }
}
```

remote-exec:

Executes a shell command on a remotely provisioned resource using SSH or WinRM. Isse hum software ko install kar sakte hai, services ko configure kar sakte hai and post-deployment script ko target machine par run kara sakte hai.

```
# You can see that hum EC2 instance resource ko create karne ke during hi remote wala prisoner (pagal)  create karre hai
resource "aws_instance" "my_server" {
  # ... instance configuration

  provisioner "remote-exec" {
    connection {
      type = "ssh"
      host = aws_instance.my_server.public_ip
      user = "ubuntu"
    }
    command = "apt update && apt install -y nginx"
  }
}
```

file:

Transfers a file from the Terraform configuration directory to a remote resource. Iske through hum terraform configration directory se remote resource tak file ko transfer kar sakte hai.

```
# We are transferring a file to an EC2 instance
resource "aws_instance" "my_server" {
  # ... instance configuration

  provisioner "file" {
    source      = "nginx.conf"
    destination = "/etc/nginx/nginx.conf"

    # isme private key nhi hai, for taking ssh to remote machine
    connection {
      type = "ssh"
      host = aws_instance.my_server.public_ip
      user = "ubuntu"
    }
  }
}
```

9. How to print if address of an instance in terraform, how to use terraform documentation, how to print the value of output block.

Hum resource (EC2 instance) se related information dekhne ke liye terrafrom documentation mei Attribute Reference ko prefer karenge. I will tell us the variable jo ki instance ka ip address hold kiya hai. At Terraform documentation we will go to **Resource: aws_instance.** Vaha par **Attribut Reference** par hu **public_ip** attribute dekhte hai. We can use it to get the ouput like :

```
# while creating instance resource
resource "aws_instance" "my_server" {
  # ... instance configuration
}

# Now our instance is created
# We want to see the public ip address using output block
output "web_server_public_ip" {

    # Here we pick up variable - public_ip from Attribute Reference from Terraform documentation
  value = aws_instance.web_server.public_ip
  description = "Public IP address of the web server instance"
}
```

Now we want to see the value of output block. Jab hum `terraform apply` command chalayange then Terraform will display a summary of the changes made and the values of all defined output blocks.

```
# Typing command at terminal
terraform apply

# we will get output as following
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:
    web_server_public_ip = <actual public IP address>
```

Hum output block ki value manually bhi dekh sakte hai

```
# Typing command at terminal
terraform output web_server_public_ip
```

10. Write a provisioner (code) to transfer complete folder from Terraform configuratin directory to remote machine. (unsolved)

11. See Tutorial again and write code to ansere the following (unsolved)

1. How does terraform solves deadlock condition. Give example for the implementation of self dependency
2. Suppose we create multiple provisioners and ek provisioner ke andar ka block multiple times repeate ho raha hai.

12. How to use command, work_dir, interpreter argument with local-exec provisioner

command and work_dir

```
# we are creating null resource as we are executing command at local, by using local-exec provisioner
resource "null_resource" "prepare_config" {

  provisioner "local-exec" {
    command = "sh prepare_config.sh"  # We provided the script, or else we can provide the script hard code
    work_dir = "scripts"               # Execute in the "scripts" directory
  }
}
```

command and interpreter

```
# we are creating null resource as we are executing command at local, by using local-exec provisioner
resource "null_resource" "prepare_config" {

  provisioner "local-exec" {

    # we provided interpreter
    interpreter = [
"/usr/bin/python3","-c"
]
    # We provided command
    command = "print('Hello world')"
  }
}
```

command and environment variable

```
# we are creating null resource as we are executing command at local, by using local-exec provisioner
resource "null_resource" "prepare_config" {

  provisioner "local-exec" {

    # We provided command, storing all env variabls in a file
    command = "env>env.txt"

    # Creating new environment variables
    environment = {
    envname = "envvalue"
    }
  }
}
```

13. Write a remote-exec provisioner with the use of argument - inline & script

`inline` **Argumen**

iss argument ke through hum terraform configuration mei directly shell command ki list create kar sakte hai. Then jo jis sequesnce mei hai, ussi sequence mei execute hoti hai on remote server.

```
# You can see that hum EC2 instance resource ko create karne ke during hi remote wala prisoner (pagal)  create karre hai
resource "aws_instance" "my_server" {
  # ... instance configuration

  provisioner "remote-exec" {
    connection {
      type = "ssh"
      host = aws_instance.my_server.public_ip
      user = "ubuntu"
    }
    inline = [
      "apt update",
      "apt install -y nginx",
      "systemctl start nginx",
    ]
  } # Closing provisioner block
}  # Closing resource block
```

`script` Argument:

iss argument ke through hum local script file ka reference dete hai , jisme commands present hoti hai , jo ki remote resource par execute hoti hai. Special baat ye hai ki ye kaam kese karti hai - Terraform sabse pahale script file ko remote machine par upload karta hai then usko execute karta hai.

```
# You can see that hum EC2 instance resource ko create karne ke during hi remote wala prisoner (pagal)  create karre hai
resource "aws_instance" "my_server" {
  # ... instance configuration

  provisioner "remote-exec" {
    connection {
      type = "ssh"
      host = aws_instance.my_server.public_ip
      user = "ubuntu"
    }

    # you can see, by this way humne script file ka path select kiya hai
    script = "path/to/my/script.sh"

  } # Closing provisioner block
}  # Closing resource block
```

14. How to get ami-id of an instance avoiding hard coding of ami-id's

```
# fetching ami-id from aws datasource name = aws_ami, it measn humko ami se related data  chahiye
# you will see "ubuntu", it means humko "ubuntu"  image chahiye
data "aws_ami" "ubuntu" {
  # humko latest image chahiye
  most_recent = true

  # hum image ko owner ki values array me provide karre hai
  owners = ["099720109477"]

  # setting filters, ami name -> ubuntu, root device type -> ebs, virtualization type -> hvm
  filter {
    name = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-arm-64-server-*"]
  }
  filter {
    name = "root-device-type"
    values = ["ebs"]
  }
  filter {
    name = "virtualization-type"
    values = ["hvm"]
  }
} # closing data source block

# finally getting our ami id in output block
output ami_id {
  values = data.aws_ami.ubuntu.id
}
```

15. Implement dynamic image_id with fixed image name

```
# creating data source block
data "aws_ami" "ubuntu" {
  # getting latest image
  most_recent = true
  owners = ["099720109477"]

  # 3 filters lagana tk nhi hai, it's better ki only name filter lagaye and aws console se full name copy kar le
  filter {
    name = "name"
    # using fixed image name defined in terraform.tfvars file
    values = ["${var.image_name}"]
  }

  filter {
    name = "root-device-type"
    values = ["ebs"]
  }

  filter {
    name = "virtualization-type"
    values = ["hvm"]
  }

} # closing data source block

# Creating aws instance resource
resource "aws_instance" "web" {
  # implementing data source to get dynamic ami-id
  ami = "${data.aws_ami.ubuntu.id}"

  # instance type value is from terrafrom.tfvars file
  instance_type = "${var.instance_type}"


  # remaining instance configuration
} # closing instance resource block
```

16. How to set up ki jab hum `terraform init` run kare to saare versions update ho jaye, as per requirement.

```
# we are creating terraform configuration file -> terraform.tf
terraform {
  # versioning of our terraform
  required_version = "1.1.0"

  # versioning of our provider
  aws = {
    source = "hashicorp/aws"
    version = "3.71.0" # see the version of provider from .terraform directory
  } # closing provider block
} # closing terraform block
```

17. Can we use variables in terraform configuration files.

No, hard code karna hi hoga

18. How to see our terraform environment dependencies in graphical format

```
# gettign svg file for our dependencies graph
terraform graph | dot -Tsvg > graph.svg

# gettign pdf file for our dependencies graph
terraform graph | dot -Tpdf > graph.pdf
```

19. What is workspace in terrafor

Terraform mei, kisi environment ko manage karne ke liye terraform state ke concept use karta hai. State ek file hoti hai jo infrastructure ki information hold karti hai. Terraform hume allow kar sakta hai ki hum miltiple isolated state kar sakte hai, jisse hum kisi resource or infrasturcture se related multiple isolated environment create kar sakte hai.

20. Describe various operations related to workspace with example

The terraform workspace command is used to manage workspaces.

List

terraform workspace list command is used to list the workspaces. by default, **default** workspace is created.The current workspace is indicated using an asterisk (*) marker.

```
# getting the list of all the present workspace with astrick on current workspace
terraform workspace list
* default
```

Create

The **terraform workspace new** command is used to create a new workspace.

This command will create a new workspace with the given name. A workspace with this name must not already exist#

```
# creating a new workspace
terraform workspace new dev
```

let's create a new workspace using below command

Show

terraform workspace show command will display the current workspace.

```
# showing the workspace jisme currently flow hai
terraform workspace show
test
```

so right now you are in test workspace and if you run terraform apply then tf.state will be create in that workspace.

Switch Workspace

If you want to switch workspace then you can use **terraform workspace select command.**

example:

```
# listing the workspace
terraform workspace list
  default
* test

#switching to default workspace
terraform workspace select default
Switched to workspace "default".

# listing the workapce againg
terraform workspace list
* default
  test
```

Delete Workspace

If you want to delete any existing workspace then you can use terraform delete command.

```
# Deleting the workspace
terraform workspace delete test
# output -> Deleted workspace "test"!
```

Lecture Questions section #3

1. What are modules in Terraform

Humare infrasturcture ko create karne ke liye modules building block hote hai. Ye basically group of resources ko combine karta hai (encapsulates) , jisse specific purpose ko achieve kiya ja sake.

Hence we can say, a Terraform module is a collection of Terraform configuration files in a single directory

Key Benefits of Modules:

- **Reusability:** Modules promote code reuse by encapsulating common infrastructure patterns.
- **Organization:** They help structure complex configurations by breaking them down into smaller, manageable units.
- **Abstraction:** Modules can hide implementation details, making configurations easier to understand and maintain.
- **Modularity:** They allow you to build hierarchical infrastructure by combining multiple modules.

2. Describe root module and child module

Root module

Humare Terraform configuration mei Root moudle top-level directory hoti hai. Basically ye humare infrastructure definition ka starting point hota hai. Root directory se hi hum ye commands chala sakte hai `terraform init`, `terraform plan`, and `terraform apply`

Child module

child moduel ek self-contained directory hoti hai jisme Terraform ki configuration files hoti hai. Ye set of resources ko represent karta hai jo infrastructure kisi component ko denote karta hai.

3. What's problem with variables in modules

Using variables in modules is really very problemetic.

- Humko 2 times variables define karna pardhta hai. Jo variables module ke andar use hote hai, vo to module ke andar define karna hi pardhta hai + same variables outside bhi define karna pardhta hai.
- You can see, we generate 2 variables files, variables.tf (outside file name) variable.tf (inside file name)
- Moduel block (module definiton) ke andar we have to pass the arguments, jo ki module ke variables ko values provide karta hai.
- Moduel block mei humko source mei directory provide karna hota hai, (gaurab sir - directory, web page - tf file, created confusion)
- Jab hum module ke through miltiple resources create karte hai to isse resources ki duplicacy ho sakti hai. There we need to decide which resource needed to be created outside the module and which one should be created inside the module.

  Explaination - Suppose hum module ke concept ke through miltipe ec2 instances create karre hai. We know that EC2 resource ke saath key pair resource and security group resource banta hai. Security Group resource module ke bahar banna chahiye and key pair resource moduel ke andar banna chahiye.

  Twist - Module ke andar banne wale resources ke duplicates ban sakte hai jo ki problemetic hota hai, so we manually need to give distinct names to the resources jo ki module ke andar bante hai.
- **Output block** - Humko output block 2 times create karna pardhta hai. Module ke andar normally output block create karte hai, jisko module ke resources ke attribute tak access hota hai (module ke bahar un resource attribute tak ka access nhi hota). Jab hum module ke bahar output block create karte hai (to get the value of output block jo module ke andar bana hai) then hum usme value ese provide karte hai - `module.<module-name>.<output-block-present-inside-module>`

4. Describe the concept of backend in terraform

Terraform backend ek crucial mechanism hai, jiske through hum apne terraform ke infrastructure as a code ke state ko manage karte hai. Iske many avantages hai jese -

- **Centralized Storage:** Store the state in a remote location like cloud storage (e.g., S3 bucket, Azure Blob Storage) or a dedicated service (e.g., Terraform Cloud, HashiCorp Consul). This allows multiple users to access and modify the state from anywhere.
- **Locking:** Backends can implement locking mechanisms to prevent conflicts when multiple users attempt to modify the infrastructure simultaneously. This ensures data consistency and avoids accidental resource changes.
- **Version Control:** Some backends offer versioning capabilities, allowing you to revert to previous state versions if necessary. This helps with rollbacks and disaster recovery.
- **Collaboration:** Centralized storage and locking facilitate collaboration among team members working on the same infrastructure.

5. Describe some best practices while crating backend with S3 in terraform

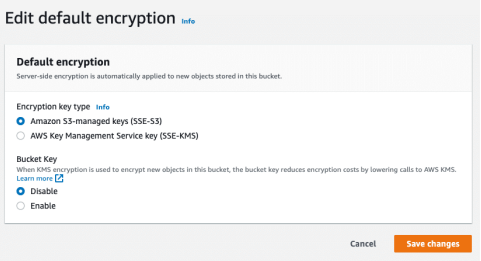
Some of the Terraform S3 backend best practices include:

1. Encryption
2. Access Control
3. Versioning
4. Locking
5. Backend First

Most of these practices are easy to implement as they are readily supported by AWS S3 service.

1. Enable file encryption

Given the sensitive nature of
Terraform state files, it makes sense to encrypt them in storage. AWS S3
buckets offer this functionality by default which makes it easy to
implement encryption at the click of a button.



Organizations
can choose to let AWS S3 manage these keys for their projects. It is
also possible to take a more sophisticated approach where organizations
take control of their keys using AWS KMS service.

2. Implement access control

Needless to say, **public access should be strictly blocked for S3 buckets used for Terraform remote state management**.

Most security threats arise from
human errors, so it is important to control manual access to state files
stored in these S3 buckets. This helps reduce accidental modifications
and unauthorized actions.

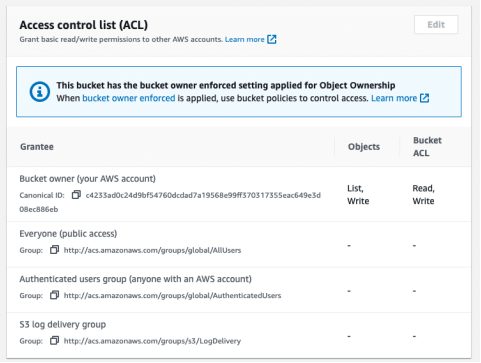
Bucket policies provide a powerful
and flexible way to manage access control for your S3 buckets. To
leverage them, you need to first identify the IAM resources that should
have access to your bucket. After that, you'll need to determine the
necessary permissions you want to grant. Generally, the required
permissions would be actions like listing the bucket contents (
s3:ListBucket), reading objects (*s3:GetObject*), and writing or deleting objects (*s3:PutObject, s3:DeleteObject*).

You will need to write a json policy similar to this:

```
 "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "AWS": "arn:aws:iam::account_id:role/your_role"
       },
       "Action": [
         "s3:ListBucket",
         "s3:GetObject",
         "s3:PutObject",
         "s3:DeleteObject"
       ],
       "Resource": [
         "arn:aws:s3:::s3bucket",
         "arn:aws:s3:::s3bucket/*"
       ]
     }
   ]
}
```

You'll
need to replace the account_id, your_role, and s3bucket with values
from your AWS account. When you have done this, you can easily go to the
permissions tab on your S3 bucket, select bucket policy, and add the above policy there.

It is not the current best practice,
but you can use ACLs in AWS S3 to implement strict access controls. As
shown below, it is possible to grant various levels of access to the
users/entities who are supposed to access the state files for taking
designated actions.



Ideally,
the only entity with write access to the S3 buckets used as Terraform's remote backend should be the user account assigned for Terraform operations. Organizations typically implement the concept of "tech users" or "service accounts" which are different from normal human user accounts.

Define ACLs that allow read and write
access to the tech account responsible to lock and modify the state information and allow read access to selected users for verification purposes.

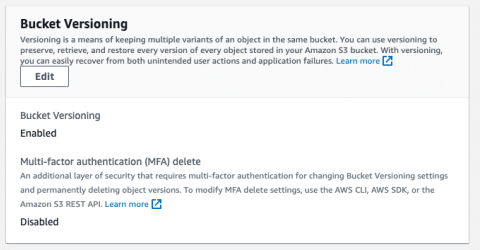
The principle of least privilege should be used in both cases.

3. Enable bucket versioning

AWS S3 bucket versioning allows us to
keep a historical record of all the modifications done to any of the files in the bucket. In the case of a disaster or file corruption, it is easier to recover the state file if previous versions are available.

Given the nature of state files, it
is better to recover from a previous version than to rebuild the state files by importing existing infrastructure manually and individually. With versioning, only the delta needs to be imported, saving a lot of time and cost.

AWS S3 also provides versioning by
default, and its capabilities are summarized in the text provided in the screenshot. As a next step, MFA should also be enabled, so that delete actions are either avoided or happen carefully.



4. Use file-locking

As discussed earlier, to avoid file
corruption due to multiple simultaneous writes, it is important to have a file-locking feature in place.

When you use AWS S3 as a remote
backend, always create a corresponding DynamoDB table as described in the Implementation section. When any developer wants to perform any operations that concern state files (plan, apply, destroy), Terraform first locks the file by updating the LockID column available in the given DynamoDB table.

This avoids race conditions. Thus DynamoDB is a very integral part of Terraform's S3 backends.

5. Follow the backend-first rule

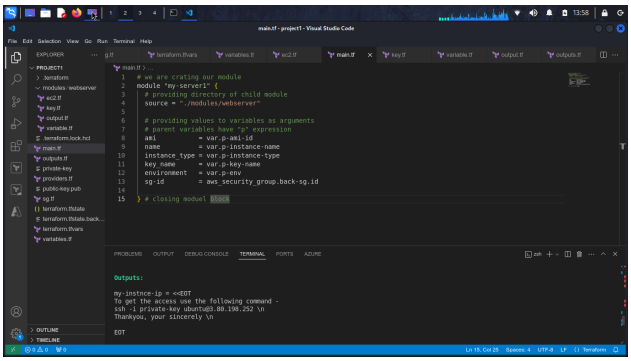
When initiating any Terraform project – especially when multiple developers are working on the same repo – **a remote backend should be configured first**. This is a general best practice and not specific to AWS S3 remote backend.

Configuring a remote backend is a
one-time activity, and once done, we need not worry about the state file maintenance during the development process.

6. How to give S3 backend to Terraform

Step 1:

Create the terraform structure, implement the concept of modules, sg, key pair, ec2, variables, output



Step 2:

Install AWS CLI and provide appropriate permissions (I created an IAM user and provided it all admin permissions, same user I am already using for terraform). Login to AWS CLI - aws configure and provide it access key and secret key along with default region

Step 3:

Create an S3 bucket, in my case I had created a bucket with name - tf-bucket-03

Step 3:

create backend block in terraform configuration.

file - main.tf

```
# our terraform configuration
terraform {
  backend "s3" {
    # providing the bucket name
    bucket = "tf-bucket-03"

    # giving name to state file
    key = "terraform.tfstate"

    # provide the region
    region = "us-east-1"
  }
}
```

Step 4:

Delete the state files and plugin directories if any exist. Then initialise the terraform `terraform init`

Step 5:

Execute the following commands and check the state file in your AWS Console

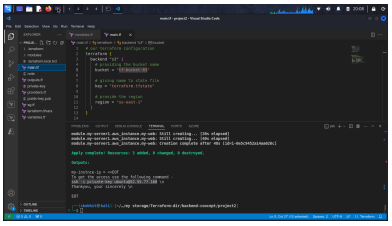
```
terraform validate
terraform fmt
terraform plan
terraform apply --auto-approve
terraform destroy --auto-approve
```

7. Implement the migration of backend in Terraform

It is very simple concept. When we have a remote backend then it is very twisting to make it local. Directle Backend Block ko hata dene se humara Backend local nhi aata. Follow the following steps.

Step 1

Understand the initial scenario. Humare Terraform infrasturecture hai, jo ki romote backend par set hai. At our Terraform infracture , we have modules, variables, outputs, EC2, SG, Key pair. There is alread an S3 bucket present, in which we have terraform.tfstate file.



Step 2

Comment or Delete the backend S3 block. I had commented whole terraform configure block as there was nothing apart form backend S3 over there.

Step 4

Now apply the changes using `terraform apply --auto-approve` . You will get an error

```
Error: initialization required, please run "terraform init"

# the run the following comnand
terraform init --migrate-state
```

You will observe that terraform.tfstate file would be created at local. Now it will work to store the state of your terraform infrastructure state. If you wish, make changes to infrastructure and apply the changes.

8. Implement the state locking using S3 and DynamoDB in Terraform

Step 1

Consider the intial scenario, we have a Terraform infrasturecture of Modules, EC2, key pair, SG. Our backend is set to local.

Step 2

Create a table in dynamoDB. Give name to table - tf-statelock-table. Give partition key - LockID. Simultaniously create an S3 bucket for setting remote S3 backend

Step 3

Give terraform configuration as following to meet the expected scenario

```
# our terraform configuration
terraform {
  backend "s3" {
    # providing the bucket name we have to create the S3 bucket manually
    bucket = "tf-bucket-03"

    # giving name to state file
    key = "terraform.tfstate"

    # provide the region
    region = "us-east-1"

    # setting state locking, we had to create the DynamoDB Table manually
    dynamodb_table = "tf-statelock-table"
  }
}
```

Step 4

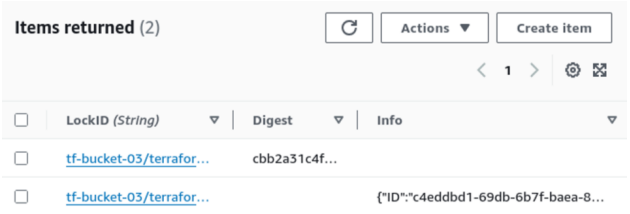
Use the following commands

```
# we have to set backend from local to romote S3
terraform init --migrate-state

# Now implement the locking mechanism by applying changes
terraform apply --auto-approve
```

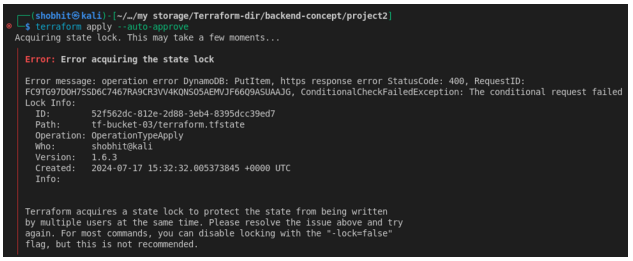
Step 5

Now do some changes in your resource configuration (I had change the name of instance) . Apply the changes `terraform apply --auto-approve` . Observe our dynamoDB Table - Explore Table Items > Items Returned. You will obseve that a new temporary file will be created. That will allow only one change to occour and avoid other change.

Below you can see that a file is created that is devoted to the last change requet. Now no other change request will be accpetd untill the current change request get finish.

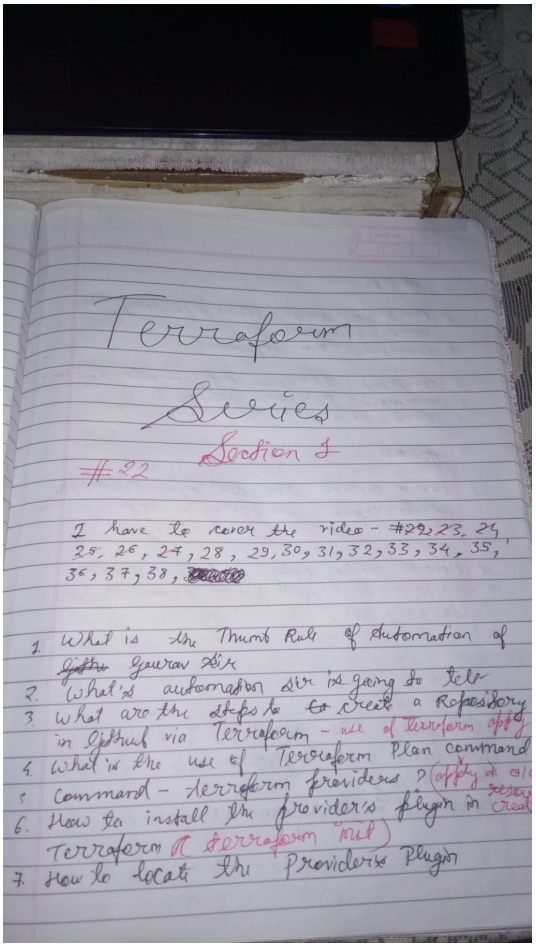


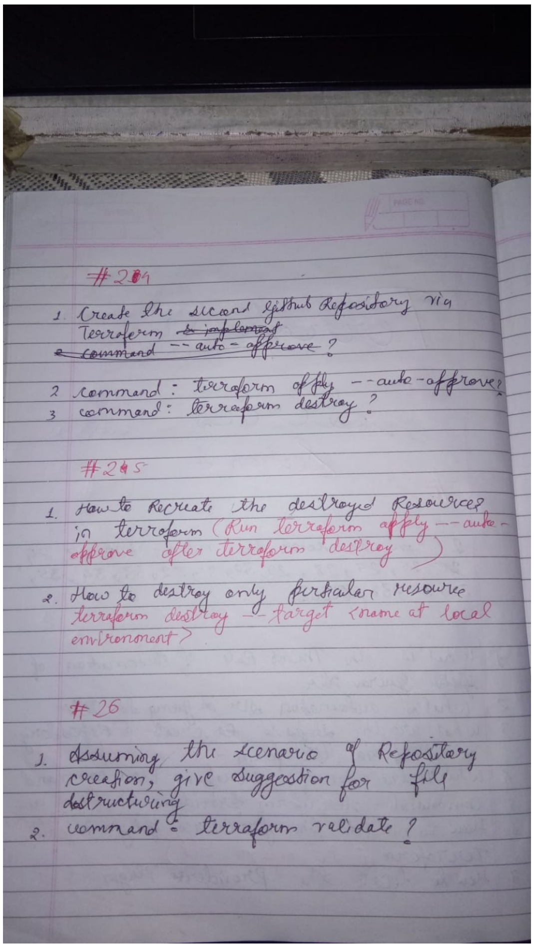
Below you can see that already we had applied the changes. We are still trying to apply the changes simultaniously, but locking mechanism is preventing the chages to occour. It thrown the error - Error acquiring the state lock



Hand wirtten notes

Section # 1





#24

1. Create the second github Repository via Terraform ~~& implement~~
~~command~~ -- auto-approve ?

2 command : terraform apply --auto-approve?
3 command : terraform destroy ?

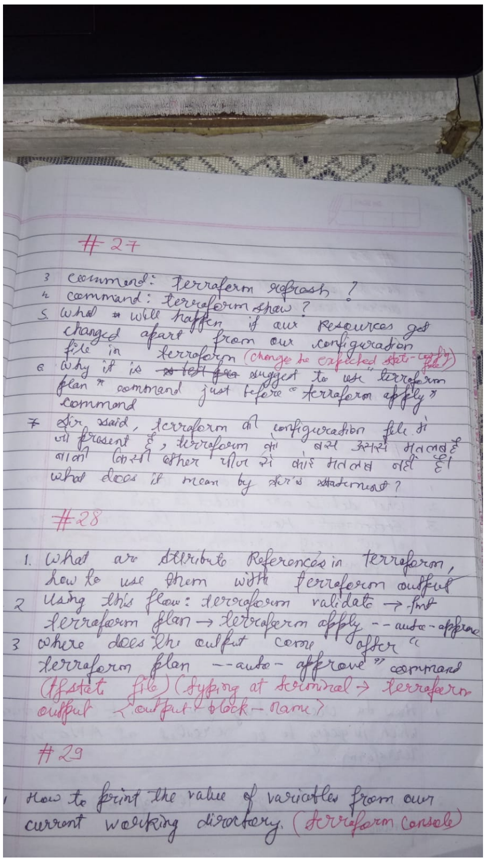
#25

1. How to Recreate the destroyed Resources in terraform (Run terraform apply --auto-approve after terraform destroy)

2. How to destroy only particular resource
terraform destroy --target <name at local environment>

#26

1. Assuming the scenario of Repository creation, give suggestion for file destructuring
2. command : terraform validate ?



#27

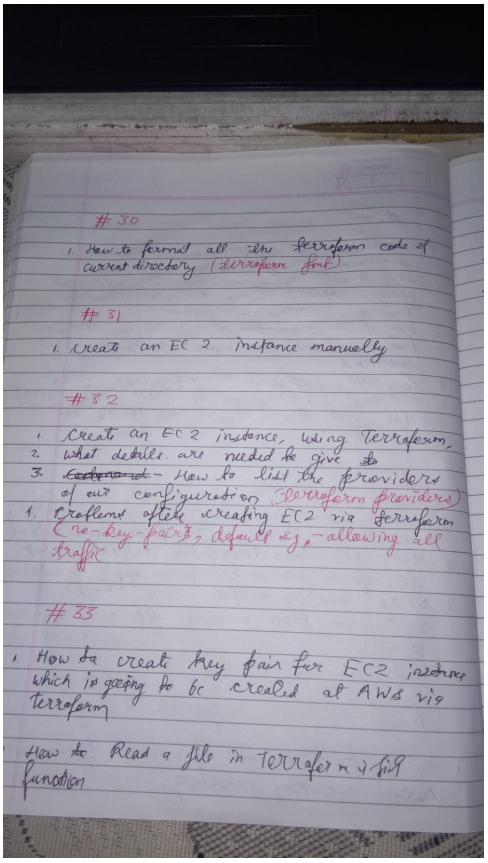
3 command: terraform refresh ?
4 command: terraform show ?
5. What will happen if our Resources get changed apart from our configuration file in terraform (change to expected state-config file)
6 Why it is ~~so best for~~ suggest to use "terraform plan" command just before "terraform apply" command
7 Sir said, terraform की configuration file में जो present है, terraform को बस उससे मतलब है बाकी किसी other चीज से कोई मतलब नहीं है।
What does it mean by sir's statement?

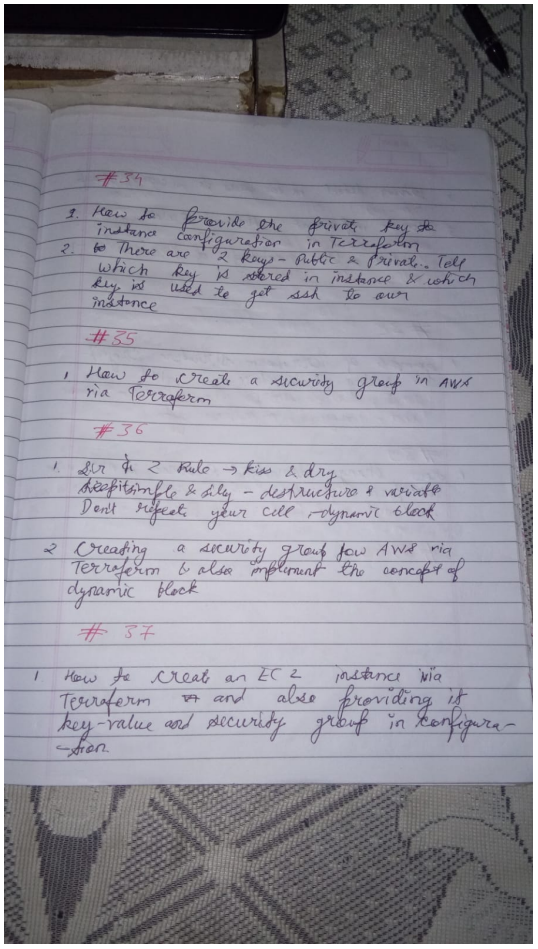
#28

1. What are attribute References in terraform, how to use them with terraform output
2 Using this flow: terraform validate → first terraform plan → terraform apply --auto-approve
3 Where does the output come after "terraform plan --auto-approve" command
(tfstate file) (typing at terminal → terraform output <output-block-name>)

#29

1 How to print the value of variables from our current working directory. (terraform console)


#30
1. How to format all the terraform code of current directory (terraform fmt)
#31
1. Create an EC2 instance manually
#32
1. Create an EC2 instance, using Terraform,
2. what details are needed to give to
3. ~~Command~~ - How to list the providers of our configuration (terraform providers)
1. Problems after creating EC2 via Terraform (no-key-pairs, default sg, - allowing all traffic
#33
1. How to create key pair for EC2 instance which is going to be created at AWS via Terraform
How to Read a file in Terraform ? file function



#34

1. How to provide the private key to instance configuration in Terraform
2. There are 2 keys - public & private. Tell which key is stored in instance & which key is used to get ssh to our instance

#35

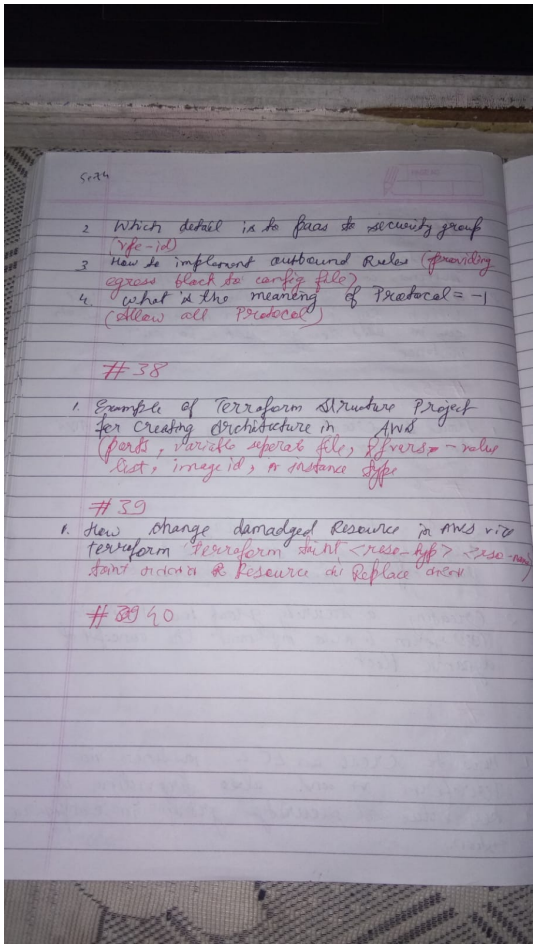
1. How to create a security group in AWS via Terraform

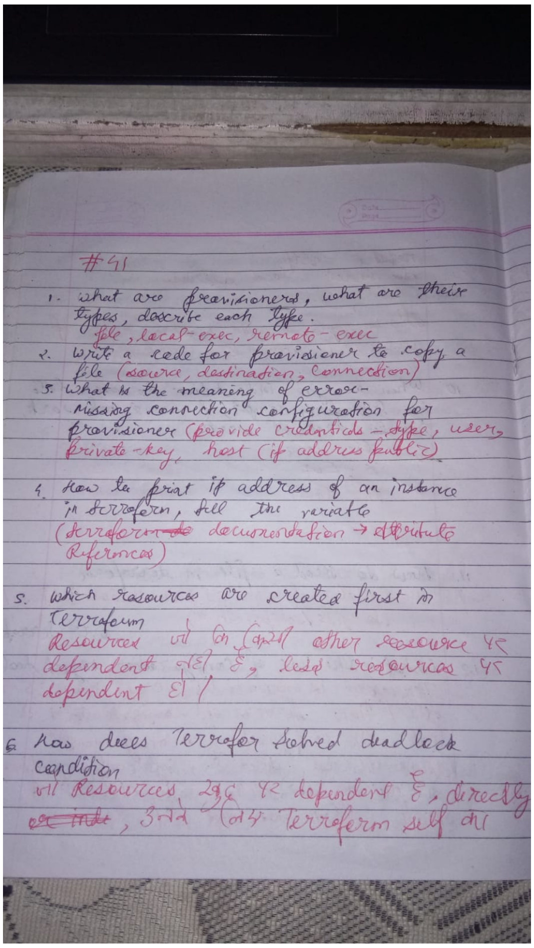
#36

1. Use of 2 rule → kiss & dry
   ~~Lifecycle~~ & life - destructure & variable
   Don't repeat your self - dynamic block
2. Creating a security group for AWS via Terraform & also implement the concept of dynamic block

#37

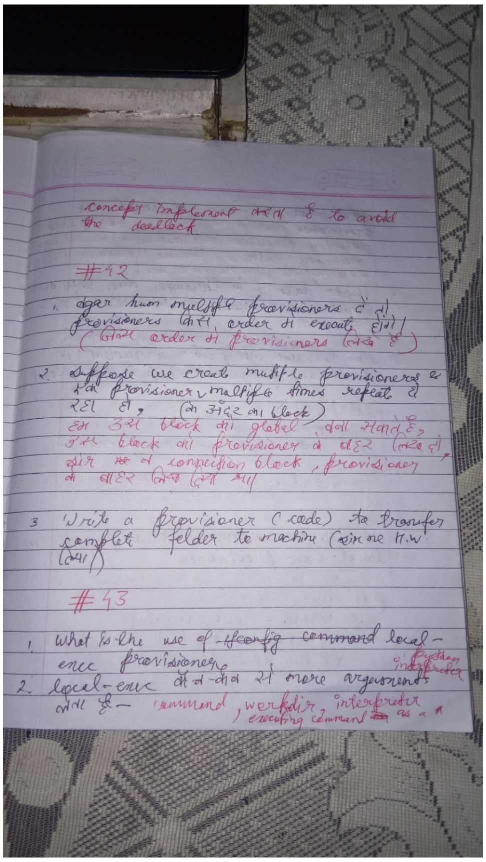
1. How to create an EC2 instance via Terraform ~~&~~ and also providing it key-value and security group in configuration.


2 Which detail is to pass to security group (vpc-id)
3 How to implement outbound Rules (providing egress block to config file)
4 What is the meaning of Protocol = -1 (allow all Protocol)
#38
1. Example of Terraform structure Project for Creating Architecture in AWS (ports, variable seperate file, ${var.} - value list, image id, instance type
#39
1. How change damaged Resource in AWS via terraform (terraform taint <rese_type> <rese_name> taint direction & Resource to Replace create
#39 40



#41

1. what are provisioners, what are their types, describe each type.
   file, local-exec, remote-exec
2. write a code for provisioner to copy a file (source, destination, connection)
3. What is the meaning of error- missing connection configuration for provisioner (provide credenticals - type, user, private-key, host (if address public))
4. How to print ip address of an instance in terraform, tell the variable
   (terraform documentation → attribute References)
5. which resources are created first in Terraform
   Resources जो कि किसी other resource पर dependent नहीं है, बाद में resources पर dependent हैं।
6. how does Terrafor solved deadlock condition
   जो Resources 2nd पर dependent है, directly, उनके लिए Terraform self का



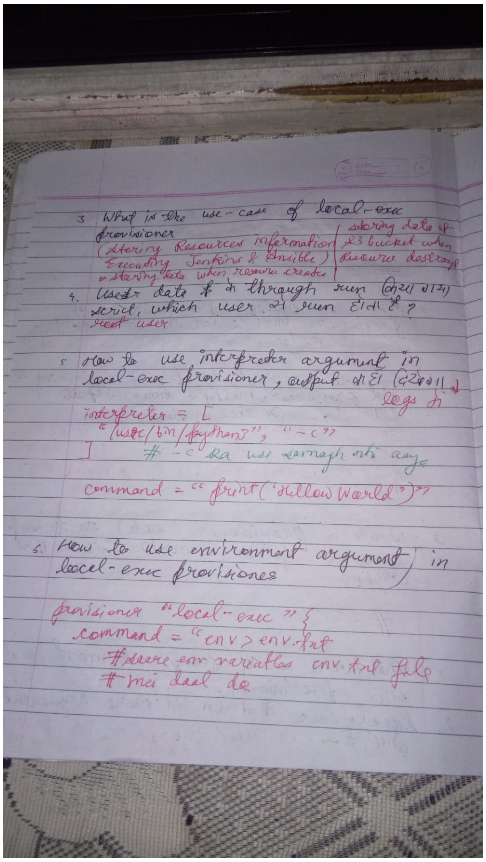
concept implement करता है to avoid the deadlock

#42

1. agar hum multiple provisioners दें तो provisioners किस order में execute होंगे।
   (जिस order में provisioners लिखे हैं)
2. Suppose we create multiple provisioners & एक provisioner, multiple times repeat हो रहा हो, (के अंदर का block)
   हम उस block को global बना सकते हैं, उस block को provisioner के बाहर लिख दो, sir ने connection block, provisioner के बाहर लिख दिया था।
3. Write a provisioner (code) to transfer complete folder to machine (sir ne H.W दिया)

#43

1. What is the use of local-exec provisioner
2. local-exec कौन-कौन से more arguments लेता है - command, workdir, interpreter (python interpreter), executing command as a



3 What is the use-case of local-exec provisioner

(Storing Resources information | Executing Jenkins & Ansible) | storing data of S3 bucket when resource destroys

+ storing data when resource create

4. User data के through run किया जाता script, which user से run होता है?

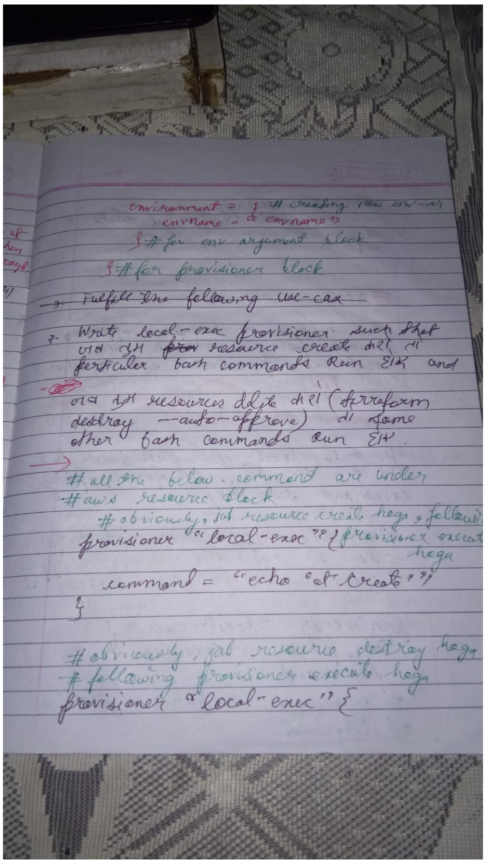
root user

5 How to use interpreter argument in local-exec provisioner, output कहा दिखता!! ↓ logs में

```
interpreter = [
  "/usr/bin/python3", "-c"
]        # -c ka use samajh nhi aaya

command = "print('Hellow World')"
```

6. How to use environment argument in local-exec provisioner

```
provisioner "local-exec" {
  command = "env > env.txt
    # saare env variables env.txt file
    # mei daal do
```



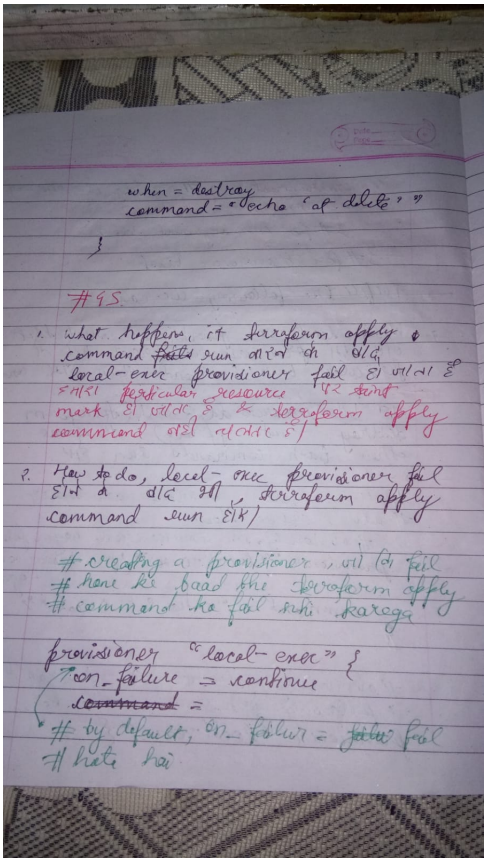
```
    environment = { # creating new env-var
        envname = "envname"
    } # for env argument block

  } # for provisioner block
```

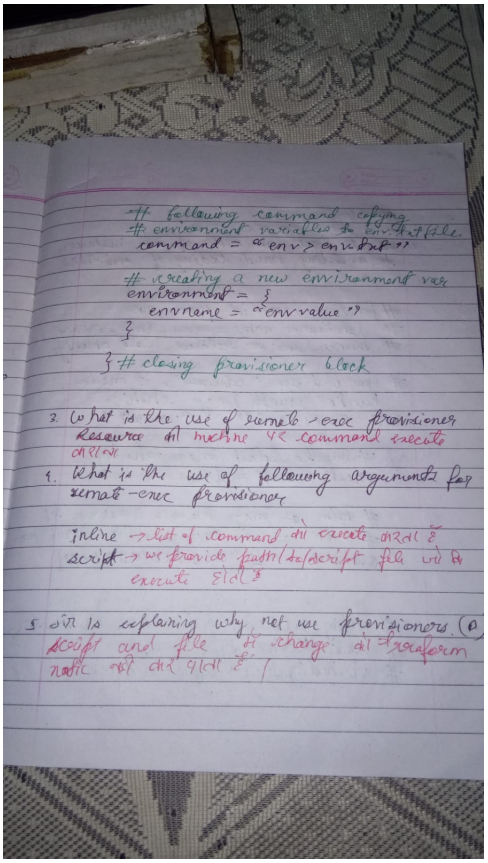
~~7. Fulfill the following use-case~~

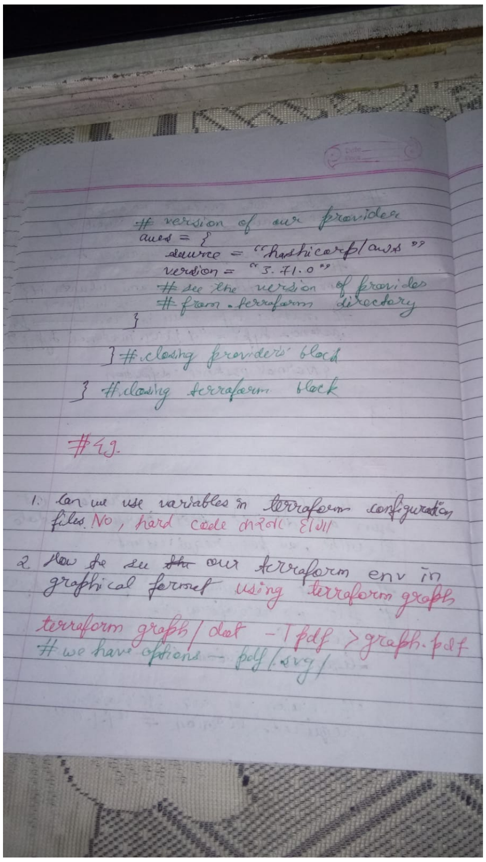
7. Write local-exec provisioner such that जब हम resource create करें तो particular bash commands Run हों and जब हम resources delete करें (terraform destroy --auto-approve) तो some other bash commands Run हों.

→

```
# all the below command are under
# aws resource block
   # obviously, jab resource create hoga, following provisioner execute hoga
provisioner "local-exec" {
   command = "echo 'at create'"
}

# obviously, jab resource destroy hoga
# following provisioner execute hoga
provisioner "local-exec" {
```


when = destroy
command = "echo 'ap dikte'"
}
#45
1. what happens, if terraform apply command fails
2. How to do, local-exec provisioner fail
# creating a provisioner, will fail
# here is how the terraform apply
# command to fail nhi karega
provisioner "local-exec" {
on_failure = continue
command =
by default, on_failure = fail
hota hai


# following command copying
# environment variables to the text file.
command = "env > envtext"
# creating a new environment var
environment = {
envname = "envvalue"
}
} # closing provisioner block
3. What is the use of remote-exec provisioner
4. What is the use of following arguments for remote-exec provisioner
inline → list of commands
script → we provide path to script file
5 sir is explaining why not use provisioners



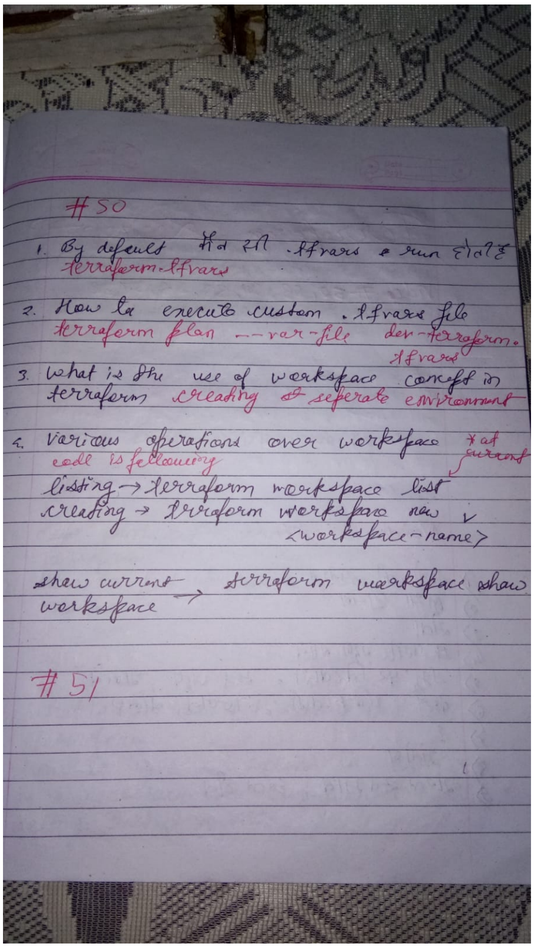
```
# version of our provider
aws = {
    source = "hashicorp/aws"
    version = "3.71.0"
    # see the version of provider
    # from .terraform directory
}

} # closing provider's block

} # closing terraform block
```

#49.

1. Can we use variables in terraform configuration files. No, hard code करना होगा।

2. How to see ~~the~~ our terraform env in graphical format using terraform graph

terraform graph | dot -Tpdf > graph.pdf
we have options — pdf / svg /



#50

1. By default कौन सी .tfvars run होती है
terraform.tfvars

2. How to execute custom .tfvars file
terraform plan --var-file dev-terraform.tfvars

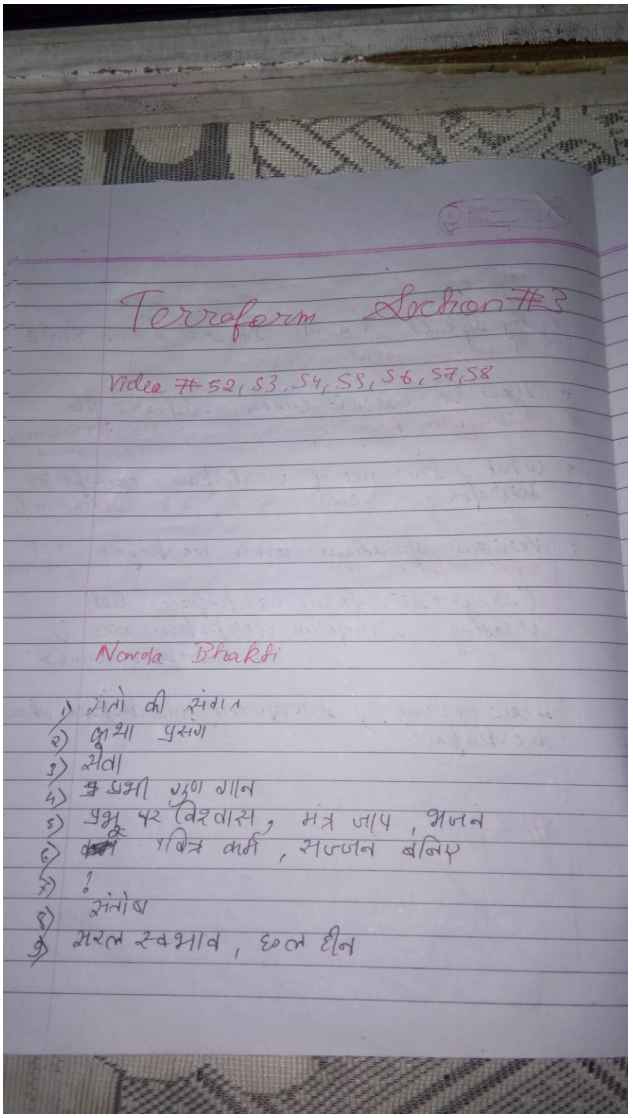
3. What is the use of workspace concept in terraform creating ~~a~~ seperate environment

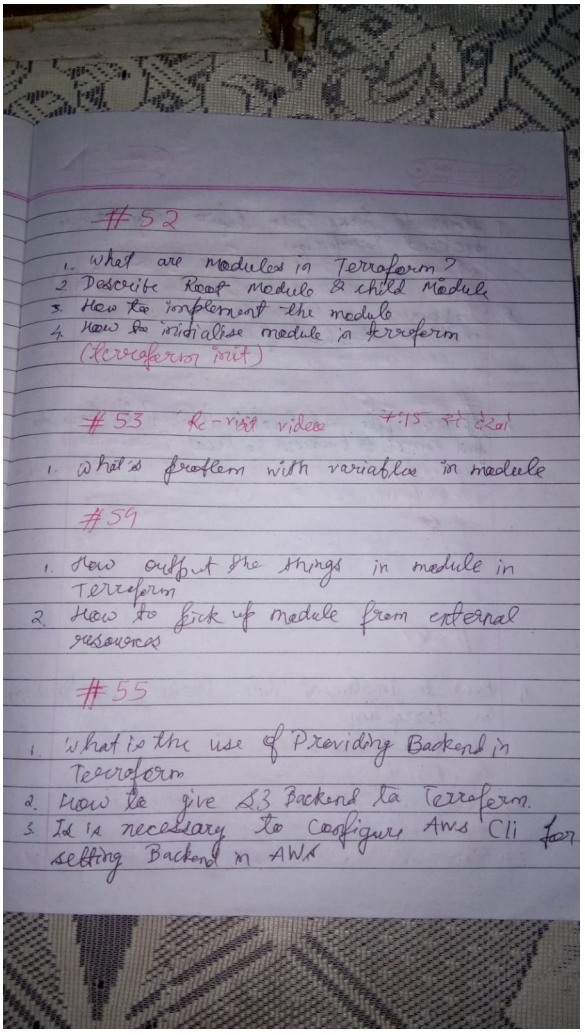
4. Various operations over workspace * at current
code is following

listing → terraform workspace list
creating → terraform workspace new <workspace-name>

show current workspace → terraform workspace show

#51


Terraform Section #3
Video #52, 53, 54, 55, 56, 57, 58
Navda Bhakti
1) संतो की संगत
2) कथा प्रसंग
3) सेवा
4) प्रभु गुण गान
5) प्रभु पर विश्वास, मंत्र जाप, भजन
6) पवित्र कर्म, सज्जन बनिए
7) !
8) संतोष
9) सरल स्वभाव, छल हीन



#52

1. What are modules in Terraform?
2. Describe Root module & child module
3. How to implement the module
4. How to initialise module in terraform (Terraform init)

#53 Re-visit video 7:15 ft 22nd

1. What's problem with variables in module

#54

1. How output the things in module in Terraform
2. How to pick up module from external resources

#55

1. What is the use of Providing Backend in Terraform
2. How to give S3 Backend to Terraform.
3. Is it necessary to configure Aws Cli for setting Backend in AWS

4 What to take care before using S3 as backend Terraform
(Provide all S3 Permissions to user
AWS cli should be already installed)

5. Implement multiple backend with multiple Workspace

S6

1. tfstate file S3 bucket में रखी है, how to put remote backend to local (Comment backend, use → terraform init -migrate-state
, terraform destroy

#

S87

1, How to implement the locking mechanism in terraform
↳ Create a Dynamo DB Table
partition key → LockID
↳ adding an attribute to backend block
dynamodb_table = <your-table-name>
↳ Run terraform apply --auto-approve

↳ see table, wait, refresh, one entry will be automatically deleted

↳ Run terraform destroy --auto-approve
you will see one more entry, it will be deleted after some time

↳ Run terraform apply --auto-approve before disappearance of above entry in dynamo DB table

You will see, an error will be occurred - Error acquiring the stat loc

S8