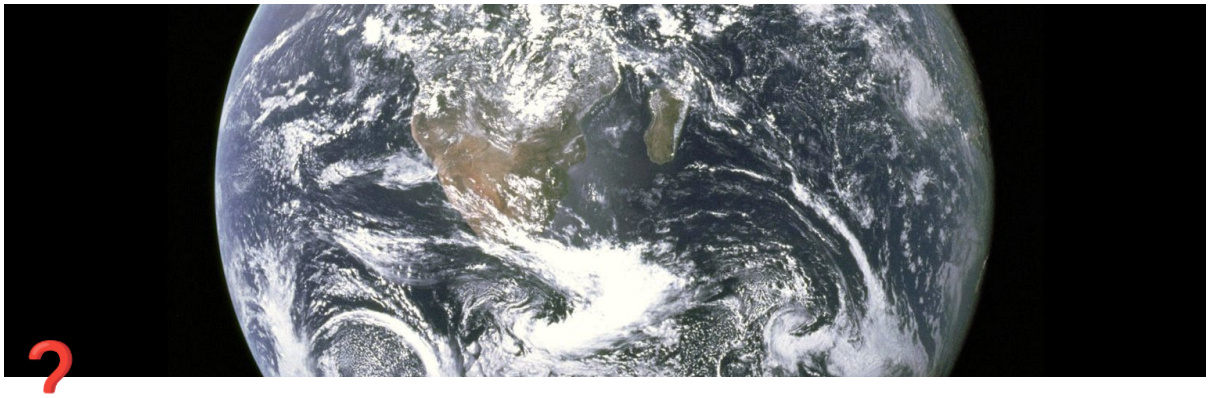

Questions & UseCases

▼ **References**

- https://medium.com/bb-tutorials-and-thoughts/250-practice-questions-for-terraform-associate-certification-7a3ccebe6a1a

▼ You run a local-exec provisioner in a null resource called null_resource.run_script and realize that you need to rerun the script.

- You can use the `terraform taint` command to mark a resource as tainted, which indicates that it needs to be recreated during the next `terraform apply` run. Since the null resource is not directly tied to any infrastructure resources, you can use this command to trigger the `local-exec` provisioner to run again.

```
terraform taint null_resource.run_script
```

▼ Which provisioner invokes a process on the resource created by Terraform?

- The remote-exec provisioner invokes a script on a remote resource after it is created.

▼ Which of the following is not true of Terraform providers?

- A. Providers can be written by individuals
  B. Providers can be maintained by a community of users
  C. Some providers are maintained by HashiCorp
  D. Major cloud vendors and non-cloud vendors can write, maintain, or collaborate on Terraform providers
  E. None of the above
- Ans: E

▼ You have deployed a new webapp with a public IP address on a cloud provider. However, you did not create any outputs for your code. What is the best method to quickly find the IP address of the resource you deployed?

- A. Run terraform output ip_address to view the result
  B. In a new folder, use the terraform_remote_state data source to load in the state file, then write an output for each resource that you find the state file
  C. Run terraform state list to find the name of the resource, then terraform state show to find the attributes including public IP address Most Voted
  D. Run terraform destroy then terraform apply and look for the IP address in stdout
- Ans: C. Run terraform state list to find the name of the resource, then terraform state show to find the attributes including public IP address

▼ Which of the following is not a key principle of infrastructure as code?

- A. Versioned infrastructure
  B. Golden images Most Voted
  C. Idempotence
  D. Self-describing infrastructure
- Ans: B Golden Images

▼ Terraform variables and outputs that set the "description" argument will store that description in the state file?

- True.

▼ If you manually destroy infrastructure, what is the best practice reflecting this change in Terraform?

- `terraform plan -refresh-only`

▼ What is not processed when running a terraform refresh?

- A. State file
  B. Configuration file
  C. Credentials
  D. Cloud provider
- Credentials

▼ What information does the public Terraform Module Registry automatically expose about published modules?

- D. All of the above. The Terraform Module Registry automatically exposes information about published modules, including required input variables, optional input variables and default values, and outputs.

▼ If a module uses a local values, you can expose that value with a terraform output.

- `False.` Outputs are used to expose values that are generated or obtained by the module and can be used by other resources outside the module. Local values are only used within the module and cannot be exposed as outputs.

▼ You should store secret data in the same version control repository as your Terraform configuration.

- `False.`

▼ You have provisioned some virtual machines (VMs) on Google Cloud Platform (GCP) using the gcloud command line tool. However, you are standardizing with
Terraform and want to manage these VMs using Terraform instead.
What are the two things you must do to achieve this? (Choose two.)

- A. Provision new VMs using Terraform with the same VM names
  B. Use the terraform import command for the existing VMs
  C. Write Terraform configuration for the existing VMs
  D. Run the terraform import-gcp command
- B & C.

▼ You have recently started a new job at a retailer as an engineer. As part of this new role, you have been tasked with evaluating multiple outages that occurred during peak shopping time during the holiday season. Your investigation found that the team is manually deploying new compute instances and configuring each compute instance manually. This has led to inconsistent configuration between each compute instance.

- A. Implement a ticketing workflow that makes engineers submit a ticket before manually provisioning and configuring a resource
  B. Implement a checklist that engineers can follow when configuring compute instances
  C. Replace the compute instance type with a larger version to reduce the number of required deployments
  D. Implement a provisioning pipeline that deploys infrastructure configurations committed to your version control system following code reviews.
- Ans: D

▼ Which of the following is available only in Terraform Enterprise or Cloud workspaces and not in Terraform CLI?

- Ans: Secure variable storage

▼ You have used Terraform to create an ephemeral development environment in the cloud and are now ready to destroy all the infrastructure described by your
Terraform configuration. To be safe, you would like to first see all the infrastructure that will be deleted by Terraform. Which command should you use to show all of the resources that will be deleted? (Choose two.)

- A. Run terraform plan -destroy.
  B. This is not possible. You can only show resources that will be created.
  C. Run terraform state rm *.
  D. Run terraform destroy and it will first output all the resources that will be deleted before prompting for approval.
- Ans: A & D

▼ Terraform can run on Windows or Linux, but it requires a Server version of the Windows operating system.

- `False.` Terraform can run on both Windows and Linux, and it does not require a Server version of the Windows operating system. It can run on Windows 10, Windows 8, and Windows 7, as well as various Linux distributions.

▼ Which task does terraform init not perform?

- A. Sources all providers present in the configuration and ensures they are downloaded and available locally
  B. Connects to the backend
  C. Sources any modules and copies the configuration locally
  D. Validates all required variables are present
- Ans: D

▼ You have declared a variable called var.list which is a list of objects that all have an attribute id.Which options will produce a list of the IDs? (Choose two.)

- B. var.list[*].id
- C. [var.list[*].id]
- D. [for o in var.list : o.id]
- A. { for o in var.list : o => o.id }

- Ans: B & D

▼ Which argument(s) is (are) required when declaring a Terraform variable?

A. type
B. default
C. description
D. All of the above
E. None of the above

- Ans: E

▼ What features does the hosted service Terraform Cloud provide? (Choose two.)

- A. Automated infrastructure deployment visualization
  B. Automatic backups
  C. Remote state storage
  D. A web-based user interface (UI)

- Ans: C & D

▼ Which option can not be used to keep secrets out of Terraform configuration files?

- `Option D, "secure string,"` is not a valid method for keeping secrets out of Terraform configuration files. There is no "secure string" functionality in Terraform. Instead, secrets can be stored in environment variables or external secret storage systems, and accessed in Terraform through interpolations or data sources. Some providers also support specific mechanisms for handling secrets, such as the "secrets" block in the AWS provider.

▼ You have never used Terraform before and would like to test it out using a shared team account for a cloud provider. The shared team account already contains
15 virtual machines (VM). You develop a Terraform configuration containing one VM, perform terraform apply, and see that your VM was created successfully.
What should you do to delete the newly-created VM with Terraform?

- A. The Terraform state file contains all 16 VMs in the team account. Execute terraform destroy and select the newly-created VM.
  B. The Terraform state file only contains the one new VM. Execute terraform destroy.
  C. Delete the Terraform state file and execute Terraform apply.
  D. Delete the VM using the cloud provider console and terraform apply to apply the changes to the Terraform state file.

- `Option B is correct`. Since the Terraform state file only contains the one new VM that was created by Terraform, executing `terraform destroy` will delete the new VM. It will not delete the 15 VMs that were pre-existing in the shared team account.

▼ Setting the TF_LOG environment variable to DEBUG causes debug messages to be logged into syslog.

- `False.` Setting the TF_LOG environment variable to DEBUG causes debug messages to be logged to standard error (stderr).

▼ In `Terraform 0.13 and above`, outside of the `required_providers block`, Terraform configurations always refer to providers by their local names.

- `True.` Outside of the required_providers block, Terraform configurations always refer to providers by their local names.

▼ Terraform providers are always installed from the Internet?

- `False.` Terraform providers can be installed from various sources, including the official Terraform Registry, private registries, or custom-built providers.

▼ When does terraform apply reflect changes in the cloud environment?

- Immediately.

▼ A Terraform provider is not responsible for?

- A. Understanding API interactions with some service
  B. Provisioning infrastructure in multiple clouds
  C. Exposing resources and data sources based on an API
  D. Managing actions to take based on resource differences
- Ans: B. Provisioning infrastructure in multiple clouds
- Option B is incorrect because Terraform providers can indeed be responsible for provisioning infrastructure in multiple clouds. For example, the AWS provider can be used to manage resources in AWS, while the Google Cloud provider can be used to manage resources in Google Cloud Platform. Providers are not limited to a single cloud provider or service.

▼ Terraform provisioners can be added to any resource block. true or false & why?

- True.
- Terraform provisioners can be added to any resource block in the Terraform configuration file. Provisioners are used to execute scripts or commands on a resource after it has been created or updated by Terraform. Since provisioners are resource-specific, they can be added to any resource block to perform any custom actions required on that resource.

▼ What is terraform refresh intended to detect?

- A. Terraform configuration code changes
  B. Empty state files
  C. State file drift
  D. Corrupt state files
- Terraform refresh is intended to detect state file drift. This occurs when the current state of the infrastructure, as described by the cloud provider's API, differs from the state stored in the Terraform state file. Refreshing the state allows Terraform to update the state file to match the current infrastructure.

▼ A Terraform local value can reference other Terraform local values.

- `True.` We can reference local values to other local values.

▼ You're building a CI/CD (continuous integration/ continuous delivery) pipeline and need to inject sensitive variables into your Terraform run.
How can you do this safely?

- A. Pass variables to Terraform with a λ€"var flag
  B. Copy the sensitive variables into your Terraform code
  C. Store the sensitive variables in a secure_vars.tf file
  D. Store the sensitive variables as plain text in a source code repository
- A. Pass variables to Terraform with a -var flag is a safe way to inject sensitive variables into a Terraform run.
- This allows you to pass in sensitive information at runtime rather than storing it in plain text in your Terraform code or in a source code repository. It is important to ensure that the -var flag is not logged or exposed in any way that could compromise the sensitive information. Additionally, you could consider using a secrets management system such as HashiCorp Vault or AWS Secrets Manager to securely store and retrieve sensitive variables at runtime.

▼ Your security team scanned some Terraform workspaces and found secrets stored in plaintext in state files.
How can you protect sensitive data stored in Terraform state files?

- A. Delete the state file every time you run Terraform
  B. Store the state in an encrypted backend
  C. Edit your state file to scrub out the sensitive data
  D. Always store your secrets in a secrets.tfvars file.
- `B. Store the state in an encrypted backend.` This can be achieved by configuring Terraform to use a remote backend that supports encryption, such as Terraform Cloud, Amazon S3 with server-side encryption, or

HashiCorp Consul. Storing the state file in an encrypted backend helps to protect sensitive data from unauthorized access.

▼ You have a simple Terraform configuration containing one virtual machine (VM) in a cloud provider. You run terraform apply and the VM is created successfully. What will happen if you delete the VM using the cloud provider console, and run terraform apply again without changing any Terraform code?

- Terraform will recreate the VM

▼ Which of these options is the most secure place to store secrets foe connecting to a Terraform remote backend?

- A. Defined in Environment variables
  B. Inside the backend block within the Terraform configuration
  C. Defined in a connection configuration outside of Terraform
  D. None of above
- `C. Defined in a connection configuration outside of Terraform is the most secure place to store secrets for connecting to a Terraform remote backend.` It is recommended to use a separate configuration file or a tool like a key management service to store sensitive information such as passwords, access keys, and API tokens outside of Terraform configuration files. This approach reduces the risk of accidental exposure and helps to maintain the security and integrity of the system.
- To store secrets for connecting to a Terraform remote backend in a secure way, you can use an external secrets management system such as HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault. These systems allow you to securely store and manage secrets and access them programmatically from within your Terraform code or from other applications.
- Another option is to use environment variables or a configuration file outside of Terraform to store the secrets and pass them into your Terraform code via input variables or interpolation. However, this approach requires careful management of the secrets to ensure they are not accidentally exposed or leaked.

▼ Your DevOps team is currently using the local backend for your Terraform configuration. You would like to move to a remote backend to begin storing the state file in a central location.
Which of the following backends would not work?

- A. Amazon S3
  B. Artifactory
  C. Git
  D. Terraform Cloud
- `C. Git would not work as a backend for Terraform state file storage.` While `Git can be used for version control and collaboration`, it is not designed to handle concurrent access to a single state file or provide locking and consistency guarantees required by Terraform's remote state management.
- Instead, options like Amazon S3, Artifactory, and Terraform Cloud are designed for this purpose and provide locking and consistency guarantees to ensure that the state file remains consistent and usable by multiple users.

▼ Which backend does the Terraform CLI use by default? without providing backend block in terraform settings.

- local backend
- If no backend block is specified in the Terraform configuration, the Terraform CLI will use the local backend by default.

▼ When you initialize Terraform, where does it cache modules from the public Terraform Module Registry?

- `.terraform` → On disk in the .terraform sub-directory
- When you initialize Terraform, it caches modules from the public Terraform Module Registry in the `.terraform` directory in the root of your configuration.

▼ You write a new Terraform configuration and immediately run terraform apply in the CLI using the local backend. Why will the apply fail?

- Terraform needs to install the necessary plugins first

▼ What features stops multiple admins from changing the Terraform state at the same time?

- A. Version control
- B. Backend types
- C. Provider constraints
- D. `State locking`
- Ans: D

▼ A fellow developer on your team is asking for some help in refactoring their Terraform code. As part of their application's architecture, they are going to tear down an existing deployment managed by Terraform and deploy new. However, there is a server resource named aws_instance.ubuntu[1] they would like to keep to perform some additional analysis.
What command should be used to tell Terraform to no longer manage the resource?

- A. terraform apply rm aws_instance.ubuntu[1]
- B. terraform state rm aws_instance.ubuntu[1]
- C. `terraform plan rm aws_instance.ubuntu[1]`
- D. terraform delete aws_instance.ubuntu[1]
- Ans: C
- To tell Terraform to no longer manage a resource, the `terraform state rm` command should be used.
- In this specific case, the command `terraform state rm aws_instance.ubuntu[1]` can be used to remove the `aws_instance` resource named `ubuntu` with index `1` from the Terraform state file. This will prevent Terraform from managing that specific resource during subsequent runs of `terraform apply` or `terraform plan`.

▼ A terraform apply can not _____ infrastructure.

- import.

▼ You just scaled your VM infrastructure and realized you set the count variable to the wrong value. You correct the value and save your change. What do you do next to make your infrastructure match your configuration?

- A. Run an apply and confirm the planned changes
- B. Inspect your Terraform state because you want to change it
- C. Reinitialize because your configuration has changed
- D. Inspect all Terraform outputs to make sure they are correct
- Ans: `C. Reinitialize because your configuration has changed`

▼ why not option A?

- Option A "Run an apply and confirm the planned changes" is also a possible solution to apply the changes. However, it is not the most efficient solution because if you have many resources, Terraform will destroy and recreate all the resources, which can take a long time to apply. Additionally, it is generally not a good practice to apply changes to infrastructure without first reviewing the planned changes. Running `terraform plan` would give you the opportunity to review the changes before applying them, reducing the chance of introducing errors in your infrastructure.

▼ Terraform validate reports syntax check errors from which of the following scenarios?

- A. Code contains tabs indentation instead of spaces
- B. There is missing value for a variable
- C. The state files does not match the current infrastructure
- D. None of the above
- Ans: B

▼ why not option A?

- The `terraform validate` command is used to validate the syntax of the terraform files. Terraform performs a syntax check on all the terraform files in the directory, and will display an error if any of the files doesn't validate.
- This command **does not** check formatting (e.g. tabs vs spaces, newlines, comments etc.).
- ▼ The following can be reported
  - invalid HCL syntax (e.g. missing trailing quote or equal sign)
  - invalid HCL references (e.g. variable name or attribute which doesn't exist)
  - same `provider` declared multiple times
  - same `module` declared multiple times
  - same `resource` declared multiple times
  - invalid `module` name
  - interpolation used in places where it's unsupported (e.g. `variable`, `depends_on`, `module.source`, `provider`)
  - missing value for a variable (none of `var foo=...` flag, `var-file=foo.vars` flag, `TF_VAR_foo` environment variable, `terraform.tfvars`, or default value in the configuration)

▼ Module variable assignments are inherited from the parent module and do not need to be explicitly set. True or False?

- `False.`
- ▼ why?
  - Module variable assignments are not inherited from the parent module and must be explicitly set, either by assigning a value directly in the child module or by passing a value from the parent module. If a variable is not set, Terraform will either use the default value (if one is defined) or prompt the user to enter a value when running Terraform commands.

▼ If writing Terraform code that adheres to the Terraform style conventions, how would you properly indent each nesting level compared to the one above it?

- With two spaces

▼ HashiCorp Configuration Language (HCL) supports user-defined functions?

- No, HashiCorp Configuration Language (HCL) does not support user-defined functions.

▼ How can you trigger a run in a Terraform Cloud workspace that is connected to a Version Control System (VCS) repository?

- ▼ options
  - A. Only Terraform Cloud organization owners can set workspace variables on VCS connected workspaces
  - B. Commit a change to the VCS working directory and branch that the Terraform Cloud workspace is connected to
  - C. Only members of a VCS organization can open a pull request against repositories that are connected to Terraform Cloud workspaces
  - D. Only Terraform Cloud organization owners can approve plans in VCS connected workspaces
- ▼ Ans
  - B. Commit a change to the VCS working directory and branch that the Terraform Cloud workspace is connected to

▼ Which statement describes a goal of infrastructure as code?

- ▼ options
  - A. An abstraction from vendor specific APIs

- B. Write once, run anywhere
- C. A pipeline process to test and deliver software
- D. The programmatic configuration of resources

▼ Ans

- D. The programmatic configuration of resources

▼ When using Terraform to deploy resources into Azure, which scenarios are true regarding state files? (Choose two.)

▼ options

- A. When a change is made to the resources via the Azure Cloud Console, the changes are recorded in a new state file
- B. When a change is made to the resources via the Azure Cloud Console, Terraform will update the state file to reflect them during the next plan or apply
- C. When a change is made to the resources via the Azure Cloud Console, the current state file will not be updated
- D. When a change is made to the resources via the Azure Cloud Console, the changes are recorded in the current state file

▼ Ans

- B. When a change is made to the resources via the Azure Cloud Console, Terraform will update the state file to reflect them during the next plan or apply
- C. When a change is made to the resources via the Azure Cloud Console, the current state file will not be updated

▼ You need to deploy resources into two different cloud regions in the same Terraform configuration. To do that, you declare multiple provider configurations as follows What meta-argument do you need to configure in a resource block to deploy the resource to the `us-west-2` AWS region?

▼ options

- A. alias = west
- B. provider = west
- C. provider = aws.west
- D. alias = aws.west

▼ Ans

- C. provider = aws.west

```
variable "provider_alias" {}

resource "aws_instance" "foo" {
    provider = "aws.${var.provider_alias}"

    # ...
}
```

▼ You have declared an input variable called environment in your parent module. What must you do to pass the value to a child module in the configuration?

▼ options

- A. Add node_count = var.node_count
- B. Declare the variable in a terraform.tfvars file
- C. Declare a node_count input variable for child module
- D. Nothing, child modules inherit variables of parent module

▼ Ans

- C. Declare a node_count input variable for child module

▼ If a module declares a variable with a default, that variable must also be defined within the terraform module?

- False. If a Terraform module declares a variable with a default value, it is not mandatory to define the variable within the module. When a variable is declared with a default value in a module, it means that the module can be used without explicitly defining that variable, as the default value will be used if no other value is provided. However, if a value for that variable is provided when using the module, then the provided value will override the default value.

▼ Which option cannot be used to keep secrets out of Terraform configuration files?

▼ options

- A. Environment Variables
  B. Mark the variable as sensitive
  C. A Terraform provider
  D. A -var flag

▼ Ans

- B. Mark the variable as sensitive.
- The `sensitive` argument is used to prevent sensitive information from being displayed in the Terraform console output and state file. However, it is not designed to keep secrets out of the Terraform configuration file, which is a potential security risk.
- The recommended options to keep secrets out of Terraform configuration files are:
  - Use an external secrets management tool, such as HashiCorp Vault or AWS Secrets Manager, to store and retrieve sensitive information at runtime.
  - Use environment variables to pass secrets to the Terraform configuration, as environment variables are not stored in the Terraform configuration file.
  - Use the `var-file` flag to pass sensitive information from an external file that is not stored in the Terraform configuration file.

▼ Your risk management organization requires that new AWS S3 buckets must be private and encrypted at rest. How can Terraform Enterprise automatically and proactively enforce this security control?

▼ options

- A. With a Sentinel policy, which runs before every apply
  B. By adding variables to each TFE workspace to ensure these settings are always enabled
  C. With an S3 module with proper settings for buckets
  D. Auditing cloud storage buckets with a vulnerability scanning tool

▼ Ans

- `A. With a Sentinel policy, which runs before every apply.`
- Sentinel is a policy-as-code framework that can be integrated with Terraform Enterprise (TFE) to enforce policy compliance. Sentinel policies can be created to check that certain security controls are in place before Terraform applies any changes. In this case, a Sentinel policy could be created to ensure that new S3 buckets are private and encrypted at rest. The Sentinel policy could then be associated with the appropriate workspaces in TFE, and would run automatically before every `apply` to check that the policy is being followed.
- Option B is not ideal because it would require manual intervention each time a new workspace is created or updated, and there would be no guarantee that the variables are correctly set. Option C is a good practice to ensure consistency across all S3 buckets, but it does not automatically enforce the security control for new S3 buckets. Option D is a good practice to identify vulnerabilities, but it does not enforce the security control.

▼ Why do we need Multiple Provider instances?

```
Some of the example scenarios:a. multiple regions for a cloud platform
b. targeting multiple Docker hosts
c. multiple Consul hosts, etc.
```

▼ Third-party plugins should be manually installed. Is that true?

- `True.`

▼ The command `terraform init` cannot install third-party plugins? True or false?

- `True.`
- Install third-party providers by placing their plugin executables in the user plugins directory. The user plugins directory is in one of the following locations, depending on the host operating system
- Once a plugin is installed, `terraform init` can initialize it normally. You must run this command from the directory where the configuration files are located.

▼ By default, provisioners that fail will also cause the Terraform apply itself to fail. Is this true?

- `True.`

▼ You are working on the different workspaces and you want to use a different number of instances based on the workspace. How do you achieve that?

```
resource "aws_instance" "example" {
  count= "${terraform.workspace == "default" ? 5 : 1}"

  # ... other arguments
}
```

▼ You are working on the different workspaces and you want to use tags based on the workspace. How do you achieve that?

```
resource "aws_instance" "example" {
  tags = {
    Name = "web - ${terraform.workspace}"
  }

  # ... other arguments
}
```

▼ How do you do debugging terraform?

```
Terraform has detailed logs which can be enabled by setting theTF_LOG environment variable to any value. This will cause
detailed logs to appear on stderr.You can setTF_LOG to one of the log levelsTRACE,DEBUG,INFO,WARN orERROR to change the
 verbosity of the logs.TRACE is the most verbose and it is the default ifTF_LOG is set to something other than a log lev
el name.To persist logged output you can setTF_LOG_PATH in order to force the log to always be appended to a specific fi
le when logging is enabled. Note that even whenTF_LOG_PATH is set,TF_LOG must be set in order for any logging to be enab
led.
```

▼ If terraform crashes where should you see the logs?

```
crash.logIf Terraform ever crashes (a "panic" in the Go runtime), it saves a log file with the debug logs from the sessi
on as well as the panic message and backtrace to crash.log.
```

▼ What is the first thing you should do when the terraform crashes?

- panic message
- The most interesting part of a crash log is the panic message itself and the backtrace immediately following. So the first thing to do is to search the file for panic

▼ You are building infrastructure for different environments for example test and dev. How do you maintain separate states?

- directories.
- workspaces.

▼ What is the difference between directory-separated and workspace-separated environments?

- Directory separated environments rely on duplicate Terraform code, which may be useful if your deployments need differ, for example to test infrastructure changes in development. But they can run the risk of creating drift between the environments over time.
- Workspace-separated environments use the same Terraform code but have different state files, which is useful if you want your environments to stay as similar to each other as possible, for example if you are providing development infrastructure to a team that wants to simulate running in production.

▼ When you are working with the workspaces how do you access the current workspace in the configuration files?

- `${terraform.workspace}`

▼ When you are using workspaces where does the Terraform save the state file for the local state?

- `terraform.tfstate.d`
- For local state, Terraform stores the workspace states in a directory called `terraform.tfstate.d`.

▼ When you are using workspaces where does the Terraform save the state file for the remote state?

- For remote state, the workspaces are stored directly in the configured backend.

▼ How do you remove items from the Terraform state?

- `terraform state rm 'packet_device.worker'` Theterraform state rm command is used to remove items from the Terraform state.
- This command can remove single resources, single instances of a resource, entire modules, and more.

▼ Where do you find and explore terraform Modules?

- TheTerraform Registry makes it simple to find and use modules.
- The search query will look at module name, provider, and description to match your search terms. On the results page, filters can be used further refine search results.

▼ How do you make sure that modules have stability and compatibility?

- By default, `only verified modules` are shown in search results. Verified modules are reviewed by HashiCorp to ensure stability and compatibility. By using the filters, you can view unverified modules as well.

▼ How do you download any modules?

- To download any modules, you can use the `terraform get` command. This command retrieves modules and installs them in the `.terraform/modules` directory in your working directory. You can also specify the -update flag to update the modules to the latest version, or use the -get-plugins flag to download any required plugins.

▼ What is the syntax for referencing a registry module?

- The syntax for referencing a registry module is as follows:

```
<NAMESPACE>/<NAME>/<PROVIDER>
// for example
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0"
}
```

▼ What is the syntax for referencing a private registry module?

```
<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>// for example
module "vpc" {
```

```
  source= "app.terraform.io/example_corp/vpc/aws"
  version= "0.9.3"
}
```

▼ Why do we use modules for?

- Organize configuration
- Encapsulate configuration
- Re-use configuration
- Provide consistency and ensure best practices

▼ How do you call modules in your configuration?

- Your configuration can use module blocks to call modules in other directories. When Terraform encounters a module block, it loads and processes that module's configuration files.

▼ How many ways you can load modules?

- Local and remote modules.
- Modules can either be loaded from the local filesystem, or a remote source.
- Terraform supports a variety of remote sources, including the Terraform Registry, most version control systems, HTTP URLs, and Terraform Cloud or Terraform Enterprise private module registries.

▼ What are the best practices for using Modules?

- Start writing your configuration with modules in mind. Even for modestly complex Terraform configurations managed by a single person, you'll find the benefits of using modules outweigh the time it takes to use them properly.
- Use local modules to organize and encapsulate your code. Even if you aren't using or publishing remote modules, organizing your configuration in terms of modules from the beginning will significantlty reduce the burden of maintaining and updating your configuration as your infrastructure grows in complexity.
- Use the public Terraform Registry to find useful modules. This way you can more quickly and confidently implement your configuration by relying on the work of others to implement common infrastructure scenarios.
- Publish and share modules with your team. Most infrastructure is managed by a team of people, and modules are important way that teams can work together to create and maintain infrastructure. As mentioned earlier, you can publish modules either publicly or privately. We will see how to do this in a future guide in this series.

▼ What are the different source types for calling modules?

- Local paths
- Terraform Registry
- GitHub
- Generic Git, Mercurial repositories
- Bitbucket
- HTTP URLs
- S3 buckets
- GCS buckets

▼ What are the arguments you need for using modules in your configuration?

```
source and version
// example
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0"
}
```

▼ How do you access output variables from the modules?

- You can access output variables from modules by using the `${module.<MODULE_NAME>.<OUTPUT_NAME>}` syntax.

▼ When you use local modules you don’t have to do the command init or get every time there is a change in the local module. why?

- When installing a local module, Terraform will instead refer directly to the source directory.
- Because of this, Terraform will automatically notice changes to local modules without having to re-run terraform init or terraform get.

▼ When you use remote modules what should you do if there is a change in the module?

- When installing a remote module, Terraform will download it into the .terraform directory in your configuration's root directory.

▼ A simple configuration consisting of a single directory with one or more .tf files is a module. Is this true?

- `True.`

▼ When you are doing initialization with terraform init, you want to skip backend initialization. What should you do?

```
terraform init -backend=false
```

▼ When you are doing initialization with terraform init, you want to skip child module installation. What should you do?

```
terraform init -get=false
```

▼ When you are doing initialization with terraform init, you want to skip plugin installation. What should you do?

```
terraform init -get-plugins=false
```

- Skips plugin installation. Terraform will use plugins installed in the user plugins directory, and any plugins already installed for the current working directory. If the installed plugins aren't sufficient for the configuration, init fails.

▼ What is the flag you should use with the terraform plan to get detailed on the exit codes?

```
terraform plan -detailed-exitcodeReturn a detailed exit code when the command exits.


When provided, this argument changes the exit codes and their meanings to provide more granular information about what t
he resulting plan contains:* 0 = Succeeded with empty diff (no changes)
* 1 = Error
```

▼ How do you target only specific resources when you run a terraform plan?

```
-target=resource - AResource Address to target.

This flag can be used multiple times.
```

▼ How do you update the state prior to checking differences when you run a terraform plan?

```
terraform plan -refresh=true
```

▼ You have the following file and created two resources docker_image and docker_container with the command `terraform apply` and you go to the terminal and delete the container with the command `docker rm.` You come back to your configuration and run the command again. Does terraform recreates the resource?

```
resource "docker_image" "nginx" {
    name = "nginx:latest"
    keep_locally = false
}

resource "docker_container" "nginx" {
    image = docker_image.nginx.latest
    name = "nginxtutorial"
    ports {
        internal = 80
        external = 8080
    }
    upload {
        source = "${abspath(path.root)}/files/index.html"
        file = "/usr/share/nginx/html/index.html"
    }
}
```

- `Yes.` Terrsform creates the resource again since the execution plan says two resources and the terraform always maintains the desired state.

▼ How do you backup the state to the remote backend?

- When configuring a backend for the first time (moving from no defined backend to explicitly configuring one), Terraform will give you the option to migrate your state to the new backend. This lets you adopt backends without losing any existing state.
- To be extra careful, we always recommend manually backing up your state as well. You can do this by simply copying your `terraform.tfstate` file to another location.

▼ What is a partial configuration in terms of configuring Backends?

```
You do not need to specify every required argument in the backend configuration. Omitting certain arguments may be desir
able to avoid storing secrets, such as access keys, within the main configuration. When some or all of the arguments are
omitted, we call this apartial configuration.
```

▼ What are the ways to provide remaining arguments when using partial configuration?

- `Interactively :` Terraform will interactively ask you for the required values, unless interactive input is disabled. Terraform will not prompt for optional values.
- `File :` A configuration file may be specified via the init command line. To specify a file, use the-backend-config=PATH option when running terraform init. If the file contains secrets it may be kept in a secure data store, such as Vault, in which case it must be downloaded to the local disk before running Terraform.
- `Command-line key/value pairs :` Key/value pairs can be specified via the init command line. Note that many shells retain command-line flags in a history file, so this isn't recommended for secrets. To specify a single key/value pair, use the -backend-config="KEY=VALUE" option when running terraform init.

▼ What is the basic requirement when using partial configuration?

```
When using partial configuration, Terraform requires at a minimum that an empty backend configuration is specified in on
e of the root Terraform configuration files, to specify the backend type// Example
terraform {
  backend "consul" {}
}
```

▼ Give an example of passing partial configuration with Command-line Key/Value pairs?

```
terraform init \
    -backend-config="address=demo.consul.io" \
    -backend-config="path=example_app/terraform_state" \
    -backend-config="scheme=https"
```

▼ How to unconfigure a backend?

```
If you no longer want to use any backend, you can simply remove the configuration from the file.

Terraform will detect this like any other change and prompt you toreinitialize.As part of the reinitialization, Terrafor
m will ask if you'd like to migrate your state back down to normal local state.

Once this is complete then Terraform is back to behaving as it does by default.
```

▼ How do you encrypt sensitive data in the state?

- Terraform Cloud always encrypts state at rest and protects it with TLS in transit. Terraform Cloud also knows the identity of the user requesting state and maintains a history of state changes.
- This can be used to control access and track activity.Terraform Enterprise also supports detailed audit logging.
- The S3 backend supports encryption at rest when the encrypt option is enabled. IAM policies and logging can be used to identify any invalid access. Requests for the state go over a TLS connection.

▼ Backends are completely optional. Is this true?

- Backends are completely optional. You can successfully use Terraform without ever having to learn or use backends.
- However, they do solve pain points that afflict teams at a certain scale. If you're an individual, you can likely get away with never using backends.

▼ What are the benefits of Backends?

- Working in a team: Backends can store their state remotely and protect that state with locks to prevent corruption. Some backends such as Terraform Cloud even automatically store a history of all state revisions.
- Keeping sensitive information off disk: State is retrieved from backends on demand and only stored in memory. If you’re using a backend such as Amazon S3, the only location the state ever is persisted is in S3.
- Remote operations: For larger infrastructures or certain changes, terraform apply can take a long, long time. Some backends support remote operations which enable the operation to execute remotely. You can then turn off your computer and your operation will still complete. Paired with remote state storage and locking above, this also helps in team environments

▼ Why should you be very careful with the Force unlocking the state?

- Terraform has aforce-unlock command to manually unlock the state if unlocking failed.
- Be very careful with this command. If you unlock the state when someone else is holding the lock it could cause multiple writers. Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed.
- To protect you, the force-unlock command requires a unique lock ID. Terraform will output this lock ID if unlocking fails. This lock ID acts as anonce, ensuring that locks and unlocks target the correct lock.

▼ How many ways you can assign variables in the configuration?

```
Command-line flags

terraform apply -var 'region=us-east-1'
```

```
From a file
To persist variable values, create a file and assign variables within this file. Create a file named terraform.tfvars wi
th the following contents:

region = "us-east-1"terraform apply \
  -var-file="secret.tfvars" \
  -var-file="production.tfvars"
```

```
From environment varibles
Terraform will read environment variables in the form of TF_VAR_name to find the value for a variable. For example,
the TF_VAR_region variable can be set in the shell to set the region variable in Terraform.
```

```
UI input

If you execute terraform apply with any variable unspecified,
Terraform will ask you to input the values interactively.
These values are not saved, but this provides a convenient workflow
when getting started with Terraform.

UI input is not recommended for everyday use of Terraform.
```

▼ Does environment variables support List and map types?

- `No.`
- Environment variables can only populate string-type variables. List and map type variables must be populated via one of the other mechanisms.

▼ How do you provision infrastructure in a staging environment or a production environment using the same Terraform configuration?

```
You can use different varible files with the same configuration

// Example

// For development
terraform apply -var-file="dev.tfvars"

// For test
terraform apply -var-file="test.tfvars"
```

▼ What are the data types for the variables?

```
string
number
bool

list(<TYPE>)
set(<TYPE>)
map(<TYPE>)

object({<ATTR NAME> = <TYPE>, ... })
tuple([<TYPE>, ...])
```

▼ What is the Variable Definition Precedence?

- The above mechanisms for setting variables can be used together in any combination. If the same variable is assigned multiple values, Terraform uses the *last* value it finds, overriding any previous values. Note that the same variable cannot be assigned multiple values within a single source.
- Terraform loads variables in the following order, with later sources taking precedence over earlier ones:

```
* Environment variables
* Theterraform.tfvars file, if present.
* Theterraform.tfvars.json file, if present.
* Any*.auto.tfvars or*.auto.tfvars.json files, processed in lexical order of their filenames.

* Any -var and -var-file options on the command line, in the order they are provided. (This includes variables set by a
 Terraform Cloud workspace.)
```

▼ What are the dynamic blocks?

- some resource types include repeatable nested blocks in their arguments, which do not accept expressions
- You can dynamically construct repeatable nested blocks like setting using a special dynamic block type, which is supported inside resource, data, provider, and provisioner blocks.

- A dynamic block acts much like a for expression, but produces nested blocks instead of a complex typed value. It iterates over a given complex value, and generates a nested block for each element of that complex value.

▼ What are the best practices for dynamic blocks?

- Overuse of dynamic blocks can make configuration hard to read and maintain, so we recommend using them only when you need to hide details in order to build a clean user interface for a re-usable module.
- Always write nested blocks out literally where possible.

▼ What is Resource Address?

- A Resource Address is a string that references a specific resource in a larger infrastructure.
- An address is made up of two parts: `[module path][resource spec]`

▼ What is the Module path?

- A module path addresses a module within the tree of modules.
- It takes the form: `module.A.module.B.module.C...`
- Multiple modules in a path indicate nesting. If a module path is specified without a resource spec, the address applies to every resource within the module. If the module path is omitted, this addresses the root module.

▼ What is the Resource spec?

- A resource spec addresses a specific resource in the config.

```
resource_type.resource_name[resource index]

*resource_type - Type of the resource being addressed.
*resource_name - User-defined name of the resource.
*[resource index] - an optional index into a resource with multiple instances, surrounded by square brace characters ([
 and ]).

// Examples
resource "aws_instance" "web" {
  # ...
  count= 4
}

aws_instance.web[3]  // Refers to only last instance
aws_instance.web     // Refers to all four "web" instances.

resource "aws_instance" "web" {
  # ...
  for_each= {
    "terraform": "value1",
    "resource":  "value2",
    "indexing":  "value3",
    "example":   "value4",
  }
}
aws_instance.web["example"] // Refers to only the "example" instance in the config.
```

▼ What are complex types and what are the collection types Terraform supports?

- A *complex* type is a type that groups multiple values into a single value.

▼ There are two categories of complex types

▼ collection types (for grouping similar values)

```
* list(...): a sequence of values identified by consecutive whole numbers starting with zero.

* map(...): a collection of values where each is identified by a string label.

* set(...): a collection of unique values that do not have any secondary identifiers or ordering.
```

▼ structural types (for grouping potentially dissimilar values).

```
*object(...): a collection of named attributes that each have their own type.

*tuple(...): a sequence of elements identified by consecutive whole numbers starting with zero, where each element
```

▼ What are the named values available and how do we refer to?

- Terraform makes several kinds of named values available.
- Each of these names is an expression that references the associated value.
- you can use them as standalone expressions, or combine them with other expressions to compute new values.
- `<RESOURCE TYPE>.<NAME>` is an object representing a managed resource of the given type and name. The attributes of the resource can be accessed using dot or square bracket notation.
- `var.name` is the value of the input variable of the given name.
- local.name is the value of the local value of the given name.
- `module.<MODULE NAME>.<OUTPUT NAME>` is the value of the specified output value from a child module called by the current module.
- `data.<DATA TYPE>.<NAME>` is an object representing a data resource of the given data source type and name. If the resource has the count argument set, the value is a list of objects representing its instances. If the resource has the for_each argument set, the value is a map of objects representing its instances.
- `path.module` is the filesystem path of the module where the expression is placed.
- `path.root` is the filesystem path of the root module of the configuration.
- `path.cwd` is the filesystem path of the current working directory. In normal use of Terraform this is the same as path.root, but some advanced uses of Terraform run it from a directory other than the root module directory, causing these paths to be different.
- `terraform.workspace` is the name of the currently selected workspace.

▼ What is Sentinel?

- It is an embedded policy-as-code framework integrated with the HashiCorp Enterprise products.
- It enables fine-grained, logic-based policy decisions, and can be extended to use information from external sources.

▼ What is the benefit of Sentinel?

- Codifying policy removes the need for ticketing queues, without sacrificing enforcement.
- One of the other benefits of Sentinel is that it also has a full testing framework.
- Avoiding a ticketing workflow allows organizations to provide more self-service capabilities and end-to-end automation, minimizing the friction for developers and operators.

▼ What is the difference between public and private module registries when defined source?

```
The public registry uses a three-part
<NAMESPACE>/<MODULE NAME>/<PROVIDER> format

private modules use a four-part
<HOSTNAME>/<ORGANIZATION>/<MODULE NAME>/<PROVIDER> format
```

▼ What are the benefits of workspaces?

Workspaces in Terraform provide several benefits:

1. Environment Isolation: Workspaces allow you to create separate environments, such as dev, staging, and production, with their own state and variables. This provides a clear separation between different environments, making it easier to manage and apply changes to each environment separately.

2. Resource Reuse: Workspaces enable you to reuse Terraform code across multiple environments. You can define modules and apply them to different workspaces, making it easier to manage and update resources across different environments.
3. Parallelism: Workspaces allow you to run Terraform apply commands in parallel for different workspaces, which can save time and reduce the risk of conflicts between different changes.
4. Collaboration: Workspaces can be used to enable collaboration between different teams or individuals, as each workspace can be associated with different permissions and access controls.
5. Testing: Workspaces allow you to test changes before applying them to production. You can create a separate workspace for testing and validation, ensuring that any changes are validated and tested before being applied to a live environment.

▼ You are configuring a remote backend in the terraform cloud. You didn't create an organization before you do terraform init. Does it work?

- No, it won't work. Before initializing the Terraform Cloud backend, you need to create an organization in Terraform Cloud.

▼ You are configuring a remote backend in the terraform cloud. You didn't create a workspace before you do terraform init. Does it work?

- No, it won't work. Before initializing the Terraform Cloud backend, you need to create a workspace in Terraform Cloud.

▼ What is Run Triggers?

- Terraform Cloud's run triggers allow you to link workspaces so that a successful apply in a source workspace will queue a run in the workspace linked to it with a run trigger.
- For example, adding new subnets to your network configuration could trigger an update to your application configuration to rebalance servers across the new subnets.

▼ What is the benefit of Run Triggers?

- When managing complex infrastructure with Terraform Cloud, organizing your configuration into different workspaces helps you to better manage and design your infrastructure.
- Configuring run triggers between workspaces allows you to set up infrastructure pipelines as part of your overall deployment strategy.

▼ What are the available permissions that terraform clouds can have?

- Terraform Cloud teams can have read, plan, write, or admin permissions on individual workspaces.

▼ Who can grant permissions on the workspaces?

- Organization owners grant permissions by grouping users into teams and giving those teams priviliges based on their need for access to individual workspaces.

▼ What is the downside to using terraform to interact with sensitive data, such as reading secrets from Vault?

- Secrets are persisted to the state file.

▼ Is list() function is deprecated?

- `list()` is deprecated in Terraform. It has been replaced with `tolist()`. The `tolist()` function converts a value to a list of one or more elements. It is recommended to use `tolist()` instead of `list()` in any new Terraform configurations.
- In terraform v0.12

▼ What are the differences between without using backend at all & using local backend?

- If you don't use a backend at all, Terraform will store the state file locally on the machine where Terraform is run. This can be problematic in a team environment, as it can result in multiple users updating the same state file simultaneously, leading to potential conflicts and errors.

- On the other hand, if you use the local backend, Terraform will store the state file in a local file system, but with the added benefit of locking the state file so that only one user can modify it at a time. This helps prevent conflicts and errors that can occur when multiple users modify the same state file simultaneously.
- In summary, using a local backend adds the benefit of locking the state file to prevent concurrent modifications, while not using a backend at all can result in conflicts and errors when multiple users update the state file simultaneously.

▼ Disable detailed logs by setting ?export TF_LOG=i,e. empty.

▼ options

- A) /home/quiz_experts/iac/workspace.d/DEV/terraform.tfstate
  B) /home/quiz_experts/iac/terraform.tfstate.d/terraform.tfstate
  C) /home/quiz_experts/iac/terraform.tfstate
  D) /home/quiz_experts/iac/DEV/terraform.tfstate
  E) /home/quiz_experts/iac/workspace.d/terraform.tfstate
  F) /home/quiz_experts/iac/terraform.tfstate.d/DEV/terraform.tfstate

▼ Ans

- F
- This directory should be treated similarly to local-only terraform.tfstate
- So for non-default workspaces state will be stored in a file - `terraform.tfstate.d/<your-workspace-name>/terraform.tfstate`.

▼ Terraform cannot reason about what the provisioner does; hence if a creation-time provisioner fails for a resource, Terraform will plan to destroy and recreate resource upon the next terraform apply.

- This statement is true. If a creation-time provisioner fails during the initial resource creation, Terraform cannot guarantee the state of the resource, so it will plan to destroy and recreate the resource upon the next `terraform apply`
  . This can cause unwanted downtime or data loss, so it's generally not recommended to rely heavily on provisioners in Terraform. Instead, it's better to use configuration management tools or other methods to manage the desired state of your resources.