Hashicorp Terraform Associate (003) Exam Questions

Ashish Lahoti Last Modified: May 13, 2023 Cloud

PAGE CONTENT

A comprehensive list of 200+ unique practice exam questions for Hashicorp Terraform Associate Certification (003).

200+ Practice Exam Questions

Go through the list of compiled questions for Hashicorp Terraform Associate certification (003) exam. All the exam questions are categorized based on different Terraform features for easy navigation. It is not an exam dump but you can expect similar questions in the real exam.

It took a lot of effort to compile these questions. If these exam questions helped you in the preparation, consider

> Also follow the post for exam guide and notes:- **Hashicorp Terraform Associate (003) Exam Guide**

Happy Learning and best of luck for the exam!

Infrastructure as Code (IaC)

Q1. What is the advantage of Infrastructure as Code tools like Terraform?

- [ ] Manage and track infrastructure
- [ ] Automate infrastructure changes
- [ ] Resusable configuration
- [ ] Collaboration using VCS (version control system)
- [ ] All of the above

Reference: https://developer.hashicorp.com/terraform/intro

Q2. Which of the following best describes Terraform?

- [ ] A programming language
- [ ] An infrastructure as code (IaC) tool
- [ ] A cloud provider
- [ ] A containerization tool

Reference: https://developer.hashicorp.com/terraform/intro

Q3. What is the main advantage to use Terraform as the IaC tool?

- [ ] Terraform can manage infrastructure on multiple cloud platforms.
- [ ] Terraform's state allows you to track resource changes throughout your deployments.
- [ ] You can commit your configurations to version control to safely collaborate on infrastructure.
- [ ] All of the above

Reference: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code

Q4. Which are some of the benefits of using Infrastructure as Code in an organization? (select three)

- [ ] IaC is written as an imperative approach, where specific commands need to be executed in the correct order
- [ ] IaC uses a human-readable configuration language to help you write infrastructure code quickly
- [ ] IaC allows you to commit your configurations to version control to safely collaborate on infrastructure
- [ ] IaC code can be used to manage infrastructure on multiple cloud platforms

Q5. Which is NOT a benefit of using Infrastructure as Code with Terraform?

- [ ] You can commit your configurations to version control to safely collaborate on infrastructure
- [ ] Manage infrastructure on multiple cloud platforms
- [ ] Reducing vulnerabilities in your publicly-facing applications
- [ ] The human-readable configuration language helps you write infrastructure code quickly

Reference: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code

Q6. Does Terraform support multiple cloud deployment?

- [ ] true
- [ ] false

Terraform lets you use the same workflow to manage multiple providers and handle cross-cloud dependencies. This simplifies management and orchestration for large-scale, multi-cloud infrastructures.
Reference: https://developer.hashicorp.com/terraform/intro/use-cases#multi-cloud-deployment

Q7. What is the core terraform workflow?

- [ ] Plan, write, apply.
- [ ] Write, plan, apply.
- [ ] Apply, write, plan.
- [ ] Apply, plan, write.

Reference: https://developer.hashicorp.com/terraform/intro/core-workflow

Terraform Cloud

Q8. What are three Terraform Cloud features? (Choose 3 answers)

- [ ] Remote state management.
- [ ] Remote Terraform Execution.
- [ ] Private Module Registry.
- [ ] Terraform Linting.

Reference: https://developer.hashicorp.com/terraform/cloud-docs/overview

Q9. Which of the following options are NOT available in Terraform OSS/CLI and Terraform Cloud Free Tier?

- [ ] Audit Logging
- [ ] Policy as code (Sentinel)
- [ ] Workspaces
- [ ] Public Module Registry
- [ ] Single Sign-On (SSO)

Reference: https://www.hashicorp.com/products/terraform/pricing

Q10. Which of the following Terraform feature is available in the Enterprise edition but NOT in Terraform Cloud for Business edition?

- [ ] Application-level logging
- [ ] SSO (Single Sign On)
- [ ] Drift detection
- [ ] Audit logging

Reference: https://www.hashicorp.com/products/terraform/pricing

Q11. Your boss has asked you to come up with a new cloud automation provider that supports a Private Module registry as part of the offering. Which Cloud Provider and plan do you choose?

- [ ] Terraform Cloud with a Terraform Enterprise
- [ ] Amazon Web Services and the Enterprise Terraform Plus Plan
- [ ] The Azure Supercharged Automation Professional Direct plan from Microsoft
- [ ] The Google GCP Terraform Deluxe Plan

https://developer.hashicorp.com/terraform/cloud-docs/registry#private-providers-and-modules

Terraform Configuration

Q12. Which file is typically used to define resources in a Terraform configuration?

- [ ] main.tf
- [ ] terraform.tfvars
- [ ] variables.tf
- [ ] outputs.tf

The `main.tf` will contain the main set of configuration for your module. You can also create other configuration files and organize them however makes sense for your project. The `variables.tf` and `outputs.tf` contains the variable and output definitions for your module. A typical file structure for the module is:-

```
.
├── LICENSE
├── README.md
├── main.tf
├── variables.tf
├── outputs.tf
```

Reference: https://developer.hashicorp.com/terraform/tutorials/modules/module-create

Q13. In the following code snippet, the block type is identified by which string?

```
resource "aws_instance" "db" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
}
```

- [ ] `resource`
- [ ] `aws_instance`
- [ ] `db`
- [ ] `instance_type`

Reference: https://developer.hashicorp.com/terraform/language/syntax/configuration#blocks

Q14. Which of the following is NOT a valid Terraform block type?

- [ ] provider
- [ ] resource
- [ ] output
- [ ] module
- [ ] data
- [ ] bucket

Terraform has following block types: terraform, provider, resource, variable, locals, data, module, output, and provisioner
Reference: https://dev.to/af/hashicorp-configuration-language-hcl-blocks-5627

Q15. What is the workflow for deploying new infrastructure with Terraform?

- terraform plan to import the current infrastructure to the state file, make code changes, and terraform apply to update the infrastructure.
- Write a Terraform configuration, run terraform show to view proposed changes, and terraform apply to create new infrastructure.
- terraform import to import the current infrastructure to the state file, make code changes, and terraform apply to update the infrastructure.
- Write a Terraform configuration, run terraform init, run terraform plan to view planned infrastructure changes, and terraform apply to create new infrastructure.

Reference: https://developer.hashicorp.com/terraform/intro/core-workflow

Q16. Which language does terraform configuration support from the below list?

- XML
- JSON
- Hashicorp Configuration Language (HCL)
- YAML

Reference: https://developer.hashicorp.com/terraform/language/syntax

Q17. When writing Terraform code, how many spaces between each nesting level does HashiCorp recommends that you use?

- 4
- 2
- 1
- 5

Reference: https://developer.hashicorp.com/terraform/language/syntax/style

Q18. Which of the following statements represents the most accurate statement about the Terraform language?

- Terraform is a mutable, imperative, Infrastructure as Code provisioning language based on Hashicorp Configuration Language, or optionally YAML.
- Terraform is an immutable, declarative, Infrastructure as Code provisioning language based on Hashicorp Configuration Language, or optionally JSON.
- Terraform is a mutable, declarative, Infrastructure as Code configuration management language based on Hashicorp Configuration Language, or optionally JSON.
- Terraform is an immutable, imperative, Infrastructure as Code configuration management language based on Hashicorp Configuration Language, or optionally JSON.

Configuration management tool like Chef and Puppet install and manage software on a machine that already exists. Terraform is not a configuration management tool, it is an Infrastructure provisioning tool to bootstrap and initialize resources.
Reference: https://developer.hashicorp.com/terraform/intro/vs/chef-puppet

Q19. Terraform is distributed as a single binary and available for many different platforms. Which of the following platform is NOT supported?

- Solaris
- AIX
- FreeBSD
- OpenBSD

There is no Terraform binary for AIX. Terraform is available for:- macOS, Windows, Linux, FreeBSD, OpenBSD, and Solaris.
Reference: https://developer.hashicorp.com/terraform/downloads

Q20. Which of the following Terraform files should be ignored by Git when committing code to a repo? (select two)

- `output.tf`
- `terraform.tfstate`
- `terraform.tfvars`
- `variables.tf`

The `.tfstate` and `.tfvars` might contain sensitive data and should be added in `.gitignore` file
Reference: https://github.com/github/gitignore/blob/main/Terraform.gitignore

Q21. You have been given requirements to create a security group for a new application. Since your organization standardizes on Terraform, you want to add this new security group with the fewest number of lines of code. What feature could you use to iterate over a list of required tcp ports to add to the new security group?

- terraform import
- dynamic blocks
- splat expression
- dynamic backend

You can dynamically construct repeatable nested blocks using a special `dynamic` block type, which is supported inside `resource`, `data`, `provider`, and `provisioner` blocks
Reference: https://developer.hashicorp.com/terraform/language/expressions/dynamic-blocks

Q22. Which of the following is the best description of a `dynamic` block?

- requests that Terraform read from a given data source and export the result under the given local name
- declares a resource of a given type with a given local name
- produces nested configuration blocks instead of a complex typed value
- exports a value exported by a module or configuration

A dynamic block acts much like a for expression, but produces nested blocks instead of a complex typed value
Reference: https://developer.hashicorp.com/terraform/language/expressions/dynamic-blocks

Q23. Which one of the following is considered as a Terraform plugin?

- Terraform provisioner
- Terraform module
- Terraform provider
- Terraform registry

A provider is a plugin that allows Terraform to manage a specific cloud provider or service.
Reference: https://developer.hashicorp.com/terraform/plugin

Q24. Terraform remembers the compatible version of dependencies such as providers and modules through dependency lock file. What is the name of that file?

- `.terraform.lock.hcl`
- `.terraform.lock.tf`
- `.dependency.lock.hcl`
- `.dependency.lock.tf`

The dependency lock file is always named `.terraform.lock.hcl`, and this name is intended to signify that it is a lock file for various items that Terraform caches in the `.terraform` subdirectory of your working directory. Terraform automatically creates or updates the dependency lock file each time you run the `terraform init` command. You should include this file in your version control repository so that you can discuss potential changes to your external dependencies via code review, just as you would discuss potential changes to your configuration itself.
Reference: https://developer.hashicorp.com/terraform/language/files/dependency-lock

Q25. Terraform Core is a statically-compiled binary written in the _____ programming language.

- Java
- C#
- Python
- Go

Terraform Core is a statically-compiled binary written in the Go programming language. The compiled binary is the command line tool (CLI) `terraform`, the entrypoint for anyone using Terraform. The code is open source and hosted at https://github.com/hashicorp/terraform.

Reference: https://developer.hashicorp.com/terraform/plugin/how-terraform-works#terraform-core

Q26. Terraform builds a dependency graph from the Terraform configurations. Which is NOT a correct step of building a Graph?

- Resources are mapped to provisioners if they have any defined.
- Resources are mapped to providers.
- Resources nodes are added to the graph from the configuration.
- Resources are not added to the graph that are no longer present in the configuration but are present in the state file.

If a state is present, any "orphan" resources are added to the graph. Orphan resources are any resources that are no longer present in the configuration but are present in the state file. Orphans never have any configuration associated with them, since the state file does not store configuration.

Reference: https://developer.hashicorp.com/terraform/internals/graph#building-the-graph

Terraform State

Q27. Which of the following best describes a Terraform state file?

- A file that contains a list of available Terraform providers
- A file that stores the current state of infrastructure managed by Terraform
- A file that contains a list of Terraform modules used in a configuration
- A file that stores the output of a Terraform plan

Reference: https://developer.hashicorp.com/terraform/language/state

Q28. The Terraform state always matches to the remote cloud resources defined in the configuration?

- true
- false

No, terraform state file not always match to the cloud resources if there is any manual update in the resources from cloud console.

Reference: https://developer.hashicorp.com/terraform/language/state

Q29. Usernames and passwords referenced in the Terraform code, even as variables, will end up in plain text in the state file?

- true
- false

Terraform state can contain sensitive data, depending on the resources in use and your definition of "sensitive." The state contains resource IDs and all resource attributes. For resources such as databases, this may contain usernames and passwords.

Reference: https://developer.hashicorp.com/terraform/language/state/sensitive-data

Q30. Without using a state file, terraform can inspect cloud resources on every run to validate that the real-world resources match the desired state.

- true
- false

State is a necessary requirement for Terraform to function. Terraform requires state file to map Terraform config to the real world.

Reference: https://developer.hashicorp.com/terraform/language/state/purpose

Q31. You injected some secrets from variables into your Terraform configuration. What happens after you run the `terraform apply` command and they are loaded into state?

- They are shown in clear-text.
- They are shown as their referenced variables.
- They are shown as encrypted values.
- They are omitted from state.

Terraform state can contain sensitive data, depending on the resources in use and your definition of "sensitive." The state contains resource IDs and all resource attributes. For resources such as databases, this may contain initial passwords.

Reference: https://developer.hashicorp.com/terraform/language/state/sensitive-data

Q32. What is the name of the default file where Terraform stores the state?

Type your answer in the field provided. The text field is not case-sensitive and all variations of the correct answer are accepted.

- *terraform.tfstate*

Reference: https://developer.hashicorp.com/terraform/language/state

Q33. Where is the default location that Terraform stores its state in?

- The current working directory in which Terraform was run.
- At the users root directory.
- In the same location that Terraform is installed. E.g. /usr/bin/terraform
- In ~/.terraform.d/plugins

In the default configuration, Terraform stores the state in a file in the current working directory where Terraform was run.

Reference: https://developer.hashicorp.com/terraform/language/state/purpose#syncing

Q34. What is the recommended way to implement Terraform's state for larger teams?

- By configuring a remote backend such that multiple teams can work in tandem and know which resources are being created and destroyed.
- By sticking the state in a cloud instance, and having team members SSH into the instance to work on their configuration files.
- Having your state synced to a github repo for members to compare to.
- By using the daily standup you are a part of sot that you can share changes to the state file.

Remote state is the recommended solution to this problem. With a fully-featured state backend, Terraform can use remote locking as a measure to avoid two or more different users accidentally running Terraform at the same time, and thus ensure that each Terraform run begins with the most recent updated state.

Reference: https://developer.hashicorp.com/terraform/language/state/purpose#syncing

Q35. You are part of a large DevOps team using the current version of Terraform, and there can be multiple changes going on to your terraform files across the company. What would you do to ensure that the state file is locked when you run `terraform apply`?

- Add the -lock=true flag to the command.
- Nothing, terraform will manage the locking by itself.
- First run terraform plan to lock in your proposed changes. Then run terraform apply to commit them.
- Add the -state-lock=true to the command.

State locking happens automatically on all operations that could write state. You won't see any message that it is happening. If state locking fails, Terraform will not continue. You can disable state locking for most commands with the `-lock` flag but it is not recommended.

Reference: https://developer.hashicorp.com/terraform/language/state/locking

Q36. If supported by your backend, Terraform will lock your state for all operations that could write state. What purpose does this serve?

- Prevents others from committing Terraform code that could override your updates.
- Locks colleagues from making manual changes to the managed infrastructure
- This prevents others from acquiring the lock and potentially corrupting your state.
- Ensures the state file cannot be moved after the initial `terraform apply`

Reference: https://developer.hashicorp.com/terraform/language/state/locking

Q37. Which of the following Terraform backend type supports state locking?

- consul
- kubernetes
- s3
- All of the above

Not all backends support locking. Following Terraform backend types supports state locking:- local, remote, azurerm, consul, cos, gcs, http, kubernetes, oss, pg, and s3

Reference: https://developer.hashicorp.com/terraform/language/state/locking

Q38. You manage the AWS cloud resources using Terraform and want to destroy all dev resources to save cost. However, your team member request you to keep the Amazon Aurora dev instance running. How can you destroy all cloud resources without impacting the database instance?

- run `terraform state rm` command to remove the database instance from terraform state before running `terraform destroy` command
- take a snapshot of database, run `terraform destroy`, and then recreate database by restoring the snapshot
- run a `terraform destroy`, modify configuration file to include only database instance, and then run `terraform apply`
- manually delete the other resource from AWS

You can use `terraform state rm` in the less common situation where you wish to remove a binding to an existing remote object without first destroying it, which will effectively make Terraform "forget" the object while it continues to exist in the remote system.

Reference: https://developer.hashicorp.com/terraform/cli/commands/state/rm

Q39. You manage the AWS cloud resources using Terraform and want to follow new naming standard for the local name within resource block. However, you don't want Terraform to replace the object after changing your configuration files. How can you change the local name from `data-bucket` to `prod-aws-s3-bucket` in the following resource block:-

```
resource "aws_s3_bucket" "data-bucket" {
  bucket = "prod-data-bucket"

  tags = {
    Name        = "prod-data-bucket"
    Environment = "prod"
  }
}
```

After renaming the local name of the resource block, what command would you run to update the local name while ensuring Terraform does not replace the existing resource?

- `terraform apply -refresh-only`
- `terrafrorm apply -replace aws_s3_bucket.data-bucket`
- `terraform state mv aws_s3_bucket.data-bucket aws_s3_bucket.prod-aws-s3-bucket`
- `terraform state rm aws_s3_bucket.data-bucket`

You can use `terraform state mv` in the less common situation where you wish to retain an existing remote object but track it as a different resource instance address in Terraform, such as if you have renamed a resource block or you have moved it into a different module in your configuration.

Reference: https://developer.hashicorp.com/terraform/cli/commands/state/mv

Q40. Which common action does not cause Terraform to refresh its state?

- `terraform state list`
- `terraform plan`
- `terraform apply`
- `terraform destroy`

Running a `terraform state list` does not cause Terraform to refresh its state. This command simply reads the state file but it will not modify it.

Reference: https://developer.hashicorp.com/terraform/cli/commands/state/list

Terraform Commands

Q41. You have recently added new resource blocks to your configuration from a different provider. What command do you need to run before you can run a `terraform plan/apply` ?

- `terraform plan`
- `terraform apply`
- `terraform init`
- `terraform validate`

The command `terraform init` is the first command that should be run after writing a new Terraform configuration, cloning an existing one from version control, adding new provider or module before you run `terraform plan/apply`

Reference: https://developer.hashicorp.com/terraform/cli/commands/init

Q42. How can you check out the configuration from version control and initialize a directory?

- `terraform init -from-module={MODULE-SOURCE}`
- `terraform init -source={PATH}`
- `terraform init {PATH}`
- `terraform init -plugin-dir={PATH}`

Given a version control source, `terraform init -from-module={MODULE-SOURCE}` can serve as a shorthand for checking out a configuration from version control and then initializing the working directory for it.

Reference: https://developer.hashicorp.com/terraform/cli/commands/init#copy-a-source-module

Q43. When you add a new module to a configuration, Terraform must download it before it can be used. What two commands can be used to download and update modules? (select two)

- `terraform plan`
- `terraform refresh`
- `terraform init`
- `terraform get`

Whenever you add a new module to a configuration, Terraform must install the module before it can be used. Both the `terraform get` and `terraform init` commands will install and update modules.

Reference: https://developer.hashicorp.com/terraform/tutorials/modules/module-create#install-the-local-module

Q44. Which command is used to create an execution plan in Terraform?

- `terraform plan`
- `terraform apply`
- `terraform init`
- `terraform validate`

Reference: https://developer.hashicorp.com/terraform/cli/commands/plan

Q45. By default, when running `terraform plan`, what files are scanned?

- All *.tf files in the current directory.
- Only files in the .terraform directory
- Only files you specify with the -file-path flag.
- All files on your hard drive.

Reference: https://developer.hashicorp.com/terraform/cli/commands/plan

Q46. Which command is used to apply changes to infrastructure in Terraform?

- `terraform destroy`
- `terraform apply`
- `terraform plan`
- `terraform validate`

Reference: https://developer.hashicorp.com/terraform/cli/commands/apply

Q47. Which command is used to destroy infrastructure in Terraform?

- `terraform destroy`
- `terraform apply`
- `terraform plan`
- `terraform validate`

Reference: https://developer.hashicorp.com/terraform/cli/commands/destroy

Q48. What two options are available to delete all of your managed infrastructure? (select two)

- `terraform init -destroy`
- `terraform apply -destroy`
- `terraform destroy`
- `terraform plan -destroy`

The `terraform destroy` command is just a convenience alias for the `terraform apply -destroy` command

Reference: https://developer.hashicorp.com/terraform/cli/commands/destroy#usage

Q49. If different teams are working on the same configuration. How do you make files to have consistent formatting?

- `terraform fmt`
- `terraform apply`
- `terraform plan`
- `terraform validate`

Reference: https://developer.hashicorp.com/terraform/cli/commands/fmt

Q50. What Terraform command modifies a HashiCorp Configuration Language (HCL) file to adhere to the recommended spacing rules for HCL files?

- `terraform fmt`
- `terraform apply`
- `terraform plan`
- `terraform validate`

Reference: https://developer.hashicorp.com/terraform/cli/commands/fmt

Q51. Your teammate is worried that if they run the terraform fmt command on their current directory, it will change their configuration files too much. What flag do you tell them to pass into the command such that they can see the differences?

- `-diff`
- `-check`
- `-refresh`
- `-list=true`

The `terraform fmt -diff` command display diffs of formatting changes

Reference: https://developer.hashicorp.com/terraform/cli/commands/fmt#diff

Q52. You need to ensure your Terraform is easily readable and follows the HCL canonical format and style. In the current directory, you have a `main.tf` that calls modules stored in a `modules` directory. What command could you run to easily rewrite your Terraform to follow the HCL style in both the current directory and all sub-directories?

- `terraform fmt -check -recursive`
- `terraform fmt -diff`
- `terraform fmt -check`
- `terraform fmt -list=true`

The `terraform fmt -recursive` command process files in subdirectories. By default, only the given directory (or current directory) is processed.

Reference: https://developer.hashicorp.com/terraform/cli/commands/fmt#recursive

Q53. Which command can be used to verify whether a configuration is syntactically valid and internally consistent?

- `terraform validate`
- `terraform apply`
- `terraform plan`
- `terraform fmt`

The `terraform validate` command runs checks that verify whether a configuration is syntactically valid and internally consistent.

Reference: https://developer.hashicorp.com/terraform/cli/commands/validate

Q54. How would you get the JSON output of the `terraform validate` command?

- `terraform validate -json`
- `terraform validate json`
- `terraform validate -output=json`
- `terraform json validate`

When you use the `-json` option, Terraform will produce validation results in JSON format to allow using the validation result for tool integrations, such as highlighting errors in a text editor.

Reference: https://developer.hashicorp.com/terraform/cli/commands/validate#json-output-format

Q55. Does the `terraform validate` command connect to remote APIs and state when being ran?

- No it does not.
- Only if configured to do so on the backend.
- If the -remote=true is set, yes it does.
- If there are providers set, it will attempt to.

The `terraform validate` command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

Reference: https://developer.hashicorp.com/terraform/cli/commands/validate

Q56. Which command provides an interactive command-line console for evaluating and experimenting with expressions?

- `terraform show`
- `terraform eval`
- `terraform console`
- `terraform exec`

The `terraform console` command provides an interactive console for evaluating and experimenting with expressions.

Reference: https://developer.hashicorp.com/terraform/cli/commands/console

Q57. Which command is used to extract the value of an output variable from the state file?

- `terraform exec`
- `terraform show`
- `terraform output`
- `terraform state`

The `terraform output` command is used to extract the value of an output variable from the state file.

Reference: https://developer.hashicorp.com/terraform/cli/commands/output

Q58. You have defined the values for your variables in the file `terraform.tfvars`, and saved it in the same directory as your Terraform configuration. Which of the following commands will use those values when creating an execution plan?

- `terraform plan`
- `terraform plan -var-file=terraform.tfvars`
- All of the above
- None of the above

You can specify Variable definition file using `-var-file` option. Terraform also automatically loads a number of variable definitions files if they are present: `terraform.tfvars` or `terraform.tfvars.json`.

Reference: https://developer.hashicorp.com/terraform/language/values/variables#variable-definitions-tfvars-files

Q59. Which two steps are required to provision new infrastructure in the Terraform workflow? Choose *TWO* correct answers.

- `terraform init`
- `terraform import`
- `terraform apply`
- `terraform validate`
- `terraform destroy`

The `terraform init` command is prerequisite to initialize the Terraform workspace before you can run `terraform apply` to provision new infrastructure. When you run `terraform apply` without passing a saved plan file, Terraform automatically creates a new execution plan as if you had run `terraform plan`.

Reference: https://developer.hashicorp.com/terraform/intro/core-workflow

Q60. It is necessary to run `terraform plan` command before `terraform apply` in the Terraform Workflow.

- true
- false

No, it is not necessary. When you run `terraform apply` without passing a saved plan file, Terraform automatically creates a new execution plan as if you had run `terraform plan`.

Reference: https://developer.hashicorp.com/terraform/cli/commands/apply#automatic-plan-mode

Q61. Someone created few resources manually using the Azure console. You have company policy to manage all infrastructure using Terraform. How can you manage manually deployed resource using Terraform without impacting other resources?

- run a `terraform get` to get the manually deployed resources that are not under Terraform management
- delete the resources created manually using the Azure console and add these resource in terraform configuration, then run `terraform apply`
- use `terraform import` to import existing resources under Terraform management
- resources created outside Terraform cannot be managed by Terraform

Reference: https://developer.hashicorp.com/terraform/cli/commands/import

Q62. How does `terraform import` run?

- As a part of `terraform init`
- As a part of `terraform plan`
- As a part of `terraform refresh`
- By an explicit call
- All of the above

Reference: https://developer.hashicorp.com/terraform/cli/commands/import

Q63. What must be provided with the `terraform import` command for Terraform to successfully import resources?

- Resource ID, resource type, and the resource name.
- The resource name.
- The full resource ARN.
- Only resource Id

The `terraform import aws_instance.foo i-abcd1234` command will find the existing resource from AWS instance ID `i-abcd1234` and import it into your Terraform state at the given ADDRESS `aws_instance.foo`.
- ID is dependent on the resource type being imported. For example, for AWS instances it is the instance ID (e.g. i-abcd1234) but for AWS Route53 zones it is the zone ID (e.g. Z12ABC4UGMOZ2N).
- ADDRESS must be a valid resource address which is made up of two parts: `resource_type.resource_name`

Reference: https://developer.hashicorp.com/terraform/cli/commands/import

Q64. You want to use `terraform import` to start managing infrastructure that was not originally provisioned through infrastructure as code. Before you can import the resource's current state, what must you do to prepare to manage these resources using Terraform?

- update the Terraform configuration file to include the new resources that match the resources you want to import
- modify the Terraform state file to add the new resources so Terraform will have a record of the resources to be managed

- shut down or stop using the resources being imported so no changes are inadvertently missed
- run `terraform apply -refresh-only` to ensure that the state file has the latest information for existing resources.

You should first add the new resource block in terraform configuration matching the remote resource or just empty resource block, then you can import the remote resource using `terraform import` by resource ID e.g. `terraform import aws_instance.foo i-abcd1234` where `i-abcd1234` is EC2 instance Id

Reference: https://developer.hashicorp.com/terraform/cli/commands/import

Q65. A user wants to list all resources which are deployed using Terraform. How can this be done?

- `terraform state show`
- `terraform state list`
- `terraform show`
- `terraform show list`

The `terraform state list` command is used to list resources within a Terraform state.

Reference: https://developer.hashicorp.com/terraform/cli/commands/state/list

Q66. Which terraform state subcommand will give you all of the resources in your state?

- list
- show
- refresh
- apply

The `terraform state list` command is used to list resources within a Terraform state.

Reference: https://developer.hashicorp.com/terraform/cli/commands/state/list

Q67. A user wants to see the resource block for resource `aws_instance` having name `foo` in state file. How can this be done?

- `terraform show aws_instance.foo`
- `terraform show aws_instance foo`
- `terraform state show aws_instance.foo`
- `terraform state show aws_instance foo`

The `terraform state show` command is used to show the attributes of a single resource in the Terraform state.

Reference: https://developer.hashicorp.com/terraform/cli/commands/state/show

Q68. Which of the following command provides the JSON reprentation of the state?

- `terraform state -json`
- `terraform state show -json`
- `terraform show -json`
- `terraform show state -json`

The `terraform show -json` command is used to provide human-readable JSON output from a state or plan file.

Reference: https://developer.hashicorp.com/terraform/cli/commands/show

Q69. Why would you use the `terraform taint` command?

- When you want to force Terraform to destroy a resource on the next apply
- When you want to force Terraform to destroy and recreate a resource on the next apply
- When you want Terraform to ignore a resource on the next apply
- When you want Terraform to destroy all the infrastructure in your workspace

The `terraform taint` command informs Terraform that a particular object has become degraded or damaged. Terraform will propose to replace it in the next plan you create.

Reference: https://developer.hashicorp.com/terraform/cli/commands/taint

Q70. The command `terraform.taint` is deprecated in v0.15.2, which command you should use intead?

- `terraform apply -replace`
- `terraform plan -replace`
- `terraform apply -taint`
- `terraform plan -taint`

The command `terraform taint` is deprecated and recommended to use `terraform apply -replace` to inform Terraform that a particular object has become degraded or damaged. Terraform will propose to replace it in the next plan you create.

Reference: https://developer.hashicorp.com/terraform/cli/commands/taint

Q71. You need Terraform to destroy and recreate a single database server that was deployed with a bunch of other resources. You don't want to modify the Terraform code. What command can be used to accomplish this task?

- `terraform apply -replace=aws_instance.database`
- `terraform apply -destroy=aws_instance.database`
- `terraform state recreate aws_instance.database`
- `terraform state destroy aws_instance.database`

The planning option `-replace=ADDRESS` instructs Terraform to plan to replace (destroy and recreate) the resource instance with the given address

Reference: https://developer.hashicorp.com/terraform/cli/commands/plan#planning-options

Q72. You have an EC2 instance that is acting up in the cloud. It handles a relatively light ephemeral workload, so it can be restarted/destroyed with no repercussions. What full command would you use to target only this instance for recreation?

- `terraform apply -replace=aws_instance.{INSTANCE_NAME}`
- `terraform apply -replace aws_instance`
- `terraform apply -replace {INSTANCE_NAME}`
- `terraform destroy --target=aws.instance{INSTANCE_NAME}` and `terraform apply`

The planning option `-replace=ADDRESS` instructs Terraform to plan to replace (destroy and recreate) the resource instance with the given address

Reference: https://developer.hashicorp.com/terraform/cli/commands/plan#replace-address

Q73. What is not processed when running a `terraform refresh` ?

- State file
- Configuration file
- Credentials
- Cloud provider

Reference: https://developer.hashicorp.com/terraform/cli/commands/refresh

Q74. Which of the following Terraform commands will automatically refresh the state unless supplied with additional flags or arguments? Choose *TWO* correct answers.

- `terraform plan`
- `terraform state`
- `terraform apply`
- `terraform validate`
- `terraform output`

Reference: https://developer.hashicorp.com/terraform/cli/commands/refresh

Q75. The command `terraform refresh` is deprecated in v0.15.4, which command is recommended to use instead? Choose *TWO* correct answers.

- `terraform apply -refresh-only`
- `terraform plan -refresh-only`
- `terraform apply -refresh`
- `terraform plan -refresh`

Reference: https://developer.hashicorp.com/terraform/cli/commands/refresh

Q76. Which of the following command will give you an opportunity to review the changes that Terraform has detected during refresh? Choose *TWO* correct answers.

- `terraform apply -refresh-only -auto-approve`
- `terraform apply -refresh-only`
- `terraform refresh`
- `terraform plan -refresh-only`

Reference: https://developer.hashicorp.com/terraform/cli/commands/refresh

Q77. What happens when you apply Terraform configuration? Choose *TWO* correct answers.

- Terraform makes any infrastructure changes defined in your configuration.
- Terraform gets the plugins that the configuration requires.
- Terraform updates the state file with any configuration changes it made.
- Terraform corrects formatting errors in your configuration.
- Terraform destroys and recreates all your infrastructure from scratch.

Q78. Which flag is used to find more information about a Terraform command? For example, you need additional information about how to use the `plan` command. You would type: `terraform plan _____` .

Type your answer in the field provided. The text field is not case-sensitive and all variations of the correct answer are accepted.

- `-h`
- `-help`
- `--help`

Answers that would also receive full credit:
`--h`
`terraform plan -h`
`terraform plan --h`
`terraform plan -help`
`terraform plan --help`
`terraform -h plan`
`terraform -help plan`
`terraform --help plan`
`plan -h`
`plan --h`
`plan -help`
`plan --help`
`-h plan`
`-help plan`
`--help plan`

Q79. Which flag would you add to terraform plan to save the execution plan to a file? You would type: `terraform plan _____`

Type your answer in the field provided. The text field is not case-sensitive and all variations of the correct answer are accepted.

- `-out=FILENAME`

The command `terraform plan -out=dev.tfplan` saves the plan to `dev.tfplan` file that you can later pass to command `terraform apply dev.tfplan` for execution. Typical convention is to use `.tfplan` file extension to save plan file.
Reference: https://developer.hashicorp.com/terraform/cli/commands/plan#out-filename

Q80. You just added a new set of resources to your configuration and would only like to see them when you run your `terraform plan` command. What flag do you specify when running the `terraform plan` command to only see their plans?

- `-target={resources}`
- `-refresh=true`
- `-state={new_state_file}`
- `-lock=true`

You can use the `-target` option to focus Terraform's attention on only a subset of resources. You can use resource address syntax to specify the constraint
Reference: https://developer.hashicorp.com/terraform/cli/commands/plan#resource-targeting

Q81. 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 remove the VM from state file
- Terraform will report an error
- Terraform will not make any changes
- Terraform will recreate the VM

Q82. You have multiple team members collaborating on infrastructure as code (IaC) using Terraform, and want to apply formatting standards for readability.How can you format Terraform HCL (HashiCorp Configuration Language) code according to standard Terraform style convention?

- Run the `terraform fmt` command during the code linting phase of your CI/CD process
- Designate one person in each team to review and format everyone's code
- Manually apply two spaces indentation and align equal sign "=" characters in every Terraform file (*.tf)
- Write a shell script to transform Terraform files using tools such as AWK, Python, and sed

Q83. You have deployed a new webapp with a public IP address on a clod 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?

- Run `terraform output ip_address` to view the result
- 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
- Run terraform state list to find the name of the resource, then terraform state show to find the attributes including public IP address
- Run terraform destroy then terraform apply and look for the IP address in stdout

Reference: https://developer.hashicorp.com/terraform/cli/commands/output

Q84. In order to reduce the time it takes to provision resources, Terraform uses parallelism. By default, how many resources will Terraform provision concurrently during a `terraform apply`?

- 100
- 10
- 5
- 1

The command `terraform apply -parallelism=20` limits the number of concurrent operation to 20 as Terraform walks the graph. default is 10.

Reference: https://developer.hashicorp.com/terraform/cli/commands/apply#parallelism-n

Q85. Say you wanted to increase the number of operations that terraform is concurrently using to create your resources. Which command would you run, with what specific flag, to accomplish this? (Choose 2 answers)

- `terraform apply`
- `-parallelism={NUMBER-OF-OPERATIONS}`
- `terraform init`
- `-concurrent={NUMBER-OF-OPERATIONS}`

The command `terraform apply -parallelism=20` limits the number of concurrent operation to 20 as Terraform walks the graph. default is 10.

Reference: https://developer.hashicorp.com/terraform/cli/commands/apply#parallelism-n

Q86. Lately you noticed that your Terraform jobs are failing in your CI/CD pipeline. The error that is coming back mentions something about hitting a rate limit. Without altering the time that the builds are ran, what could you pass into the terraform apply command to slow your operations down?

- `-parallelism={NUMBER_OF_OPERATIONS}`
- `-concurrent={NUMBER_OF_OPERATIONS}`
- `-rate-limit={NUMBER_OF_OPERATIONS}`
- `-refresh=false`

Reference: https://developer.hashicorp.com/terraform/cli/commands/apply#parallelism-n

Q87. What Terraform command can be used to remove the lock on the state for the current configuration?

- `terraform unlock`
- `terraform force-unlock`
- Removing the lock on the state file is not possible
- `terrafom state unlock`

The command `terraform force-unlock` Manually unlock the state for the defined configuration.

Reference: https://developer.hashicorp.com/terraform/cli/commands/force-unlock

Terraform Backend

Q88. What does the default `local` Terraform backend store?

- `*.tfplan` files
- Terraform binary
- Provider plugins
- `terraform.tfstate` file

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/local

Q89. What two configuration variables are available to a default local backend? (Choose 2 answers)

- `path`
- `workspace_dir`
- `working_dir`
- `path_dest`

The `path` and `workspace_dir` are two optional configuration supported by local backend

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/local#configuration-variables

Q90. What is NOT true about the Terraform backend?

- A backend is where Terraform stores its state data files.
- By default, Terraform uses a backend called local, which stores state as a local file on disk.
- A terraform configuration can only provide one backend block.
- A backend block can refer to named values (like input variables, locals, or data source attributes).

A backend block cannot refer to named values (like input variables, locals, or data source attributes).

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration#using-a-backend-block

Q91. How is the Terraform `remote` backend different than other state backends such as `s3`, `http` and `consul`, etc.?

- It can execute Terraform runs on dedicated infrastructure on premises or in Terraform Cloud
- It doesn't show the output of a terraform apply locally
- It is only available to enterprise customers
- All of the above

The remote backend is unique among all other Terraform backends because it can both store state snapshots and execute operations for Terraform Cloud's CLI-driven run workflow. It used to be called an "enhanced" backend

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/remote

Q92. How do you supply remaining arguments to a partial backend configuration? (Choose 2 answers)

- Specify file `terraform init -backend-config=PATH`
- Specify key/value pairs `terraform init -backend-config="KEY=VALUE"`
- Environment variable `export TF_VAR_key=value`
- Set variable `terraform init -var="KEY=VALUE"`

A backend block cannot refer to Environment or Input variables, you can supply the arguments through `-backend-config` option in `init` command

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration#partial-configuration

Q93. You are a part of a growing Cloud Infrastructure team. Your boss asks you to transition the team off of local backends, and onto remote backends. Within Terraform, what do you do to use the S3 buckets as a remote backend? (Choose 2 answers)

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
```

```
    region = "us-east-1"
  }
}
```

- [ ] Specify the key to store state file inside the S3 bucket
- [ ] Make sure Terraform gets AWS IAM permission on target backend bucket and stored state file
- [ ] Export your AWS API key to TF_BACKEND_KEY
- [ ] Encrypt your AWS buckets with SSE.

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/s3#example-configuration

Q94. Which of the following is a type of backend configurable in Terraform?

- [ ] local
- [ ] standard
- [ ] enhanced
- [ ] advanced

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration#backend-types

Q95. Which of the following is NOT a supported backend type?

- [ ] consul
- [ ] github
- [ ] local
- [ ] s3

Terraform v1.4.x supports the following backend types:- local, remote, azurerm, consul, cos, gcs, http, kubernetes, oss, pg, s3
Terraform v1.2.x also supports following backend types:- artifactory, etcd, etcdv3, manta, swift

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration

Q96. All standard backend types support remote state storage, state locking, and encryption at rest?

- [ ] true
- [ ] false

No, All the standard backend doesn't support all three. `local` *backend type doesn't support remote state storage,* `artifactory` *and* `etcd` *backend types doesn't support state locking.*

Q97. Which of the following backend type doesn't support state locking?

- [ ] local
- [ ] s3
- [ ] remote
- [ ] artifactory

Backend types support state locking:- local, remote, azurerm, consul, cos, gcs, http, kubernetes, oss, pg, s3, etcdv3, manta, swift
Backend types doesn't support state locking:- artifactory, etcd

Q98. Which of the following backend type doesn't support remote state storage?

- [ ] remote
- [ ] Terraform Cloud
- [ ] github
- [ ] artifactory

Github is not a valid backend type supported by Terraform

Q99. Your co-worker has decided to migrate Terraform state to a remote backend. They configure Terraform with the backend configuration, including the type, location, and credentials. However, you want to better secure this configuration. Rather than storing them in plaintext, where should you store the credentials? (select two)

- [ ] use a variable
- [ ] credentials file
- [ ] on the remote system
- [ ] environment variables

Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration#credentials-and-sensitive-data

Q100. You can migrate the Terraform backend but only if there are no resources currently being managed.

- [ ] false
- [ ] true

Terraform will automatically detect any changes in your configuration/backend and request a reinitialization. As part of the reinitialization process, Terraform will ask if you'd like to migrate your existing state to the new configuration. This allows you to easily switch from one backend to another.
Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration#changing-configuration

Q101. You have decided to migrate the Terraform state to a remote s3 backend. You have added the `backend` block in the Terraform configuration. Which command you should run to migrate the state?

```
terraform {
  backend "s3" {
    bucket = "terraform-s3-bucket-name"
    key    = "s3 key path"
    region = "us-west-1"
  }
}
```

- [ ] `terraform init`
- [ ] `terraform push`
- [ ] `terraform apply`
- [ ] `terraform plan`

When you change a backend's configuration, you must run terraform init again to validate and configure the backend before you can perform any plans, applies, or state operations.
Reference: https://developer.hashicorp.com/terraform/language/settings/backends/configuration#initialization

Terraform Provisioners

Q102. Provisioners should only be used as a last resort.

- [ ] true
- [ ] false

Provisioners should only be used as a last resort. They add a considerable amount of complexity and uncertainty to Terraform usage. For most common situations there are better alternatives available.
Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax#provisioners-are-a-last-resort

Q103. You want to use a Terraform provisioner to execute a script on the remote machine. What block type would use to declare the provisioner?

- `terraform` block
- `data` block
- `provider` block
- `resource` block

You can add a provisioner block inside the resource block of a compute instance for e.g. below provisioner will be executed when the `aws_instance` resource is built.

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command = "echo The server's IP address is ${self.private_ip}"
  }
}
```

Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax#how-to-use-provisioners

Q104. Which option will you use to run provisioners that are not associated with any resources?

- null_resource
- file
- local-exec
- remote-exec

null_resource has been renamed to terraform_data in Terraform v1.4.x and later version
Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/null_resource

Q105. Which provisioner copies files or directories from the machine running Terraform to the newly created resource?

- null_resource
- file
- local-exec
- remote-exec

Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/file

Q106. Which type of connections supported by file provisioner? Select all valid options.

- ssh
- sftp
- winrm
- rdc

Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/file

Q107. Which provisioner invokes a process on the machine running Terraform, not on the resource?

- null_resource
- file
- local-exec
- remote-exec

Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/local-exec

Q108. Where does the 'local-exec' provisioner execute its code provided in its block?

- On the remote resource specified.
- On the local machine running terraform.
- On a spot-instance on your cloud provider.
- In a container on your machine provided by the Terraform binary.

The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource.
Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/local-exec

Q109. Which provisioner invokes a process on the resource created by Terraform?

- null_resource
- file
- local-exec
- remote-exec

Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/remote-exec

Q110. What are the two accepted values for provisioners that have the "on_failure" key specified? (Choose 2 answers)

- continue
- fail
- abort
- retry

By default, provisioners that fail will also cause the Terraform apply itself to fail. The on_failure setting can be used to change this. The allowed values are: continue and fail
Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax#failure-behavior

Q111. What does the following provisioner block specify?

```
provisioner "local-exec" {
  when    = destroy
  command = "echo 'Destroy-time provisioner'"
}
```

- Before the resource is destroyed, the provisioner will invoke "echo 'Destroy-time provisioner'"
- If the resource receives a 'destroy' command locally, it will echo 'Destroy-time provisioner'
- After the resource is destroyed, it will invoke "echo 'Destroy-time provisioner'"
- On the next 'terraform apply' the resource will be destroyed

Destroy provisioners are run before the resource is destroyed. If they fail, Terraform will error and rerun the provisioners again on the next terraform apply.
Reference: https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax#destroy-time-provisioners

Terraform Providers

Q112. Which of the following best describes a Terraform provider?

- A collection of resources that can be used to define a specific piece of infrastructure
- A plugin that allows Terraform to interact with a specific cloud provider or service
- A tool for managing Docker containers
- A set of variables used to configure Terraform resources

Reference: https://developer.hashicorp.com/terraform/language/providers

Q113. Which of the following is NOT true of Terraform providers?

- Providers can be written by individuals
- Providers can be maintained by a community of users
- Some providers are maintained by HashiCorp
- Major cloud vendors and non-cloud vendors can write, maintain, or collaborate on Terraform providers
- None of the above

Q114. A provider configuration block is required in every Terraform configuration.

```
provider "provider_name" {
  ...
}
```

- true
- false

Q115. Official Terraform providers are owned and maintained by HashiCorp.

- true
- false

1. Hashicorp Official providers and modules are owned and maintained by HashiCorp. Namespace = `hashicorp`
2. Partner providers and modules are owned and maintained by a technology company that has gone through our Terraform Integration Program and maintains a partnership with HashiCorp. Namespace e.g. `mongodb/mongodbatlas`
3. Anyone can publish and share a provider by signing into the Registry using their GitHub account and following a few additional steps Reference: https://developer.hashicorp.com/terraform/docs/partnerships#terraform-provider-integrations

Q116. Which provider configuration can be used to define multiple aws provider with different regions?

- provider
- source
- region
- alias

Reference: https://developer.hashicorp.com/terraform/language/providers/configuration#alias-multiple-provider-configurations

Q117. What is a provider block without an `alias` meta argument?

- The default provider configuration.
- A broken provider configuration.
- A partial provider configuration.
- There must be an alias meta argument.

Reference: https://developer.hashicorp.com/terraform/language/providers/configuration#default-provider-configurations

Q118. How do you select the alternate aws provider for us-west-2 region?

```
# The default provider configuration
provider "aws" {
  region = "us-east-1"
}
# Additional provider configuration for west coast region
provider "aws" {
  alias  = "west"
  region = "us-west-2"
}
```

- `resource "aws_instance" "foo" { provider = aws }`
- `resource "aws_instance" "foo" { provider = aws.west }`
- `resource "aws_instance" "foo" { provider = aws.us-west-2 }`
- `resource "aws_instance" "foo" { provider = west }`

When Terraform needs the name of an alternare provider configuration, it expects a reference of the form `<PROVIDER NAME>.<ALIAS>`
Reference: https://developer.hashicorp.com/terraform/language/providers/configuration

Q119. Terraform uses a lock file to ensure predictable runs when using ambiguous provider version constraints. How do you update the lock file?

- `terraform providers lock`
- `terraform lock`
- `terraform apply lock`
- `terraform lock provider -provider={PROVIDER_NAME}`

The `terraform providers lock` *will analyze the configuration in the current working directory to find all of the providers it depends on, and it will fetch the necessary data about those providers from their origin registries and then update the dependency lock file to include a selected version for each provider and all of the package checksums that are covered by the provider developer's cryptographic signature.*
Reference: https://developer.hashicorp.com/terraform/cli/commands/providers/lock

Q120. Terraform is running in an isolated network without access to Terraform registry. How can you configure Terraform to consult only a local filesystem mirror to download plugins?

- `terraform providers mirror`
- `terraform mirror`
- `terraform providers local`
- `terraform plugins mirror`

Reference: https://developer.hashicorp.com/terraform/cli/commands/providers/mirror

Q121. In the terraform block, which configuration would be used to identify the specific version of a provider required?

- `required_providers`
- `required_provider`
- `required_versions`
- `required_version`

Each Terraform module must declare which providers it requires, so that Terraform can install and use them. Provider requirements are declared in a `required_providers` block.

```
terraform {
  required_providers {
    mycloud = {
      source  = "mycorp/mycloud"
      version = "~> 1.0"
    }
  }
}
```

Reference: **https://developer.hashicorp.com/terraform/language/providers/requirements#requiring-providers**

Q122. Which of the following is true for installing provider when `terraform init` command runs?

- If any acceptable versions are installed, Terraform uses the newest installed version that meets the constraint (even if the Terraform Registry has a newer acceptable version)
- If no acceptable versions are installed and the plugin is one of the providers distributed by HashiCorp, Terraform downloads the newest acceptable version from the Terraform Registry and saves it in a subdirectory under `.terraform/providers/`
- If no acceptable versions are installed and the plugin is not distributed in the Terraform Registry, initialization fails and the user must manually install an appropriate version.
- All of the above

Reference: **https://developer.hashicorp.com/terraform/plugin/how-terraform-works#selecting-plugins**

Terraform Resources

Q123. Who is the provider for the below resource?

```
resource "aws_vpc" "main" {
    name = "test"
}
```

- vpc
- main
- aws
- test

Resource type must always start with their containing provider's name followed by an underscore, so a resource type from the provider `aws` might be named `aws_vpc`.

Reference: **https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc**

Q124. What is the name assigned by Terraform to reference this resource?

```
resource "google_computer_instance" "main" {
    name = "test"
}
```

- computer_instance
- main
- google
- test

Q125. Examine the following Terraform configuration, which uses the data source for an AWS AMI. What value should you enter for the ami argument in the AWS instance resource?

```
data "aws_ami" "ubuntu" {
  ...
}

resource "aws_instance" "web" {
  ami           = _____________________
  instance_type = "t3.micro"
}
```

- aws_ami.ubuntu
- data.aws_ami.ubuntu
- data.aws_ami.ubuntu.id
- aws_ami.ubuntu.id

Data source attributes can be used in other resources using expression `data.<TYPE>.<NAME>.<ATTRIBUTE>` i.e. `data.aws_ami.ubuntu.id`

Reference: **https://developer.hashicorp.com/terraform/language/data-sources#description**

Q126. What's the correct syntax for referencing a resource within the configuration file?

- `<RESOURCE TYPE>.<NAME>`
- `<NAME>.<RESOURCE TYPE>`
- `<PROVIDER>.<RESOURCE TYPE>`
- `<LOCAL/REMOTE STATE>.<RESOURCE TYPE>`

Reference: **https://developer.hashicorp.com/terraform/cli/state/resource-addressing#resource-spec**

Q127. From the code below, identify the implicit dependency:

```
resource "aws_eip" "public_ip" {
    vpc = true
    instance = aws_instance.web_server.id
}

resource "aws_instance" "web_server" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
  depends_on = [aws_s3_bucket.company_data]
}
```

- AMI with an id of `ami-2757f631`
- S3 Bucket labeled `company_data`
- Instance Type `t2.micro`
- EC instance labeled `web-server`

EC2 instance must exist before the Elastic IP can be created and attached, Terraform handled this automatically as part of implicit dependency

Reference: https://developer.hashicorp.com/terraform/tutorials/configuration-language/dependencies#manage-implicit-dependencies

Q128. In the example below, the `depends_on` argument creates what type of dependency?

```
resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
  depends_on = [aws_s3_bucket.company_data]
}
```

- non-dependency resource
- implicit dependency
- explicit dependency
- internal dependency

Reference: https://developer.hashicorp.com/terraform/tutorials/configuration-language/dependencies#manage-explicit-dependencies

Q129. What is the syntax to correctly reference a data source?

- `data.<DATA TYPE>.<NAME>`
- `data.<NAME>`
- `data.<NAME>.<DATA TYPE>`
- `<DATA TYPE>.<NAME>.data`

`data.<DATA TYPE>.<NAME>` is an object representing a data resource of the given data source type and name.

Reference: https://developer.hashicorp.com/terraform/language/expressions/references#data-sources

Q130. You want to use the `terraform state show` to see the attributes of a single resource created by the `for_each` in below resource block. What resource address should be used for the instance related to `vault` ?

```
resource "aws_instance" "demo" {
  # ...
  for_each = {
    "terraform": "infrastructure",
    "vault":     "security",
    "consul":    "connectivity",
    "nomad":     "scheduler",
  }
}
```

- `aws_instance.demo[1]`
- `aws_instance.demo["2"]`
- `aws_instance.demo.vault`
- `aws_instance.demo["vault"]`

Reference: https://developer.hashicorp.com/terraform/cli/state/resource-addressing#for_each-example

Q131. How can you obtain a list of all of the `device_name` values from `ebs_block_device` nested blocks, that are created by this `resource` block?

```
resource "aws_instance" "example" {
  ami           = "ami-abc123"
  instance_type = "t2.micro"

  ebs_block_device {
    device_name = "sda2"
    volume_size = 16
  }
  ebs_block_device {
    device_name = "sda3"
    volume_size = 20
  }
}
```

- `aws_instance.example.ebs_block_device[*].device_name`
- `aws_instance.example.ebs_block_device[0,1].device_name`
- `aws_instance.example.*.device_name`
- `aws_instance.*.*.device_name`

The splat expression special `[*]` iterates over all of the elements of the list given to its left and accesses from each one the attribute name given on its right

Reference: https://developer.hashicorp.com/terraform/language/expressions/references#references-to-resource-attributes

Reference: https://developer.hashicorp.com/terraform/language/expressions/splat

Q132. Terraform can only manage dependencies between resources if the `depends_on` argument is explicitly set for the dependent resources.

- true
- false

Terraform automatically infers when one resource depends on another by studying the resource attributes used in interpolation expressions. Terraform uses this dependency information to determine the correct order in which to create the different resources

Reference: https://developer.hashicorp.com/terraform/tutorials/configuration-language/dependencies

Terraform Variables and Outputs

Q133. How can you set the value to a variable "region" declared in the configuration file?

- Using command line `terraform apply -var="region=us-east-1"`
- Using variable file `terraform apply -var-file="variables.tfvars"` where the file contains: region=us-east-1
- Using environment variable `export TF_VAR_region=us-east-1`
- All of the above

Reference: https://developer.hashicorp.com/terraform/language/values/variables

Q134. Which one of the following takes higher precedence in loading variable in Terraform?

- Command line flag - `terraform apply -var="region=us-east-1"`
- Configuration file - set in your `terraform.tfvars` file
- Environment variable - `export TF_VAR_region=us-east-1`
- Default Config - default value in `variables.tf`

Reference: https://developer.hashicorp.com/terraform/language/values/variables

Q135. Which of the following is an invalid argument for defining input variable in Terraform?

- default
- type
- description
- validation
- sensitive
- nullable
- depends_on

depends_on is an optional argument for declaring output value, not for declaring input variable

Reference: https://developer.hashicorp.com/terraform/language/values/variables

Q136. How would you configure your input variable to fallback to a pre-declared value in your variable block?

- By specifying the default meta-argument.
- By specifying the fallback meta-argument.
- Terraform has a list of fallbacks that it will always implement if nothing is specified. E.g. aws_instance will fall back to a t2.micro if the size is not specifed.
- Terraform will ask you to set a fallback when you run the terraform apply command.

Reference: https://developer.hashicorp.com/terraform/language/values/variables#default-values

Q137. You defined a variable and would like to reference it in your terraform configuration file. What is the syntax required to do so?

- `var.<VARIABLE_NAME>`
- `<VARIABLE_NAME>.var`
- `var.<VARIABLE_NAME>.<RESOURCE_NAME>`
- `<RESOURCE_NAME>.var.<VARIABLE_NAME>`

Reference: https://developer.hashicorp.com/terraform/language/values/variables#using-input-variable-values

Q138. Consider the following configuration snippet: How would you define the `cidr_block` for `us-east-1` in the aws_vpc resource using a variable?

```
variable "vpc_cidrs" {
  type = map
  default = {
    us-east-1 = "10.0.0.0/16"
    us-east-2 = "10.1.0.0/16"
    us-west-1 = "10.2.0.0/16"
    us-west-2 = "10.3.0.0/16"
  }
}

resource "aws_vpc" "shared" {
  cidr_block = ____________
}
```

- `var.vpc_cidrs["us-east-1"]`
- `var.vpc_cidrs.0`
- `vpc_cidrs["us-east-1"]`
- `var.vpc_cidrs[0]`

Variable of type map values are referenced using key e.g. `us-east-1`

Q139. A new variable `fruits` has been created of type `list` as shown below. How would you reference `banana` in your configuration?

```
variable "fruits" {
    type = list(string)
    default = [
        "mango",
        "apple",
        "banana",
        "orange",
        "grapes"
    ]
}
```

- `var.fruits[2]`
- `var.fruits.banana`
- `var.list.fruits[2]`
- `var.fruits[3]`

Variable of type list values are referenced using index that start with 0

Q140. A Terraform local value can reference other Terraform local values?

- true
- false

Reference: https://developer.hashicorp.com/terraform/language/values/locals

Q141. When are output variables ran and sent to stdout?

- Only with terraform apply.
- Only on terraform plan or apply.
- With any terraform command.
- Only if you specify the -outputs flag on apply.

Reference: https://developer.hashicorp.com/terraform/language/values/outputs

Q142. You want to know from which paths Terraform is loading providers referenced in your Terraform configuration (*.tf files). You need to enable detailed logging to find this out. Which of the following would achieve this?

- Set the environment variable TF_LOG=TRACE
- Set the environment variable TF_INPUT=1

- Set the environment variable TF_VAR_LOG=TRACE
- Set the environment variable TF_LOG_PATH=./terraform.log

You can set TF_LOG to one of the log levels TRACE, DEBUG, INFO, WARN or ERROR to change the verbosity of the logs. TRACE is most verbose logging

Reference: https://developer.hashicorp.com/terraform/cli/config/environment-variables#tf_log

Q143. After running into issues with Terraform, you need to enable verbose logging to assist with troubleshooting the error. Which of the following values provides the MOST verbose logging?

- TRACE
- DEBUG
- WARN
- INFO

Reference: https://developer.hashicorp.com/terraform/internals/debugging

Q144. You are required to setup Terraform logs. Your boss asks you to make sure they always end up in one location such that they can be collected, and that they be set to the informational level. How would you accomplish this? (Choose 2 answers)

- Export the environment variable of TF_LOG to be INFO
- Export the TF_LOG_PATH environment variable to the requested path location.
- Only invoke the terraform apply command in the location your boss wants the logs, because terraform automatically saves a .log file in the working directory.
- Export the TF_PATH_LOG environment variable to the requested path location.

Reference: https://developer.hashicorp.com/terraform/cli/config/environment-variables

Q145. You have a Terraform variable that is declared as follows:

```
variable "num" {
 default = 3
}
```

You have also defined the following environment variables in your BASH shell:-

```
export TF_VAR_num=10
```

You also have a *terraform.tfvars* file with the following contents:-

```
num = 7
```

When you run the following apply command, what is the value assigned to the num variable?

```
terraform apply -var num=4
```

- 4
- 7
- 3
- 10

Q146. Which of the following is a valid variable name in Terraform?

- 1234
- 1_aws_vpc
- invalid
- count

The name of a variable can be any valid identifier except the following: source, version, providers, count, for_each, lifecycle, depends_on, locals. Valid Identifiers can contain letters, digits, underscores (_), and hyphens (-). The first character of an identifier must not be a digit, to avoid ambiguity with literal numbers.

Reference: https://developer.hashicorp.com/terraform/language/values/variables#declaring-an-input-variable

Q147. What are Data Sources in terraform?

- Data to be fetched or computed for use elsewhere in terraform configuration.
- Similar to resources, they specify data to be created in the corresponding provider.
- A binary set of operators that tell resources how to behave with certain meta-arguments.
- Data sources are a way for terraform to keep track of all resources created in the provider's infrastructure.

Reference: https://developer.hashicorp.com/terraform/language/data-sources

Terraform Module

Q148. Which of the following best describes a Terraform module?

- A collection of resources that make up a specific piece of infrastructure
- A plugin that allows Terraform to interact with a specific cloud provider or service
- A set of variables used to configure Terraform resources
- A tool for managing Docker containers

Reference: https://developer.hashicorp.com/terraform/language/modules

Q149. In Terraform, what is a module?

- A group of related resources
- A singular, non-abstractive, resource.
- Essentially a comment, it doesn't do anything except to describe a set of resources.
- Similar to programming functions, modules are used to write code in Golang for direct interaction with Terraform.

Reference: https://developer.hashicorp.com/terraform/language/modules

Q150. In Terraform, What are modules used for?

- Organize configuration
- Encapsulate configuration
- Re-use configuration
- All of the above

Reference: https://developer.hashicorp.com/terraform/tutorials/modules/module#what-are-modules-for

Q151. When initializing Terraform, you notice that Terraform's CLI output states it is downloading the modules referenced in your code. Where does Terraform cache these modules?

- in the `/temp` directory on the machine executing Terraform
- in a `/modules` directory in the current working directory
- in the `/downloads` directory for the user running the terraform init
- in the `.terraform/modules` subdirectory in the current working directory

Reference: https://developer.hashicorp.com/terraform/tutorials/modules/module-create

Q152. Which one of the following is the required argument for calling a child module?

- version
- source
- providers
- depends_on

The `source` argument is mandatory for calling a child module. The `version` argument is recommended for modules from a registry. You can use following meta-arguments for modules: `count`, `for_each`, `providers`, and `depends_on`.

Reference: https://developer.hashicorp.com/terraform/language/modules/syntax#calling-a-child-module

Q153. What are three meta-arguments, along with `source` and `version`, that a module can use? (Choose 3 answers)

- `for_each`
- `count`
- `max`
- `depends_on`

Terraform module has following optional meta-arguments: `count`, `for_each`, `providers`, and `depends_on`.

Reference: https://developer.hashicorp.com/terraform/language/modules/syntax#meta-arguments

Q154. A module that has been called by another module is often referred to as a child module. Where is the child module stored in below module block?

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name = "my-vpc"
  cidr = "10.0.0.0/16"

  azs             = ["eu-west-1a", "eu-west-1b", "eu-west-1c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]
}
```

- in a local directory named `.terraform/terraform-aws-modules/vpc/aws`
- in a remote code repository
- in terraform cloud private module registry
- in terraform public module registry

terraform-aws-modules/vpc/aws is a public module registry which creates VPC resources on AWS

Reference: https://registry.terraform.io/browse/modules

Q155. Content of a file named `main.tf` is shown below. Which of the following statements are true about this code? (select two)

```
module "servers" {
  source = "./app-cluster"

  servers = 5
}
```

- `app-cluster` is the child module
- `app-cluster` is the calling module or parent module
- `main.tf` is the child module
- `main.tf` is the calling module or parent module

Reference: https://developer.hashicorp.com/terraform/language/modules/syntax#calling-a-child-module

Q156. Which one of the following is a valid source type to download the source code of a module?

- Local Paths
- Terraform Registry
- Github
- Bitbucket
- HTTP URLs
- S3 buckets
- All of the above

Reference: https://developer.hashicorp.com/terraform/language/modules/sources

Q157. Which one of the following file extension recognized by terraform while fetching archived module over HTTP?

- zip
- tar.bz2 and tbz2
- tar.gz and tgz
- tar.xz and txz
- All of the above

Reference: https://developer.hashicorp.com/terraform/language/modules/sources#fetching-archives-over-http

Q158. How do you download a module configured in your Terraform code?

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

- `terraform get module consul`
- `terraform install modules consul`
- `terraform init`
- `terraform module init`

Reference: https://developer.hashicorp.com/terraform/language/modules/sources

Q159. What feature of Terraform Cloud allows you to publish and maintain a set of custom modules which can be used within your organization?

- remote runs
- terraform registry
- private module registry
- custom VCS integration

Reference: https://developer.hashicorp.com/terraform/cloud-docs/registry#private-providers-and-modules

Q160. How do you correctly reference a private registry module source?

- `<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>`
- `<NAMESPACE>/<NAME>/<PROVIDER>`
- `<HOSTNAME>/<NAMESPACE>/<PROVIDER>`
- `<NAMESPACE>/<NAME>/<PROVIDER>/<HOSTNAME>`

When specifying a source for a private registry, the correct Syntax is `<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>` e.g. `app.terraform.io/example_corp/vpc/aws` . It is different than the public registry because it includes the `<HOSTNAME>` field.

Reference: https://developer.hashicorp.com/terraform/registry/modules/use#private-registry-module-sources

Q161. How do you reference module source from public terraform registry?

- `<NAMESPACE>/<NAME>/<PROVIDER>`
- `<NAMESPACE>/<PROVIDER>/<NAME>`
- `<NAMESPACE>/<PROVIDER>`
- `<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>`

Let's look at the example of referencing module from public terraform registry:-

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

Reference: https://developer.hashicorp.com/terraform/language/modules/sources#terraform-registry

Q162. When specifying a module, what is the best practice for the implementation of the meta-argument version?

- The best practice is to explicitly set the version argument as a version constraint string from the Terraform registry.
- The best practice is to use no version and accept the latest version.
- The best practice is to download the module, place it in your working directory, then source that module, and specify the version that was downloaded.
- The best practice is to always ensure you append beta to the end of the version. This allows you and your team to always be working on the latest and greatest features for that module.

Reference: https://developer.hashicorp.com/terraform/language/modules/syntax#version

Q163. How do you access module attributes?

- Through the child module, by declaring an output value to selectively export certain values to be accessed by the calling module.
- Through the parent module, by declaring an output value to selectively export certain values to be accessed by the calling module.
- By specifying the outputs block.
- When apply is ran, you must pass in -resource-output={ATTRIBUTE.NAME}.

The resources defined in a module are encapsulated, so the calling module cannot access their attributes directly. However, the child module can declare output values to selectively export certain values to be accessed by the calling module.

Reference: https://developer.hashicorp.com/terraform/language/modules/syntax#accessing-module-output-values

Q164. Who can publish and share modules on the Terraform Registry?

- Anyone
- Only specific providers
- Those who have passed the Hashicorp Terraform Associate exam
- Only those who have contributed to Open Source Terraform

Anyone can publish and share modules on the Terraform Registry.

Reference: https://developer.hashicorp.com/terraform/registry/modules/publish

Q165. What are some of the requirements that must be met in order to publish a module on the Terraform Public Module Registry? (select three)

- The module must be PCI/HIPPA compliant.
- The module must be on GitHub and must be a public repo.
- To publish a module initially, at least one release tag must be present e.g. `v1.0.4` and `0.9.2`
- Module repositories must use this three-part name format, `terraform-<PROVIDER>-<NAME>` e.g. `terraform-google-vault`

Reference: https://developer.hashicorp.com/terraform/registry/modules/publish#requirements

Q166. What are some of the requirements for publishing Private Modules to the Terraform Cloud Private Registry? (select three)

- The module must be PCI/HIPPA compliant.
- The module must be on your configured VCS providers, and Terraform Cloud's VCS user account must have admin access to the repository
- The module must adhere to the standard module structure
- Module repositories must use this three-part name format, `terraform-<PROVIDER>-<NAME>` e.g. `terraform-google-vault`

The requirements for Publishing to Terraform Cloud Private Registry is same as publishing to Terraform Public Registry except that module repository can be on your configured VCS providers in case of private registry whereas it must be public Github repo in case of public registry

Reference: https://developer.hashicorp.com/terraform/cloud-docs/registry/publish-modules

Q167. In a parent module, outputs of child modules are available in expressions as?

- `module.<MODULE NAME>.<OUTPUT NAME>`
- `<MODULE NAME>.<OUTPUT NAME>`
- `module.<OUTPUT NAME>`
- `output.<MODULE NAME>.<OUTPUT NAME>`

In a parent module, outputs of child modules are available in expressions as: `module.<MODULE NAME>.<OUTPUT NAME>`

Reference: https://developer.hashicorp.com/terraform/language/values/outputs#accessing-child-module-outputs

Q168. You have a module named `prod_subnet` that outputs the `subnet_id` of the subnet created by the module. How would you reference the subnet ID when using it for an input of another module?

- `subnet = module.outputs.prod_subnet.subnet_id`
- `subnet = module.prod_subnet.subnet_id`
- `subnet = prod_subnet.subnet_id`
- `subnet = prod_subnet.outputs.subnet_id`

In a parent module, outputs of child modules are available in expressions as: `module.<MODULE NAME>.<OUTPUT NAME>`

Reference: https://developer.hashicorp.com/terraform/language/values/outputs#accessing-child-module-outputs

Terraform Security

Q169. You want to ensure that your S3 buckets provisioned by Terraform are securely encrypted. What is the best way to achieve this?

- Create a Git hook that checks if the encryption parameter is enabled.
- Use AWS KMS to store a security key.
- Create a lambda function triggered on a "create bucket CloudTrail" event.
- Create a security policy using Sentinel policies.

Reference: https://developer.hashicorp.com/terraform/cloud-docs/policy-enforcement/sentinel

Q170. Which of the following allows Terraform users to apply policy as code to enforce standardized configurations for resources being deployed via infrastructure as code?

- Resources
- Functions
- Sentinel
- Workspaces

Reference: https://www.hashicorp.com/sentinel

Q171. HashiCorp Sentinel is a(n) ____ framework.

- platform as a service
- function as a service
- infrastructure as code
- policy as code

Reference: https://www.hashicorp.com/sentinel

Q172. Terraform Cloud provides imports to define Sentinel Policy Rules. Which of the following is not a valid import?

- `tfplan`
- `tfconfig`
- `tfstate`
- `tfapply`

Terraform Cloud provides four imports to define policy rules for the plan, configuration, state, and run associated with a policy check. They are: tfplan, tfconfig, tfstate, and tfrun

Reference: https://developer.hashicorp.com/terraform/cloud-docs/policy-enforcement/sentinel#sentinel-imports

Q173. You want to create a sentinel policy to ensure that naming convention is being followed in Terraform Configuration as per organization-wide standard. Which sentinel import can be used to access Terraform Configuration?

- `tfplan`
- `tfconfig`
- `tfstate`
- `tflan`

The `tfconfig` *import provides access to a Terraform configuration. Use cases of* `tfconfig` *import includes Organizational naming conventions, Required inputs and outputs, Enforcing particular modules, and Enforcing particular providers or resources.*

Reference: https://developer.hashicorp.com/terraform/cloud-docs/policy-enforcement/sentinel/import/tfconfig

Q174. Which is NOT a valid sentinel policy enforcement level?

- advisory
- soft mandatory
- warning
- hard mandatory

You can set an enforcement level for each policy that determines what happens when a Terraform plan does not pass the policy rule. Sentinel provides three policy enforcement levels: advisory, soft mandatory, and hard mandatory.

Reference: https://developer.hashicorp.com/terraform/cloud-docs/policy-enforcement/manage-policy-sets#policy-enforcement-levels

Q175. You have enabled Sentinel Policy in Terraform Cloud. When Terraform Cloud evaluates policies?

- On every Terraform Run
- After successful `terraform plan`
- During `terraform plan`
- Before `terraform apply`

Reference: https://developer.hashicorp.com/terraform/cloud-docs/run/states#the-sentinel-policy-check-stage

Q176. Your security team scanned some Terraform workspaces and found secrets stored in a plaintext in state files. How can you protect sensitive data stored in Terraform state files?

- Delete the state file every time you run Terraform
- Store the state in an encrypted backend
- Edit your state file to scrub out the sensitive data
- Always store your secrets in a secrets.tfvars file.

Reference: https://developer.hashicorp.com/terraform/language/state/sensitive-data

Q177. You are worried about unauthorized access to the Terraform state file since it might contain sensitive information. What are some ways you can protect the state file? (select two)

- use the S3 bucket using the `encrypt` option to ensure state is encrypted
- enable native encryption in Terraform as configured in the `terraform` block
- use Terraform Cloud which always encrypts state at rest
- replicate the state file to an encrypted storage device

If you manage any sensitive data with Terraform (like database passwords, user passwords, or private keys), treat the state itself as sensitive data. Storing state remotely can provide better security. Remote backend should encrypt the state data at rest

Reference: https://developer.hashicorp.com/terraform/language/state/sensitive-data

Terraform Workspace

Q178. Each Terraform CLI Workspace uses its own state file to manage the infrastructure associated with that particular workspace.

- true
- false

Terraform CLI Workspace refer to separate instances of state data inside the same Terraform working directory

Reference: https://developer.hashicorp.com/terraform/cli/workspaces

Q179. What Terraform feature is most applicable for managing small differences between different environments, for example development and production?

- Workspaces
- States
- Repositories
- Versions

Reference: https://developer.hashicorp.com/terraform/language/state/workspaces

Q180. Where are Terraform Workspace local state files stored?

- a directoy called `terraform.tfstate.d`
- a file called `terraform.tfstate`
- a temp directory called `.tfstate*`
- a directory called `terraform.workspaces.tfstate`

For local state, Terraform stores the workspace states in a directory called `terraform.tfstate.d`. This directory should be treated similarly to local-only `terraform.tfstate`.

Reference: https://developer.hashicorp.com/terraform/cli/workspaces#workspace-internals

Q181. You would like to reuse the same Terraform configuration for your development and production environments with a different state file for each. Which command would you use?

- `terraform import`
- `terraform workspace`
- `terraform state`
- `terraform init`

The `terraform workspace` CLI commands can be used to create multiple working directories to maintain multiple instances of same configuration with completely separate state data. Terraform CLI workspace is different from Terraform Cloud Workspace. Each Terraform Cloud workspace has its own Terraform configuration, set of variable values, state data, run history, and settings.

Reference: https://developer.hashicorp.com/terraform/cli/commands/workspace

Q182. One of your colleagues is new to Terraform and wants to add a new workspace named new-hire. What command he should execute from the following?

- `terraform workspace –new –new-hire`
- `terraform workspace new new-hire`
- `terraform workspace init new-hire`
- `terraform workspace new-hire`

The `terraform workspace new` command is used to create a new workspace with the given name

Reference: https://developer.hashicorp.com/terraform/cli/commands/workspace/new

Q183. As a prestigious Sr. Cloud Engineer, your colleague comes up to you and asks for a new Development workspace. What's the fastest way to accomplish this?

- Through CLI `terraform workspace new dev`
- Head to the Terraform Enterprise console and create a new workspace there.
- Specify in the configuration block the new workspace to be created.
- Have them submit a Jira ticket and tell them you'll get around to it in the next Sprint.

The Terraform CLI `terraform workspace new` command is fastest and easiest way to create new workspace

Reference: https://developer.hashicorp.com/terraform/cli/commands/workspace/new

Q184. A user creates three workspaces from the command line: `prod`, `dev`, and `test`. Which of the following commands will the user run to switch to the `dev` workspace?

- `terraform workspace switch dev`
- `terraform workspace select dev`
- `terraform workspace dev`
- `terraform workspace -switch dev`

The `terraform workspace select` command is used to choose a different workspace to use for further operations.

Reference: https://developer.hashicorp.com/terraform/cli/commands/workspace/select

Q185. Most workspaces in Terraform Cloud are associated with a VCS repository, which provides Terraform configurations for that workspace. Which of the following VCS providers Terraform Cloud supports?

- Github.com
- Gitlab.com
- Bitbucket Cloud
- CVS Version Control

Terraform Cloud supports the following VCS providers:

- GitHub
- GitHub App for TFE
- GitHub.com (OAuth)
- GitHub Enterprise
- GitLab.com
- GitLab EE and CE
- Bitbucket Cloud
- Bitbucket Server
- Azure DevOps Server
- Azure DevOps Services

Reference: https://developer.hashicorp.com/terraform/cloud-docs/vcs#supported-vcs-providers

Q186. Which is NOT true about Terraform Cloud and Terraform CLI Workspaces?

- Each Terraform Cloud workspace has its own Terraform configuration, variables, state file, backup of previous state files, run history, credentials & secrets, and settings.
- Each Terraform CLI workspace is a persistent working directory, which may contains a configuration, state data, and variables.
- You cannot manage resources in Terraform Cloud without creating at least one workspace.
- You must create a local working directory using Terraform CLI to manage resources in local.

By default when you run `terraform init`, Terraform CLI intialize the working directory with workspace name `default`. You don't need to create workspace manually.

Reference: https://developer.hashicorp.com/terraform/cloud-docs/workspaces#terraform-cloud-vs-terraform-cli-workspaces

Terraform Version Constraint

Q187. Which version constraint should use to set both a lower and upper bound on versions for each provider. Also known as pessimistic constraint operator?

- `>=`
- `~>`
- `!=`
- `<>`

Reference: https://developer.hashicorp.com/terraform/language/expressions/version-constraints

Q188. What does the specified contraint version = "~> 1.0.4" means in required_providers block?

```
terraform {
  required_providers {
    mycloud = {
      source  = "mycorp/mycloud"
      version = "~> 1.0.4"
    }
  }
}
```

- >= 1.0.4 and <= 1.1.0
- >= 1.0.4 and < 1.1.0
- > 1.0.4 and < 2.0.0
- >= 1.0.5 and < 1.1.0

`~>` *symbol before version* `x.y.z` *allows only the rightmost version component* `z` *to increment*

Reference: https://developer.hashicorp.com/terraform/language/expressions/version-constraints

Q189. What does this symbol version = "~> 1.0" mean when defining versions?

- > 1.0 and < 2.0
- >= 1.0 and < 2.0
- >= 1.0 and <= 2.0
- > 1.0.0 and < 2.0.0

`~>` *symbol before version* `x.y` *allows only the rightmost version component* `y` *to increment*

Reference: https://developer.hashicorp.com/terraform/language/expressions/version-constraints

Q190. What is the provider version of Google Cloud being used in Terraform? Select all valid options.

```
provider "google" {
  version = "~> 1.9.0"
}
```

- 1.9.1
- 1.10.0
- 1.8.0
- 1.9.9

`~>` *symbol before version* `x.y.z` *allows only the rightmost version component* `z` *to increment*

Reference: https://developer.hashicorp.com/terraform/language/expressions/version-constraints

Q191. How do you force users to use a particular version of required providers in your terraform code?

- `terraform { required_providers { aws = { source = "hashicorp/aws" version ="3.74.1" } } }`
- `terraform { aws = { source = "hashicorp/aws" version = "~>3.74.1" } }`
- `aws = { source = "hashicorp/aws" version = "3.74.1" }`
- `terraform { required_providers { aws = { source = "hashicorp/aws" version ="~>3.74.1" } } }`

Provider requirements such as version are declared in a `required_providers` *block using* `name = { source version }` *syntax*

Reference: https://developer.hashicorp.com/terraform/language/providers/requirements

Q192. Why might a user opt to include the following snippet in their configuration file?

```
terraform {
  required_version = ">= 1.3.8"
}
```

- this ensures that all Terraform providers are above a certain version to match the application being deployed
- versions before Terraform 1.3.8 were not approved by HashiCorp to be used in production
- The user wants to specify the minimum version of Terraform that is required to run the configuration
- The user wants to ensure that the application being deployed is a minimum version of 1.3.8

Reference: https://developer.hashicorp.com/terraform/language/settings#specifying-a-required-terraform-version

Terraform Types and Functions

Q193. You are adding a new variable to your configuration. Which of the following is not a valid primitive variable type in Terraform?

- string
- number
- float
- bool

Terraform has following primitive types: string, number and bool

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#primitive-types

Q194. What are two complex types in terraform? (Choose 2 answers)

- A Collection Type
- A Structural Type
- A String Type
- A float64 type

Collection and Structural types are the two types that are considered complex types in terraform

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#complex-types

Q195. What are complex types in terraform?

- A type that groups multiple values into a single value.
- A variation of a string type.
- A variance of a data source.
- A type that derives its value from RegEx logic.

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#complex-types

Q196. If an input variable has no type value set, what type does it accept?

- Any type.
- None, it has to have a type value set.
- Terraform infers the type when it is referenced.
- Type string. As strings can be interpreted in a number of ways by Terraform.

The type argument in a variable block allows you to restrict the type of value that will be accepted as the value for a variable. If no type constraint is set then a value of any type is accepted.

Reference: https://developer.hashicorp.com/terraform/language/values/variables#type-constraints

Q197. Which of the following is not a valid Terraform Collection type?

- list
- map
- tree
- set

The `list`, `map`, and `set` are three Terraform Collection types.

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#collection-types

Q198. Which of the followings are valid Terraform Structural types? (Choose 2 answers)

- optional
- object
- pair
- tuple

The `object` and `tuple` are two Terraform Structural types.

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#structural-types

Q199. You want to define a single input variable to store information about servers mainly server-name of type string and memory-size of type number. Which variable type should you choose?

- list
- map
- object
- set

A structural type `object` allows multiple values of several distinct types to be grouped together as a single value.

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#structural-types

Q200. You need to input variables that follow a key/value type structure. What type of variable would be used for this use case?

- list
- tuple
- map
- set

A `map` is a collection of values where each is identified by a string label.

Reference: https://developer.hashicorp.com/terraform/language/expressions/type-constraints#map

Q201. Which of the following is not a valid string function in Terraform?

- split()
- join()
- slice()
- chomp()

Reference: https://developer.hashicorp.com/terraform/language/functions

Q202. What are some built-in functions that terraform provides? (Choose 3 answers)

- max()
- regex()
- alltrue()
- delete()

Reference: https://developer.hashicorp.com/terraform/language/functions

See Also

- Hashicorp Terraform Associate (003) Exam Guide
- Leading SAFe (Scaled Agile Framework) Exam Notes
- AWS Certified Solutions Architect Associate (SAA-C02) Exam Notes
- MarkDown Getting Started
- Elastic Search - Basic Concepts

About Ashish Lahoti

Ashish Lahoti has 10+ years of experience in front-end and back-end technologies. He is a technology enthusiast and has a passion for coding & blogging.

« PREVIOUS
Hashicorp Terraform Associate (003) Exam Guide