



# HashiCorp

## Exam Questions Terraform-Associate-003

HashiCorp Certified: Terraform Associate (003)

## About ExamBible

### *Your Partner of IT Exam*

## Found in 1998

ExamBible is a company specialized on providing high quality IT exam practice study materials, especially Cisco CCNA, CCDA, CCNP, CCIE, Checkpoint CCSE, CompTIA A+, Network+ certification practice exams and so on. We guarantee that the candidates will not only pass any IT exam at the first attempt but also get profound understanding about the certificates they have got. There are so many alike companies in this industry, however, ExamBible has its unique advantages that other companies could not achieve.

## Our Advances

### \* 99.9% Uptime

All examinations will be up to date.

### \* 24/7 Quality Support

We will provide service round the clock.

### \* 100% Pass Rate

Our guarantee that you will pass the exam.

### \* Unique Gurantee

If you do not pass the exam at the first time, we will not only arrange FULL REFUND for you, but also provide you another exam of your claim, ABSOLUTELY FREE!

### NEW QUESTION 1

When does Terraform create the .terraform.lock.hcl file?

- A. After your first terraform plan
- B. After your first terraform apply
- C. After your first terraform init
- D. When you enable state locking

**Answer: C**

#### Explanation:

Terraform creates the .terraform.lock.hcl file after the first terraform init command. This lock file ensures that the dependencies for your project are consistent across different runs by locking the versions of the providers and modules used.

### NEW QUESTION 2

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

- A. Self-describing infrastructure
- B. Idempotence
- C. Versioned infrastructure
- D. Golden images

**Answer: D**

#### Explanation:

The key principle of infrastructure as code that is not listed among the options is golden images. Golden images are pre-configured, ready-to-use virtual machine images that contain a specific set of software and configuration. They are often used to create multiple identical instances of the same environment, such as for testing or production. However, golden images are not a principle of infrastructure as code, but rather a technique that can be used with or without infrastructure as code. The other options are all key principles of infrastructure as code, as explained below:

? Self-describing infrastructure: This means that the infrastructure is defined in code that describes its desired state, rather than in scripts that describe the steps to create it. This makes the infrastructure easier to understand, maintain, and reproduce.

? Idempotence: This means that applying the same infrastructure code multiple times will always result in the same state, regardless of the initial state. This makes the infrastructure consistent and predictable, and avoids errors or conflicts caused by repeated actions.

? Versioned infrastructure: This means that the infrastructure code is stored in a version control system, such as Git, that tracks the changes and history of the code. This makes the infrastructure code reusable, auditable, and collaborative, and enables practices such as branching, merging, and rollback. References = [Introduction to Infrastructure as Code with Terraform], [Infrastructure as Code in a Private or Public Cloud]

### NEW QUESTION 3

You've used Terraform to deploy a virtual machine and a database. You want to replace this virtual machine instance with an identical one without affecting the database. What is the best way to achieve this using Terraform?

- A. Use the terraform state rm command to remove the VM from state file
- B. Use the terraform taint command targeting the VMs then run terraform plan and terraform apply
- C. Use the terraform apply command targeting the VM resources only
- D. Delete the Terraform VM resources from your Terraform code then run terraform plan and terraform apply

**Answer: B**

#### Explanation:

The terraform taint command marks a resource as tainted, which means it will be destroyed and recreated on the next apply. This way, you can replace the VM instance without affecting the database or other resources. References = [Terraform Taint]

### NEW QUESTION 4

A terraform apply can not infrastructure.

- A. change
- B. destroy
- C. provision
- D. import

**Answer: D**

#### Explanation:

The terraform import command is used to import existing infrastructure into Terraform's state. This allows Terraform to manage and destroy the imported infrastructure as part of the configuration. The terraform import command does not modify the configuration, so the imported resources must be manually added to the configuration after the import. References = [Importing Infrastructure]

### NEW QUESTION 5

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

- A. A Terraform provider
- B. Environment variables
- C. A -var flag
- D. secure string

**Answer: D**

**Explanation:**

A secure string is not a valid option to keep secrets out of Terraform configuration files. A secure string is a feature of AWS Systems Manager Parameter Store that allows you to store sensitive data encrypted with a KMS key. However, Terraform does not support secure strings natively and requires a custom data source to retrieve them. The other options are valid ways to keep secrets out of Terraform configuration files. A Terraform provider can expose secrets as data sources that can be referenced in the configuration. Environment variables can be used to set values for input variables that contain secrets. A -var flag can be used to pass values for input variables that contain secrets from the command line or a file. References = [AWS Systems Manager Parameter Store], [Terraform AWS Provider Issue #55], [Terraform Providers], [Terraform Input Variables]

**NEW QUESTION 6**

You must initialize your working directory before running terraform validate.

- A. True
- B. False

**Answer:** A

**Explanation:**

You must initialize your working directory before running terraform validate, as it will ensure that all the required plugins and modules are installed and configured properly. If you skip this step, you may encounter errors or inconsistencies when validating your configuration files.

**NEW QUESTION 7**

When should you use the force-unlock command?

- A. You have a high priority change
- B. Automatic unlocking failed
- C. apply failed due to a state lock
- D. You see a status message that you cannot acquire the lock

**Answer:** B

**Explanation:**

You should use the force-unlock command when automatic unlocking failed. Terraform will lock your state for all operations that could write state, such as plan, apply, or destroy. This prevents others from acquiring the lock and potentially corrupting your state. State locking happens automatically on all operations that could write state and 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. If acquiring the lock is taking longer than expected, Terraform will output a status message. If Terraform doesn't output a message, state locking is still occurring if your backend supports it. Terraform has a force-unlock command to manually unlock the state if unlocking failed. Be very careful with this command. If you unlock the state when someone else is holding the lock it could cause multiple writers. Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed. To protect you, the force-unlock command requires a unique lock ID. Terraform will output this lock ID if unlocking fails. This lock ID acts as a nonce, ensuring that locks and unlocks target the correct lock. The other situations are not valid reasons to use the force-unlock command. You should not use the force-unlock command if you have a high priority change, if apply failed due to a state lock, or if you see a status message that you cannot acquire the lock. These situations indicate that someone else is holding the lock and you should wait for them to finish their operation or contact them to resolve the issue. Using the force-unlock command in these cases could result in data loss or inconsistency. References = [State Locking], [Command: force-unlock]

**NEW QUESTION 8**

terraform validate confirms that your infrastructure matches the Terraform state file.

- A. True
- B. False

**Answer:** B

**Explanation:**

terraform validate does not confirm that your infrastructure matches the Terraform state file. It only checks whether the configuration files in a directory are syntactically valid and internally consistent. To confirm that your infrastructure matches the Terraform state file, you need to use terraform plan or terraform apply with the -refresh- only option.

**NEW QUESTION 9**

A developer on your team is going to leave down an existing deployment managed by Terraform and deploy a new one. However, there is a server resource named aws\_instance.ubuntu[1] they would like to keep. What command should they use to tell Terraform to stop managing that specific resource?

- A. Terraform plan rm:aws\_instance.ubuntu[1]
- B. Terraform state rm:aws\_instance.ubuntu[1]
- C. Terraform apply rm:aws\_instance.ubuntu[1]
- D. Terraform destroy rm:aws\_instance.ubuntu[1]

**Answer:** B

**Explanation:**

To tell Terraform to stop managing a specific resource without destroying it, you can use the terraform state rm command. This command will remove the resource from the Terraform state, which means that Terraform will no longer track or update the corresponding remote object. However, the object will still exist in the remote system and you can later use terraform import to start managing it again in a different configuration or workspace. The syntax for this command is terraform state rm <address>, where <address> is the resource address that identifies the resource instance to remove. For example, terraform state rm aws\_instance.ubuntu[1] will remove the second instance of the aws\_instance resource named ubuntu from the state. References = : Command: state rm : Moving Resources

#### NEW QUESTION 10

Which of the following statements about Terraform modules is not true?

- A. Modules can call other modules
- B. A module is a container for one or more resources
- C. Modules must be publicly accessible
- D. You can call the same module multiple times

**Answer:** C

#### Explanation:

This is not true, as modules can be either public or private, depending on your needs and preferences. You can use the Terraform Registry to publish and consume public modules, or use Terraform Cloud or Terraform Enterprise to host and manage private modules.

#### NEW QUESTION 10

Which command should you run to check if all code in a Terraform configuration that references multiple modules is properly formatted without making changes?

- A. terraform fmt -write=false
- B. terraform fmt -list -recursive
- C. terraform fmt -check -recursive
- D. terraform fmt -check

**Answer:** C

#### Explanation:

This command will check if all code in a Terraform configuration that references multiple modules is properly formatted without making changes, and will return a non-zero exit code if any files need formatting. The other commands will either make changes, list the files that need formatting, or not check the modules.

#### NEW QUESTION 15

All standard backend types support state locking, and remote operations like plan, apply, and destroy.

- A. True
- B. False

**Answer:** B

#### Explanation:

Not all standard backend types support state locking and remote operations like plan, apply, and destroy. For example, the local backend does not support remote operations and state locking. State locking is a feature that ensures that no two users can make changes to the state file at the same time, which is crucial for preventing race conditions. Remote operations allow running Terraform commands on a remote server, which is supported by some backends like remote or consul, but not all.

References:

? Terraform documentation on backends: Terraform Backends

? Detailed backend support: Terraform Backend Types

#### NEW QUESTION 17

What are some benefits of using Sentinel with Terraform Cloud/Terraform Cloud? Choose three correct answers.

- A. You can restrict specific resource configurations, such as disallowing the use of CIDR=0.0.0.0/0.
- B. You can check out and check in cloud access keys
- C. Sentinel Policies can be written in HashiCorp Configuration Language (HCL)
- D. Policy-as-code can enforce security best practices
- E. You can enforce a list of approved AWS AMLs

**Answer:** ADE

#### Explanation:

Sentinel is a policy-as-code framework that allows you to define and enforce rules on your Terraform configurations, states, and plans<sup>1</sup>. Some of the benefits of using Sentinel with Terraform Cloud/Terraform Enterprise are:

- You can restrict specific resource configurations, such as disallowing the use of CIDR=0.0.0.0/0, which would open up your network to the entire internet. This can help you prevent misconfigurations or security vulnerabilities in your infrastructure<sup>2</sup>.

- Policy-as-code can enforce security best practices, such as requiring encryption, authentication, or compliance standards. This can help you protect your data and meet regulatory requirements<sup>3</sup>.

- You can enforce a list of approved AWS AMLs, which are pre-configured images that contain the operating system and software you need to run your applications. This can help you ensure consistency, reliability, and performance across your infrastructure<sup>4</sup>. References =

- 1: Terraform and Sentinel | Sentinel | HashiCorp Developer

- 2: Terraform Learning Resources: Getting Started with Sentinel in Terraform Cloud

- 3: Exploring the Power of HashiCorp Terraform, Sentinel, Terraform Cloud ??

- 4: Using New Sentinel Features in Terraform Cloud – Medium

#### NEW QUESTION 19

Which method for sharing Terraform configurations fulfills the following criteria:

- \* 1. Keeps the configurations confidential within your organization
- \* 2. Support Terraform??s semantic version constrains
- \* 3. Provides a browsable directory

- A. Subfolder within a workspace
- B. Generic git repository
- C. Terraform Cloud private registry

D. Public Terraform module registry

**Answer:** C

**Explanation:**

This is the method for sharing Terraform configurations that fulfills the following criteria:

? Keeps the configurations confidential within your organization

? Supports Terraform's semantic version constraints

? Provides a browsable directory

The Terraform Cloud private registry is a feature of Terraform Cloud that allows you to host and manage your own modules within your organization, and use them in your Terraform configurations with versioning and access control.

**NEW QUESTION 24**

The Terraform binary version and provider versions must match each other in a single configuration.

A. True

B. False

**Answer:** B

**Explanation:**

The Terraform binary version and provider versions do not have to match each other in a single configuration. Terraform allows you to specify provider version constraints in the configuration's terraform block, which can be different from the Terraform binary version1. Terraform will use the newest version of the provider that meets the configuration's version constraints2. You can also use the dependency lock file to ensure Terraform is using the correct provider version3.

References =

•1: Providers - Configuration Language | Terraform | HashiCorp Developer

•2: Multiple provider versions with Terraform - Stack Overflow

•3: Lock and upgrade provider versions | Terraform - HashiCorp Developer

**NEW QUESTION 29**

You add a new resource to an existing Terraform configuration, but do not update the version constraint in the configuration. The existing and new resources use the same provider. The working contains a .terraform.lock, hc1 file.

How will Terraform choose which version of the provider to use?

A. Terraform will use the version recorded in your lock file

B. Terraform will use the latest version of the provider for the new resource and the version recorded in the lock file to manage existing resources

C. Terraform will check your state file to determine the provider version to use

D. Terraform will use the latest version of the provider available at the time you provision your new resource

**Answer:** A

**Explanation:**

This is how Terraform chooses which version of the provider to use, when you add a new resource to an existing Terraform configuration, but do not update the version constraint in the configuration. The lock file records the exact version of each provider that was installed in your working directory, and ensures that Terraform will always use the same provider versions until you run terraform init -upgrade to update them.

**NEW QUESTION 33**

If a DevOps team adopts AWS CloudFormation as their standardized method for provisioning public cloud resources, which of the following scenarios poses a challenge for this team?

A. The team is asked to manage a new application stack built on AWS-native services

B. The organization decides to expand into Azure wishes to deploy new infrastructure

C. The team is asked to build a reusable code based that can deploy resources into any AWS region

D. The DevOps team is tasked with automating a manual, web console-based provisioning.

**Answer:** B

**Explanation:**

This is the scenario that poses a challenge for this team, if they adopt AWS CloudFormation as their standardized method for provisioning public cloud resources, as CloudFormation only supports AWS services and resources, and cannot be used to provision infrastructure on other cloud platforms such as Azure.

**NEW QUESTION 34**

What are some benefits of using Sentinel with Terraform Cloud/Terraform Cloud? Choose three correct answers.

A. You can enforce a list of approved AWS AMIs

B. Policy-as-code can enforce security best practices

C. You can check out and check in cloud access keys

D. You can restrict specific resource configurations, such as disallowing the use of CIDR=0.0.0.0/0.

E. Sentinel Policies can be written in HashiCorp Configuration Language (HCL)

**Answer:** ABD

**Explanation:**

These are some of the benefits of using Sentinel with Terraform Cloud/Terraform Enterprise, as they allow you to implement logic-based policies that can access and evaluate the Terraform plan, state, and configuration. The other options are not true, as Sentinel does not manage cloud access keys, and Sentinel policies are written in Sentinel language, not HCL.



#### NEW QUESTION 36

Setting the TF\_LOG environment variable to DEBUG causes debug messages to be logged into stdout.

- A. True
- B. False

**Answer:** A

#### Explanation:

Setting the TF\_LOG environment variable to DEBUG causes debug messages to be logged into stdout, along with other log levels such as TRACE, INFO, WARN, and ERROR. This can be useful for troubleshooting or debugging purposes.

#### NEW QUESTION 39

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

- A. Set verbose for each provider in your Terraform configuration
- B. Set the environment variable TF\_LOG\_TRACE
- C. Set the environment variable TF\_LOG\_PATH
- D. Set the environment variable TF\_log\_TRACE

**Answer:** B

#### Explanation:

This will enable additional logging messages to find out from which paths Terraform is loading providers referenced in your Terraform configuration files, as it will set the log level to TRACE, which is the most verbose and detailed level.

#### NEW QUESTION 41

You want to define multiple data disks as nested blocks inside the resource block for a virtual machine. What Terraform feature would help you define the blocks using the values in a variable?

- A. Local values
- B. Count arguments
- C. Collection functions
- D. Dynamic blocks

**Answer:** D

#### Explanation:

Dynamic blocks in Terraform allow you to define multiple nested blocks within a resource based on the values of a variable. This feature is particularly useful for scenarios where the number of nested blocks is not fixed and can change based on variable input.

#### NEW QUESTION 46

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

- A. choaf
- B. join
- C. Split
- D. slice

**Answer:** A

#### Explanation:

This is not a valid string function in Terraform. The other options are valid string functions that can manipulate strings in various ways.

#### NEW QUESTION 47

What type of block is used to construct a collection of nested configuration blocks?

- A. Dynamic
- B. For\_each
- C. Nesting
- D. repeated.

**Answer:** A

#### Explanation:

This is the type of block that is used to construct a collection of nested configuration blocks, by using a for\_each argument to iterate over a collection value and generate a nested block for each element. For example, you can use a dynamic block to create multiple ingress rules for a security group resource.

#### NEW QUESTION 52

Terraform can only manage resource dependencies if you set them explicitly with the depends\_on argument.

- A. True
- B. False

**Answer:** B

**Explanation:**

Terraform can manage resource dependencies implicitly or explicitly. Implicit dependencies are created when a resource references another resource or data source in its arguments. Terraform can infer the dependency from the reference and create or destroy the resources in the correct order. Explicit dependencies are created when you use the depends\_on argument to specify that a resource depends on another resource or module. This is useful when Terraform cannot infer the dependency from the configuration or when you need to create a dependency for some reason outside of Terraform's scope. References = : Create resource dependencies : Terraform Resource Dependencies Explained

**NEW QUESTION 53**

How is terraform import run?

- A. As a part of terraform init
- B. As a part of terraform plan
- C. As a part of terraform refresh
- D. By an explicit call
- E. All of the above

**Answer:** D

**Explanation:**

The terraform import command is not part of any other Terraform workflow. It must be explicitly invoked by the user with the appropriate arguments, such as the resource address and the ID of the existing infrastructure to import. References = [Importing Infrastructure]

**NEW QUESTION 54**

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

- A. Run terraform refresh
- B. It will happen automatically
- C. Manually update the state file
- D. Run terraform import

**Answer:** B

**Explanation:**

If you manually destroy infrastructure, Terraform will automatically detect the change and update the state file during the next plan or apply. Terraform compares the current state of the infrastructure with the desired state in the configuration and generates a plan to reconcile the differences. If a resource is missing from the infrastructure but still exists in the state file, Terraform will attempt to recreate it. If a resource is present in the infrastructure but not in the state file, Terraform will ignore it unless you use the terraform import command to bring it under Terraform's management. References = [Terraform State]

**NEW QUESTION 58**

Module variable assignments are inherited from the parent module and you do not need to explicitly set them.

- A. True
- B. False

**Answer:** B

**Explanation:**

Module variable assignments are not inherited from the parent module and you need to explicitly set them using the source argument. This allows you to customize the behavior of each module instance.

**NEW QUESTION 63**

How does the Terraform cloud integration differ from other state backends such as S3, Consul,etc?

- A. It can execute Terraform runs on dedicated infrastructure in Terraform Cloud
- B. It doesn't show the output of a terraform apply locally
- C. It is only arable lo paying customers
- D. All of the above

**Answer:** A

**Explanation:**

This is how the Terraform Cloud integration differs from other state backends such as S3, Consul, etc., as it allows you to perform remote operations on Terraform Cloud's servers instead of your local machine. The other options are either incorrect or irrelevant.

**NEW QUESTION 64**

Outside of the required\_providers block, Terraform configurations always refer to providers by their local names.

- A. True
- B. False

**Answer:** B

**Explanation:**

Outside of the required\_providers block, Terraform configurations can refer to providers by either their local names or their source addresses. The local name is a short name that can be used throughout the configuration, while the source address is a global identifier for the provider in the format registry.terraform.io/namespace/type. For example, you can use either aws or registry.terraform.io/hashicorp/aws to refer to the AWS provider.



#### NEW QUESTION 66

Module version is required to reference a module on the Terraform Module Registry.

- A. True
- B. False

**Answer:** B

#### Explanation:

Module version is optional to reference a module on the Terraform Module Registry. If you omit the version constraint, Terraform will automatically use the latest available version of the module

#### NEW QUESTION 68

The public Terraform Module Registry is free to use.

- A. True
- B. False

**Answer:** A

#### Explanation:

The public Terraform Module Registry is free to use, as it is a public service that hosts thousands of self-contained packages called modules that are used to provision infrastructure. You can browse, use, and publish modules to the registry without any cost.

#### NEW QUESTION 73

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

- A. Copy the sensitive variables into your Terraform code
- B. Store the sensitive variables in a secure\_varS.tf file
- C. Store the sensitive variables as plain text in a source code repository
- D. Pass variables to Terraform with a -var flag

**Answer:** D

#### Explanation:

This is a secure way to inject sensitive variables into your Terraform run, as they will not be stored in any file or source code repository. You can also use environment variables or variable files with encryption to pass sensitive variables to Terraform.

#### NEW QUESTION 75

You have a Terraform configuration that defines a single virtual machine with no references to it, You have run terraform apply to create the resource, and then removed the resource definition from your Terraform configuration file. What will happen you run terraform apply in the working directory again?

- A. Terraform will remove the virtual machine from the state file, but the resource will still exist
- B. Nothing
- C. Terraform will error
- D. Terraform will destroy the virtual machine

**Answer:** D

#### Explanation:

This is what will happen if you run terraform apply in the working directory again, after removing the resource definition from your Terraform configuration file. Terraform will detect that there is a resource in the state file that is not present in the configuration file, and will assume that you want to delete it.

#### NEW QUESTION 78

The \_\_\_\_\_ determines how Terraform creates, updates, or delete resources.

- A. Terraform configuration
- B. Terraform provisioner
- C. Terraform provider
- D. Terraform core

**Answer:** C

#### Explanation:

This is what determines how Terraform creates, updates, or deletes resources, as it is responsible for understanding API interactions with some service and exposing resources and data sources based on that API.

#### NEW QUESTION 83

Once you configure a new Terraform backend with a terraform code block, which command(s) should you use to migrate the state file?

- A. terraform destroy, then terraform apply
- B. terraform init
- C. terraform push
- D. terraform apply

**Answer:** A

**Explanation:**

This command will initialize the new backend and prompt you to migrate the existing state file to the new location<sup>4</sup>. The other commands are not relevant for this task.

**NEW QUESTION 88**

Define the purpose of state in Terraform.

- A. State maps real world resources to your configuration and keeps track of metadata
- B. State lets you enforce resource configurations that relate to compliance policies
- C. State stores variables and lets you quickly reuse existing code
- D. State codifies the dependencies of related resources

**Answer:** A

**Explanation:**

The purpose of state in Terraform is to keep track of the real-world resources managed by Terraform, mapping them to the configuration. The state file contains metadata about these resources, such as resource IDs and other important attributes, which Terraform uses to plan and manage infrastructure changes. The state enables Terraform to know what resources are managed by which configurations and helps in maintaining the desired state of the infrastructure. References = This role of state in Terraform is outlined in Terraform's official documentation, emphasizing its function in mapping configuration to real-world resources and storing vital metadata .

**NEW QUESTION 93**

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

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

**Answer:** D

**Explanation:**

This is a secure way to protect sensitive data in the state file, as it will be encrypted at rest and in transit<sup>2</sup>. The other options are not recommended, as they could lead to data loss, errors, or security breaches.

**NEW QUESTION 94**

What is the provider for this resource?

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

- A. Vpc
- B. Test
- C. Main
- D. aws

**Answer:** D

**Explanation:**

In the given Terraform configuration snippet: resource "aws\_vpc" "main" {  
name = "test"  
}

The provider for the resource aws\_vpc is aws. The provider is specified by the prefix of the resource type. In this case, aws\_vpc indicates that the resource type vpc is provided by the aws provider.

References:

? Terraform documentation on providers: Terraform Providers

**NEW QUESTION 95**

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?

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

**Answer:** A

**Explanation:**

The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style. This command applies a subset of the Terraform language style conventions, along with other minor adjustments for readability. Running this command on your configuration files before committing them to source control can help ensure consistency of style between different Terraform codebases, and can also make diffs easier to read. You can also use the -check and -diff options to check if the files are formatted and display the formatting changes respectively<sup>2</sup>. Running the terraform fmt command during the code linting phase of your CI/CD process can help automate this process and enforce the formatting standards for your team. References = [Command: fmt]<sup>2</sup>

**NEW QUESTION 97**

Terraform providers are part of the Terraform core binary.

- A. True
- B. False

**Answer:** B

**Explanation:**

Terraform providers are not part of the Terraform core binary. Providers are distributed separately from Terraform itself and have their own release cadence and version numbers. Providers are plugins that Terraform uses to interact with various APIs, such as cloud providers, SaaS providers, and other services. You can find and install providers from the Terraform Registry, which hosts providers for most major infrastructure platforms. You can also load providers from a local mirror or cache, or develop your own custom providers. To use a provider in your Terraform configuration, you need to declare it in the provider requirements block and optionally configure its settings in the provider block. References = : Providers - Configuration Language | Terraform : Terraform Registry  
- Providers Overview | Terraform

**NEW QUESTION 99**

It is best practice to store secret data in the same version control repository as your Terraform configuration.

- A. True
- B. False

**Answer:** B

**Explanation:**

It is not a best practice to store secret data in the same version control repository as your Terraform configuration, as it could expose your sensitive information to unauthorized parties or compromise your security. You should use environment variables, vaults, or other mechanisms to store and provide secret data to Terraform.

**NEW QUESTION 103**

You are working on some new application features and you want to spin up a copy of your production deployment to perform some quick tests. In order to avoid having to configure a new state backend, what open source Terraform feature would allow you create multiple states but still be associated with your current code?

- A. Terraform data sources
- B. Terraform local values
- C. Terraform modules
- D. Terraform workspaces
- E. None of the above

**Answer:** D

**Explanation:**

Terraform workspaces allow you to create multiple states but still be associated with your current code. Workspaces are like ??environments?? (e.g. staging, production) for the same configuration. You can use workspaces to spin up a copy of your production deployment for testing purposes without having to configure a new state backend. Terraform data sources, local values, and modules are not features that allow you to create multiple states. References = Workspaces and How to Use Terraform Workspaces

**NEW QUESTION 106**

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

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

**Answer:** B

**Explanation:**

This will trigger a run in the Terraform Cloud workspace, which will perform a plan and apply operation on the infrastructure defined by the Terraform configuration files in the VCS repository.

**NEW QUESTION 110**

What is a key benefit of the Terraform state file?

- A. A state file can schedule recurring infrastructure tasks
- B. A state file is a source of truth for resources provisioned with Terraform
- C. A state file is a source of truth for resources provisioned with a public cloud console
- D. A state file is the desired state expressed by the Terraform code files

**Answer:** B

**Explanation:**

This is a key benefit of the Terraform state file, as it stores and tracks the metadata and attributes of the resources that are managed by Terraform, and allows Terraform to compare the current state with the desired state expressed by your configuration files.

**NEW QUESTION 115**

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

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

**Answer:** BC

**Explanation:**

To import existing resources into Terraform, you need to do two things1:

? Write a resource configuration block for each resource, matching the type and name used in your state file.

? Run terraform import for each resource, specifying its address and ID. There is no such command as terraform Import-gcp, and provisioning new VMs with the same names will not import them into Terraform.

**NEW QUESTION 120**

Variables declared within a module are accessible outside of the module.

- A. True
- B. False

**Answer:** B

**Explanation:**

Variables declared within a module are only accessible within that module, unless they are explicitly exposed as output values1.

**NEW QUESTION 125**

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

- A. Plan
- B. Import
- C. Alidate
- D. Init
- E. apply

**Answer:** DE

**Explanation:**

The two steps that are required to provision new infrastructure in the Terraform workflow are init and apply. The terraform init command initializes a working directory containing Terraform configuration files. It downloads and installs the provider plugins that are needed for the configuration, and prepares the backend for storing the state. The terraform apply command applies the changes required to reach the desired state of the configuration, as described by the resource definitions in the configuration files. It shows a plan of the proposed changes and asks for confirmation before making any changes to the infrastructure. References = [The Core Terraform Workflow], [Initialize a Terraform working directory with init], [Apply Terraform Configuration with apply]

**NEW QUESTION 129**

A developer accidentally launched a VM (virtual machine) outside of the Terraform workflow and ended up with two servers with the same name. They don't know which VM Terraform manages but do have a list of all active VM IDs.

Which of the following methods could you use to discover which instance Terraform manages?

- A. Run terraform state list to find the names of all VMs, then run terraform state show for each of them to find which VM ID Terraform manages
- B. Update the code to include outputs for the ID of all VMs, then run terraform plan to view the outputs
- C. Run terraform taint/code on all the VMs to recreate them
- D. Use terraform refresh/code to find out which IDs are already part of state

**Answer:** A

**Explanation:**

The terraform state list command lists all resources that are managed by Terraform in the current state file1. The terraform state show command shows the attributes of a single resource in the state file2. By using these two commands, you can compare the VM IDs in your list with the ones in the state file and identify which one is managed by Terraform.

**NEW QUESTION 130**

How does Terraform manage most dependencies between resources?

- A. Terraform will automatically manage most resource dependencies
- B. Using the depends\_on parameter
- C. By defining dependencies as modules and including them in a particular order
- D. The order that resources appear in Terraform configuration indicates dependencies

**Answer:** A

**Explanation:**

This is how Terraform manages most dependencies between resources, by using the references between them in the configuration files. For example, if resource A depends on resource B, Terraform will create resource B first and then pass its attributes to resource A.

**NEW QUESTION 135**

How can a ticket-based system slow down infrastructure provisioning and limit the ability to scale? Choose two correct answers.

- A. End-users have to request infrastructure changes
- B. Ticket based systems generate a full audit trail of the request and fulfillment process
- C. Users can access catalog of approved resources from drop down list in a request form
- D. The more resources your organization needs, the more tickets your infrastructure team has to process

**Answer:** A

**Explanation:**

These are some of the ways that a ticket-based system can slow down infrastructure provisioning and limit the ability to scale, as they introduce delays, bottlenecks, and manual interventions in the process of creating and modifying infrastructure.

**NEW QUESTION 139**

Terraform variable names are saved in the state file.

- A. True
- B. False

**Answer:** B

**Explanation:**

Terraform variable names are not saved in the state file, only their values are. The state file only stores the attributes of the resources and data sources that are managed by Terraform, not the variables that are used to configure them.

**NEW QUESTION 143**

Which task does terraform init not perform?

- A. Validates all required variables are present
- B. Sources any modules and copies the configuration locally
- C. Connects to the backend
- D. Sources all providers used in the configuration and downloads them

**Answer:** A

**Explanation:**

The terraform init command is used to initialize a working directory containing Terraform configuration files. This command performs several different initialization steps to prepare the current working directory for use with Terraform, which includes initializing the backend, installing provider plugins, and copying any modules referenced in the configuration. However, it does not validate whether all required variables are present; that is a task performed by terraform plan or terraform apply<sup>1</sup>.

References = This information can be verified from the official Terraform documentation on the terraform init command provided by HashiCorp Developer<sup>1</sup>.

**NEW QUESTION 147**

terraform validate reports syntax check errors for which of the following?

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

**Answer:** D

**Explanation:**

The terraform validate command is used to check for syntax errors and internal consistency within Terraform configurations, such as whether all required arguments are specified. It does not check for indentation styles, missing variable values (as variables might not be defined at validation time), or state file consistency with the current infrastructure. Therefore, none of the provided options are correct in the context of what terraform validate reports. References = Terraform's official documentation details the purpose and function of the terraform validate command, specifying that it focuses on syntax and consistency checks within Terraform configurations themselves, not on external factors like the state file or infrastructure state. Direct references from the HashiCorp Terraform Associate (003) study materials to this specific detail were not found in the provided files.

**NEW QUESTION 149**

Which type of block fetches or computes information for use elsewhere in a Terraform configuration?

- A. data
- B. local
- C. resource
- D. provider

**Answer:** A



**Explanation:**

In Terraform, a data block is used to fetch or compute information from external sources for use elsewhere in the Terraform configuration. Unlike resource blocks that manage infrastructure, data blocks gather information without directly managing any resources. This can include querying for data from cloud providers, external APIs, or other Terraform states. References = This definition and usage of data blocks are covered in Terraform's official documentation, highlighting their role in fetching external information to inform Terraform configurations.

**NEW QUESTION 150**

Where in your Terraform configuration do you specify a state backend?

- A. The resource block
- B. The data source block
- C. The terraform block
- D. The provider block

**Answer: C**

**Explanation:**

In Terraform, the backend configuration, which includes details about where and how state is stored, is specified within the terraform block of your configuration. This block is the correct place to define the backend type and its configuration parameters, such as the location of the state file for a local backend or the bucket details for a remote backend like S3. References = This practice is outlined in Terraform's core documentation, which provides examples and guidelines on how to configure various aspects of Terraform's behavior, including state backends .

**NEW QUESTION 154**

Terraform configuration (including any module references) can contain only one Terraform provider type.

- A. True
- B. False

**Answer: B**

**Explanation:**

Terraform configuration (including any module references) can contain more than one Terraform provider type. Terraform providers are plugins that Terraform uses to interact with various cloud services and other APIs. A Terraform configuration can use multiple providers to manage resources across different platforms and services. For example, a configuration can use the AWS provider to create a virtual machine, the Cloudflare provider to manage DNS records, and the GitHub provider to create a repository. Terraform supports hundreds of providers for different use cases and scenarios. References = [Providers], [Provider Requirements], [Provider Configuration]

**NEW QUESTION 157**

Multiple team members are collaborating on infrastructure using Terraform and want to format the\* Terraform code following standard Terraform-style convention. How should they ensure the code satisfies conventions?

- A. Terraform automatically formats configuration on terraform apply
- B. Run terraform validate prior to executing terraform plan or terraform apply
- C. Use terraform fmt
- D. Replace all tabs with spaces

**Answer: C**

**Explanation:**

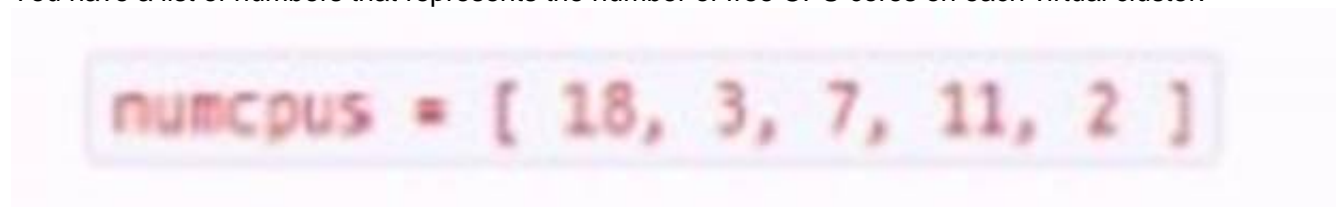
The terraform fmt command is used to format Terraform configuration files to a canonical format and style. This ensures that all team members are using a consistent style, making the code easier to read and maintain. It automatically applies Terraform's standard formatting conventions to your configuration files, helping maintain consistency across the team's codebase.

References:

? Terraform documentation on terraform fmt: Terraform Fmt

**NEW QUESTION 160**

You have a list of numbers that represents the number of free CPU cores on each virtual cluster:



What Terraform function could you use to select the largest number from the list?

- A. top(numcpus)
- B. max(numcpus)
- C. ceil (numcpus)
- D. hight[numcpus]

**Answer: B**

**Explanation:**

In Terraform, the max function can be used to select the largest number from a list of numbers. The max function takes multiple arguments and returns the highest one. For the list numcpus = [18, 3, 7, 11, 2], using max(numcpus...) will return 18, which is the largest number in the list.

References:

? Terraform documentation on max function: Terraform Functions - max



#### NEW QUESTION 163

Infrastructure as Code (IaC) can be stored in a version control system along with application code.

- A. True
- B. False

**Answer:** A

#### Explanation:

Infrastructure as Code (IaC) can indeed be stored in a version control system along with application code. This practice is a fundamental principle of modern infrastructure management, allowing teams to apply software development practices like versioning, peer review, and CI/CD to infrastructure management. Storing IaC configurations in version control facilitates collaboration, history tracking, and change management. While this concept is a foundational aspect of IaC and is widely accepted in the industry, direct references from the HashiCorp Terraform Associate (003) study materials were not found in the provided files. However, this practice is encouraged in Terraform's best practices and various HashiCorp learning resources.

#### NEW QUESTION 168

Why does this backend configuration not follow best practices?

```
terraform {
  backend "s3" {
    bucket      = "terraform-state-prod"
    key         = "network/terraform.tfstate"
    region      = "us-east-1"
    access_key  = "AKIAIOSFODNN7EXAMPLE"
    secret_key  = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.38"
    }
  }

  required_version = ">= 0.15"
}
```

- A. An alias meta-argument should be included in backend blocks whenever possible
- B. You should use the local enhanced storage backend whenever possible
- C. You should not store credentials in Terraform configuration
- D. The backend configuration should contain multiple credentials so that more than one user can execute terraform plan and terraform apply

**Answer:** C

#### Explanation:

This is a bad practice, as it exposes your credentials to anyone who can access your configuration files or state files. You should use environment variables, credential files, or other mechanisms to provide credentials to Terraform.

#### NEW QUESTION 172

Which of the following should you put into the required\_providers block?

- A. version >= 3.1
- B. version = ??>= 3.1??
- C. version ~> 3.1

**Answer:** B

#### Explanation:

The required\_providers block is used to specify the provider versions that the configuration can work with. The version argument accepts a version constraint string, which must be enclosed in double quotes. The version constraint string can use operators such as >=, ~>, =, etc. to specify the minimum, maximum, or exact version of the provider. For example, version = ">= 3.1" means that the configuration can work with any provider version that is 3.1 or higher. References = [Provider Requirements] and [Version Constraints]

#### NEW QUESTION 175

As a member of an operations team that uses infrastructure as code (IaC) practices, you are tasked with making a change to an infrastructure stack running in a public cloud. Which pattern would follow IaC best practices for making a change?

- A. Make the change via the public cloud API endpoint

- B. Clone the repository containing your infrastructure code and then run the code
- C. Use the public cloud console to make the change after a database record has been approved
- D. Make the change programmatically via the public cloud CLI
- E. Submit a pull request and wait for an approved merge of the proposed changes

**Answer:** E

**Explanation:**

You do not need to use different Terraform commands depending on the cloud provider you use. Terraform commands are consistent across different providers, as they operate on the Terraform configuration files and state files, not on the provider APIs directly.

**NEW QUESTION 176**

Which of these are features of Terraform Cloud? Choose two correct answers.

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

**Answer:** AD

**Explanation:**

Terraform Cloud includes several features designed to enhance collaboration and infrastructure management. Two of these features are:

? A web-based user interface (UI): This allows users to interact with Terraform Cloud

through a browser, providing a centralized interface for managing Terraform configurations, state files, and workspaces.

? Remote state storage: This feature enables users to store their Terraform state

files remotely in Terraform Cloud, ensuring that state is safely backed up and can be accessed by team members as needed.

**NEW QUESTION 179**

What does Terraform not reference when running a terraform apply -refresh-only ?

- A. State file
- B. Credentials
- C. Cloud provider
- D. Terraform resource definitions in configuration files

**Answer:** D

**Explanation:**

When running a terraform apply -refresh-only, Terraform does not reference the configuration files, but only the state file, credentials, and cloud provider. The purpose of this command is to update the state file with the current status of the real resources, without making any changes to them<sup>1</sup>.

**NEW QUESTION 182**

Which of the following is not true of Terraform providers?

- A. An individual person can write a Terraform Provider
- B. A community of users can maintain a provider
- C. HashiCorp maintains some providers
- D. Cloud providers and infrastructure vendors can write, maintain, or collaborate on Terraform
- E. providers
- F. None of the above

**Answer:** F

**Explanation:**

All of the statements are true of Terraform providers. Terraform providers are plugins that enable Terraform to interact with various APIs and services<sup>1</sup>. Anyone can write a Terraform provider, either as an individual or as part of a community<sup>2</sup>. HashiCorp maintains some providers, such as the AWS, Azure, and Google Cloud providers<sup>3</sup>. Cloud providers and infrastructure vendors can also write, maintain, or collaborate on Terraform providers, such as the VMware, Oracle, and Alibaba Cloud providers. References =

•<sup>1</sup>: Providers - Configuration Language | Terraform | HashiCorp Developer

•<sup>2</sup>: Plugin Development - How Terraform Works With Plugins | Terraform | HashiCorp Developer

•<sup>3</sup>: Terraform Registry

•: Terraform Registry

**NEW QUESTION 184**

What is terraform refresh-only intended to detect?

- A. Terraform configuration code changes
- B. Corrupt state files
- C. State file drift
- D. Empty state files

**Answer:** C

**Explanation:**

The terraform refresh-only command is intended to detect state file drift. This command synchronizes the state file with the actual infrastructure, updating the state to reflect any changes that have occurred outside of Terraform.

#### NEW QUESTION 185

terraform plan updates your state file.

- A. True
- B. False

**Answer:** B

#### Explanation:

The terraform plan command does not update the state file. Instead, it reads the current state and the configuration files to determine what changes would be made to bring the real-world infrastructure into the desired state defined in the configuration. The plan operation is a read-only operation and does not modify the state or the infrastructure. It is the terraform apply command that actually applies changes and updates the state file. References = Terraform's official guidelines and documentation clarify the purpose of the terraform plan command, highlighting its role in preparing and showing an execution plan without making any changes to the actual state or infrastructure .

#### NEW QUESTION 188

A Terraform output that sets the "sensitive" argument to true will not store that value in the state file.

- A. True
- B. False

**Answer:** A

#### Explanation:

A Terraform output that sets the "sensitive" argument to true will store that value in the state file. The purpose of setting sensitive = true is to prevent the value from being displayed in the CLI output during terraform plan and terraform apply, and to mask it in the Terraform UI. However, it does not affect the storage of the value in the state file. Sensitive outputs are still written to the state file to ensure that Terraform can manage resources correctly during subsequent operations.

References:

? Terraform documentation on sensitive outputs: Terraform Output Values

#### NEW QUESTION 193

What is one disadvantage of using dynamic blocks in Terraform?

- A. Dynamic blocks can construct repeatable nested blocks
- B. Terraform will run more slowly
- C. They cannot be used to loop through a list of values
- D. They make configuration harder to read and understand

**Answer:** D

#### Explanation:

This is one disadvantage of using dynamic blocks in Terraform, as they can introduce complexity and reduce readability of the configuration. The other options are either advantages or incorrect statements.

#### NEW QUESTION 197

What does terraform import do?

- A. Imports existing resources into the state file
- B. Imports all infrastructure from a given cloud provider
- C. Imports a new Terraform module
- D. Imports clean copies of tainted resources
- E. None of the above

**Answer:** A

#### Explanation:

The terraform import command is used to import existing infrastructure into your Terraform state. This command takes the existing resource and associates it with a resource defined in your Terraform configuration, updating the state file accordingly. It does not generate configuration for the resource, only the state.

#### NEW QUESTION 199

Which are examples of infrastructure as code? Choose two correct answers.

- A. Cloned virtual machine images
- B. Versioned configuration files
- C. Change management database records
- D. Doctor files

**Answer:** B

#### Explanation:

These are examples of infrastructure as code (IaC), which is a practice of managing and provisioning infrastructure through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

#### NEW QUESTION 202

Select the command that doesn't cause Terraform to refresh its state.

- A. Terraform destroy

- B. Terraform apply
- C. Terraform plan
- D. Terraform state list

**Answer:** D

**Explanation:**

This is the command that does not cause Terraform to refresh its state, as it only lists the resources that are currently managed by Terraform in the state file. The other commands will refresh the state file before performing their operations, unless you use the `-refresh=false` flag.

**NEW QUESTION 207**

Which of the following module source paths does not specify a remote module?

- A. Source = `??module/consul????`
- B. Source = `????github.comicrop/example????`
- C. Source = `????git@github.com:hasicrop/example.git????`
- D. Source = `????hasicrop/consul/aws????`

**Answer:** A

**Explanation:**

The module source path that does not specify a remote module is `source = "module/consul"`. This specifies a local module, which is a module that is stored in a subdirectory of the current working directory. The other options are all examples of remote modules, which are modules that are stored outside of the current working directory and can be accessed by various protocols, such as Git, HTTP, or the Terraform Registry. Remote modules are useful for sharing and reusing code across different configurations and environments. References = [Module Sources], [Local Paths], [Terraform Registry], [Generic Git Repository], [GitHub]

**NEW QUESTION 209**

Which of these statements about Terraform Cloud workspaces is false?

- A. They have role-based access controls
- B. You must use the CLI to switch between workspaces
- C. Plans and applies can be triggered via version control system integrations
- D. They can securely store cloud credentials

**Answer:** B

**Explanation:**

The statement that you must use the CLI to switch between workspaces is false. Terraform Cloud workspaces are different from Terraform CLI workspaces. Terraform Cloud workspaces are required and represent all of the collections of infrastructure in an organization. They are also a major component of role-based access in Terraform Cloud. You can grant individual users and user groups permissions for one or more workspaces that dictate whether they can manage variables, perform runs, etc. You can create, view, and switch between Terraform Cloud workspaces using the Terraform Cloud UI, the Workspaces API, or the Terraform Enterprise Provider<sup>5</sup>. Terraform CLI workspaces are optional and allow you to create multiple distinct instances of a single configuration within one working directory. They are useful for creating disposable environments for testing or experimenting without affecting your main or production environment. You can create, view, and switch between Terraform CLI workspaces using the `terraform workspace` command<sup>6</sup>. The other statements about Terraform Cloud workspaces are true. They have role-based access controls that allow you to assign permissions to users and teams based on their roles and responsibilities. You can create and manage roles using the Teams API or the Terraform Enterprise Provider<sup>7</sup>. Plans and applies can be triggered via version control system integrations that allow you to link your Terraform Cloud workspaces to your VCS repositories. You can configure VCS settings, webhooks, and branch tracking to automate your Terraform Cloud workflow<sup>8</sup>. They can securely store cloud credentials as sensitive variables that are encrypted at rest and only decrypted when needed. You can manage variables using the Terraform Cloud UI, the Variables API, or the Terraform Enterprise Provider<sup>9</sup>. References = [Workspaces]<sup>5</sup>, [Terraform CLI Workspaces]<sup>6</sup>, [Teams and Organizations]<sup>7</sup>, [VCS Integration]<sup>8</sup>, [Variables]<sup>9</sup>

**NEW QUESTION 210**

.....

## Relate Links

**100% Pass Your Terraform-Associate-003 Exam with ExamBible Prep Materials**

<https://www.exambible.com/Terraform-Associate-003-exam/>

## Contact us

We are proud of our high-quality customer service, which serves you around the clock 24/7.

Viste - <https://www.exambible.com/>