

# HashiCorp

## Exam Questions Terraform-Associate-003

HashiCorp Certified: Terraform Associate (003)



**NEW QUESTION 1**

How would you reference the volume IDs associated with the ebs\_block\_device blocks in this configuration?

```
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  
  }  
}
```

- A. aws\_instance.example.ebs\_block\_device[sda2,sda3].volume\_id
- B. aws\_instance.example.ebs\_block\_device.[\*].volume\_id
- C. aws\_instance.example.ebs\_block\_device.volume\_ids
- D. aws\_instance.example-ebs\_block\_device.\*.volume\_id

**Answer:** D

**Explanation:**

This is the correct way to reference the volume IDs associated with the ebs\_block\_device blocks in this configuration, using the splat expression syntax. The other options are either invalid or incomplete.

**NEW QUESTION 2**

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 3**

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 4**

Terraform configuration can only import modules from the public registry.

- A. True
- B. False

**Answer:** B

**Explanation:**

Terraform configuration can import modules from various sources, not only from the public registry. Modules can be sourced from local file paths, Git repositories, HTTP URLs, Mercurial repositories, S3 buckets, and GCS buckets. Terraform supports a number of common conventions and syntaxes for specifying module sources, as documented in the [Module Sources] page. References = [Module Sources]

**NEW QUESTION 5**

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 6**

What does this code do?

```
terraform {
  required_providers {
    aws = "~> 3.0"
  }
}
```

- A. Requires any version of the AWS provider > = 3.0 and <4.0
- B. Requires any version of the AWS provider >= 3.0
- C. Requires any version of the AWS provider > = 3.0 major releases
- D. like 4.1
- E. Requires any version of the AWS provider > 3.0

**Answer:** A

**Explanation:**

This is what this code does, by using the pessimistic constraint operator (~>), which specifies an acceptable range of versions for a provider or module.

**NEW QUESTION 7**

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 8**

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 9

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 of the following is not a valid Terraform collection type?

- A. Tree
- B. Map
- C. List
- D. set

**Answer:** A

#### Explanation:

This is not a valid Terraform collection type, as Terraform only supports three collection types: list, map, and set. A tree is a data structure that consists of nodes with parent-child relationships, which is not supported by Terraform.

#### NEW QUESTION 10

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 15

When you use a remote backend that needs authentication, HashiCorp recommends that you:

- A. Write the authentication credentials in the Terraform configuration files
- B. Keep the Terraform configuration files in a secret store
- C. Push your Terraform configuration to an encrypted git repository
- D. Use partial configuration to load the authentication credentials outside of the Terraform code

**Answer:** D

#### Explanation:

This is the recommended way to use a remote backend that needs authentication, as it allows you to provide the credentials via environment variables, command-line arguments, or interactive prompts, without storing them in the Terraform configuration files.

#### NEW QUESTION 16

Only the user that generated a plan may apply it.

- A. True
- B. False

**Answer:** B

#### Explanation:

Any user with permission to apply a plan can apply it, not only the user that generated it. This allows for collaboration and delegation of tasks among team members.

#### NEW QUESTION 19

You should run terraform fmt to rewrite all Terraform configurations within the current working directory to conform to Terraform-style conventions.

- A. True
- B. False

**Answer:** A

**Explanation:**

You should run terraform fmt to rewrite all Terraform configurations within the current working directory to conform to Terraform-style conventions. This command applies a subset of the Terraform language style conventions, along with other minor adjustments for readability. It is recommended to use this command to ensure consistency of style across different Terraform codebases. The command is optional, opinionated, and has no customization options, but it can help you and your team understand the code more quickly and easily. References = : Command: fmt : Using Terraform fmt Command to Format Your Terraform Code

**NEW QUESTION 23**

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 version<sup>1</sup>. Terraform will use the newest version of the provider that meets the configuration's version constraints<sup>2</sup>. You can also use the dependency lock file to ensure Terraform is using the correct provider version<sup>3</sup>.

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 27**

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 32**

What feature stops multiple users from operating on the Terraform state at the same time?

- A. State locking
- B. Version control
- C. Provider constraints
- D. Remote backends

**Answer:** A

**Explanation:**

State locking prevents other users from modifying the state file while a Terraform operation is in progress. This prevents conflicts and data loss<sup>1</sup>.

**NEW QUESTION 33**

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 35**

You can develop a custom provider to manage its resources using Terraform.

- A. True
- B. False

**Answer:** A

**Explanation:**

You can develop a custom provider to manage its resources using Terraform, as Terraform is an extensible tool that allows you to write your own plugins in Go

language. You can also publish your custom provider to the Terraform Registry or use it privately.

**NEW QUESTION 39**

You modified your Terraform configuration and run Terraform plan to review the changes. Simultaneously, your teammate manually modified the infrastructure component you are working on. Since you already ran terraform plan locally, the execution plan for terraform apply will be the same.

- A. True
- B. False

**Answer: B**

**Explanation:**

The execution plan for terraform apply will not be the same as the one you ran locally with terraform plan, if your teammate manually modified the infrastructure component you are working on. This is because Terraform will refresh the state file before applying any changes, and will detect any differences between the state and the real resources.

**NEW QUESTION 44**

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

- A. Required input variables
- B. Optional inputs variables and default values
- C. Outputs
- D. All of the above
- E. None of the above

**Answer: D**

**Explanation:**

The public Terraform Module Registry automatically exposes all the information about published modules, including required input variables, optional input variables and default values, and outputs. This helps users to understand how to use and configure the modules.

**NEW QUESTION 47**

You want to know from which paths Terraform is loading providers referenced in your Terraform configuration (\* 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 48**

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 50**

In a Terraform Cloud workspace linked to a version control repository speculative plan run start automatically commit changes to version control.

- A. True
- B. False

**Answer: A**

**Explanation:**

When you use a remote backend that needs authentication, HashiCorp recommends that you:

**NEW QUESTION 53**

How could you reference an attribute from the vsphere\_datacenter data source for use with the datacenter\_id argument within the vsphere\_folder resource in the following configuration?



```
data "vsphere_datacenter" "dc" {}

resource "vsphere_folder" "parent" {
    path = "Production"
    type = "vm"
    datacenter_id = _____
}
```

- A. Data.vsphere\_datacenter.DC.id
- B. Vsphere\_datacenter.dc.id
- C. Data,dc,id
- D. Data.vsphere\_datacenter,dc

**Answer:** A

**Explanation:**

The correct way to reference an attribute from the vsphere\_datacenter data source for use with the datacenter\_id argument within the vsphere\_folder resource in the following configuration is data.vsphere\_datacenter.dc.id. This follows the syntax for accessing data source attributes, which is data.TYPE.NAME.ATTRIBUTE. In this case, the data source type is vsphere\_datacenter, the data source name is dc, and the attribute we want to access is id. The other options are incorrect because they either use the wrong syntax, the wrong punctuation, or the wrong case. References = [Data Source: vsphere\_datacenter], [Data Source: vsphere\_folder], [Expressions: Data Source References]

**NEW QUESTION 56**

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**

When should you write Terraform configuration files for existing infrastructure that you want to start managing with Terraform?

- A. You can import infrastructure without corresponding Terraform code
- B. Terraform will generate the corresponding configuration files for you
- C. Before you run terraform Import
- D. After you run terraform import

**Answer:** C

**Explanation:**

You need to write Terraform configuration files for the existing infrastructure that you want to import into Terraform, otherwise Terraform will not know how to manage it. The configuration files should match the type and name of the resources that you want to import.

**NEW QUESTION 62**

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 63

Which of the following does terraform apply change after you approve the execution plan? (Choose two.)

- A. Cloud infrastructure Most Voted
- B. The .terraform directory
- C. The execution plan
- D. State file
- E. Terraform code

**Answer:** AD

#### Explanation:

The terraform apply command changes both the cloud infrastructure and the state file after you approve the execution plan. The command creates, updates, or destroys the infrastructure resources to match the configuration. It also updates the state file to reflect the new state of the infrastructure. The .terraform directory, the execution plan, and the Terraform code are not changed by the terraform apply command. References = Command: apply and Purpose of Terraform State

### 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 71

When do changes invoked by terraform apply take effect?

- A. After Terraform has updated the state file
- B. Once the resource provider has fulfilled the request
- C. Immediately
- D. None of the above are correct

**Answer:** B

#### Explanation:

Changes invoked by terraform apply take effect once the resource provider has fulfilled the request, not after Terraform has updated the state file or immediately. The state file is only a reflection of the real resources, not a source of truth.

### NEW QUESTION 72

You have used Terraform to create an ephemeral development environment in the cloud and are now ready to destroy all the Infrastructure described by your Terraform configuration. To be safe, you would like to first see all the infrastructure that Terraform will delete.

Which command should you use to show all of the resources that will be deleted? Choose two correct answers.

- A. Run terraform state rm ??
- B. Run terraform show :destroy
- C. Run terraform destroy and it will first output all the resource that will be deleted before prompting for approval
- D. Run terraform plan .destory

**Answer:** CD

#### Explanation:

To see all the resources that Terraform will delete, you can use either of these two commands:  
? terraform destroy will show the plan of destruction and ask for your confirmation before proceeding. You can cancel the command if you do not want to destroy the resources.  
? terraform plan -destroy will show the plan of destruction without asking for confirmation. You can use this command to review the changes before running terraform destroy. References = : Destroy Infrastructure : Plan Command: Options

### NEW QUESTION 76

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 79**

In a Terraform Cloud workspace linked to a version control repository, speculative plan runs start automatically when you merge or commit changes to version control.

- A. True
- B. False

**Answer:** B

**Explanation:**

In Terraform Cloud, speculative plan runs are not automatically started when changes are merged or committed to the version control repository linked to a workspace. Instead, speculative plans are typically triggered as part of proposed changes in merge requests or pull requests to give an indication of what would happen if the changes were applied, without making any real changes to the infrastructure. Actual plan and apply operations in Terraform Cloud workspaces are usually triggered by specific events or configurations defined within the Terraform Cloud workspace settings. References = This behavior is part of how Terraform Cloud integrates with version control systems and is documented in Terraform Cloud's usage guidelines and best practices, especially in the context of VCS-driven workflows.

**NEW QUESTION 80**

You are creating a Terraform configuration which needs to make use of multiple providers, one for AWS and one for Datadog. Which of the following provider blocks would allow you to do this?

A)

```
terraform {  
  provider "aws" {  
    profile = var.aws_profile  
    region  = var.aws_region  
  }  
  
  provider "datadog" {  
    api_key = var.datadog_api_key  
    app_key = var.datadog_app_key  
  }  
}
```

B)

```
provider "aws" {  
    profile = var.aws_profile  
    region  = var.aws_region  
}  
  
provider "datadog" {  
    api_key = var.datadog_api_key  
    app_key = var.datadog_app_key  
}
```

C)

```
provider "aws" {  
    profile = var.aws_profile  
    region  = var.aws_region  
}  
  
provider "datadog" {  
    api_key = var.datadog_api_key  
    app_key = var.datadog_app_key  
}
```

D)

```
provider {  
  "aws" {  
    profile = var.aws_profile  
    region  = var.aws_region  
  }  
  
  "datadog" {  
    api_key = var.datadog_api_key  
    app_key = var.datadog_app_key  
  }  
}
```

- A. Option A
- B. Option B
- C. Option C
- D. Option D

**Answer:** C

**Explanation:**

Option C is the correct way to configure multiple providers in a Terraform configuration. Each provider block must have a name attribute that specifies which provider it configures<sup>2</sup>. The other options are either missing the name attribute or using an invalid syntax.

**NEW QUESTION 84**

Which of the following command would be use to access all of the attributes and details of a resource managed by Terraform?

- A. Terraform state show ?? provider\_type\_name
- B. Terraform state list
- C. Terraform get provider\_type\_name
- D. Terraform state list provider\_type\_name

**Answer:** A

**Explanation:**

This is the command that you would use to access all of the attributes and details of a resource managed by Terraform, by providing the resource address as an argument. For example, terraform state show 'aws\_instance.example' will show you all the information about the AWS instance named example.

**NEW QUESTION 85**

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 file<sup>1</sup>. The terraform state show command shows the attributes of a single resource in the state file<sup>2</sup>. 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 86

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 88

When should you run terraform init?

- A. Every time you run terraform apply
- B. Before you start coding a new Terraform project
- C. After you run terraform plan for the time in a new terraform project and before you run terraform apply
- D. After you start coding a new terraform project and before you run terraform plan for the first time.

**Answer:** D

#### Explanation:

You should run terraform init after you start coding a new Terraform project and before you run terraform plan for the first time. This command will initialize the working directory by downloading the required providers and modules, creating the initial state file, and performing other necessary tasks. References = : Initialize a Terraform Project

#### NEW QUESTION 90

How can terraform plan aid in the development process?

- A. Initializes your working directory containing your Terraform configuration files
- B. Validates your expectations against the execution plan without permanently modifying state
- C. Formats your Terraform configuration files
- D. Reconciles Terraform's state against deployed resources and permanently modifies state using the current status of deployed resources

**Answer:** B

#### Explanation:

The terraform plan command is used to create an execution plan. It allows you to see what actions Terraform will take to reach the desired state defined in your configuration files. It evaluates the current state and configuration, showing a detailed outline of the resources that will be created, updated, or destroyed. This is a critical step in the development process as it helps you verify that the changes you are about to apply will perform as expected, without actually modifying any state or infrastructure.

References:

? Terraform documentation on terraform plan: Terraform Plan

#### NEW QUESTION 95

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 98**

You're writing a Terraform configuration that needs to read input from a local file called id\_rsa.pub . Which built-in Terraform function can you use to import the file's contents as a string?

- A. file("id\_rsa.pub")
- B. templatefile("id\_rsa.pub")
- C. filebase64("id\_rsa.pub")
- D. fileset("id\_rsa.pub")

**Answer: A**

**Explanation:**

To import the contents of a local file as a string in Terraform, you can use the built-in file function. By specifying file("id\_rsa.pub"), Terraform reads the contents of the id\_rsa.pub file and uses it as a string within your Terraform configuration. This function is particularly useful for scenarios where you need to include file data directly into your configuration, such as including an SSH public key for provisioning cloud instances. References = This information is a standard part of Terraform's functionality with built-in functions, as outlined in Terraform's official documentation and commonly used in various Terraform configurations.

**NEW QUESTION 99**

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

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

**Answer: D**

**Explanation:**

The best way to automatically and proactively enforce the security control that new AWS S3 buckets must be private and encrypted at rest is with a Sentinel policy, which runs before every apply. Sentinel is a policy as code framework that allows you to define and enforce logic-based policies for your infrastructure. Terraform Cloud supports Sentinel policies for all paid tiers, and can run them before any terraform plan or terraform apply operation. You can write a Sentinel policy that checks the configuration of the S3 buckets and ensures that they have the proper settings for privacy and encryption, and then assign the policy to your Terraform Cloud organization or workspace. This way, Terraform Cloud will prevent any changes that violate the policy from being applied. References = [Sentinel Policy Framework], [Manage Policies in Terraform Cloud], [Write and Test Sentinel Policies for Terraform]



**NEW QUESTION 100**

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 104**

All modules published on the official Terraform Module Registry have been verified by HashiCorp.

- A. True
- B. False

**Answer:** B

**Explanation:**

Not all modules published on the official Terraform Module Registry have been verified by HashiCorp. While HashiCorp verifies some modules, there are many community-contributed modules that are not verified. Verified modules have a "Verified" badge indicating that HashiCorp has reviewed them for security and best practices, but the registry also includes unverified modules.

References:

? Terraform Module Registry documentation: Terraform Registry

**NEW QUESTION 107**

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

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

**Answer:** C

**Explanation:**

This is the best way to delete the newly-created VM with Terraform, as it will only affect the resource that was created by your configuration and state file. The other options are either incorrect or inefficient.

**NEW QUESTION 109**

You must use different Terraform commands depending on the cloud provider you use.

- A. True
- B. False

**Answer:** B

**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 111**

You decide to move a Terraform state file to Amazon S3 from another location. You write the code below into a file called backend.tf.

```
terraform {  
  backend "s3" {  
    bucket = "my-tf-bucket"  
    region = "us-east-1"  
  }  
}
```

Which command will migrate your current state file to the new S3 remote backend?

- A. terraform state
- B. terraform init



- C. terraform push
- D. terraform refresh

**Answer:** B

**Explanation:**

This command will initialize the new backend and prompt you to migrate the existing state file to the new location<sup>3</sup>. The other commands are not relevant for this task.

**NEW QUESTION 112**

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 113**

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 116**

Any user can publish modules to the public Terraform Module Registry.

- A. True
- B. False

**Answer:** A

**Explanation:**

The Terraform Registry allows any user to publish and share modules. Published modules support versioning, automatically generate documentation, allow browsing version histories, show examples and READMEs, and more. Public modules are managed via Git and GitHub, and publishing a module takes only a few minutes. Once a module is published, releasing a new version of a module is as simple as pushing a properly formed Git tag<sup>1</sup>.

References = The information can be verified from the Terraform Registry documentation on Publishing Modules provided by HashiCorp Developer<sup>1</sup>.

**NEW QUESTION 121**

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 123**

You can reference a resource created with for\_each using a Splat ( \*) expression.

- A. True

B. False

**Answer:** B

**Explanation:**

You cannot reference a resource created with `for_each` using a splat (\*) expression, as it will not work with resources that have non-numeric keys. You need to use a `for` expression instead to iterate over the resource instances.

**NEW QUESTION 126**

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 130**

.....

## Thank You for Trying Our Product

### We offer two products:

1st - We have Practice Tests Software with Actual Exam Questions

2nd - Questions and Answers in PDF Format

### Terraform-Associate-003 Practice Exam Features:

- \* Terraform-Associate-003 Questions and Answers Updated Frequently
- \* Terraform-Associate-003 Practice Questions Verified by Expert Senior Certified Staff
- \* Terraform-Associate-003 Most Realistic Questions that Guarantee you a Pass on Your FirstTry
- \* Terraform-Associate-003 Practice Test Questions in Multiple Choice Formats and Updatesfor 1 Year

**100% Actual & Verified — Instant Download, Please Click**  
**[Order The Terraform-Associate-003 Practice Test Here](#)**