

## Exam Questions Terraform-Associate-003

HashiCorp Certified: Terraform Associate (003)

<https://www.2passeasy.com/dumps/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**

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 have to initialize a Terraform backend before it can be configured.

- A. True
- B. False

**Answer:** B

**Explanation:**

You can configure a backend in your Terraform code before initializing it. Initializing a backend will store the state file remotely and enable features like locking and workspaces. References = [Terraform Backends]

**NEW QUESTION 4**

Changing the Terraform backend from the default "local" backend to a different one after performing your first terraform apply is:

- A. Optional
- B. Impossible

- C. Mandatory
- D. Discouraged

**Answer:** D

**Explanation:**

Changing the Terraform backend after performing the initial terraform apply is technically possible but strongly discouraged. This is because changing backends can lead to complexities in state management, requiring manual intervention such as state migration to ensure consistency. Terraform's documentation and best practices advise planning the backend configuration carefully before applying Terraform configurations to avoid such changes. References = This guidance is consistent with Terraform's official documentation, which recommends careful consideration and planning of backend configurations to avoid the need for changes.

**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**

If a module declares a variable with a default, that variable must also be defined within the module.

- A. True
- B. False

**Answer:** B

**Explanation:**

A module can declare a variable with a default value without requiring the caller to define it. This allows the module to provide a sensible default behavior that can be customized by the caller if needed. References = [Module Variables]

**NEW QUESTION 7**

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 8**

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 9**

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.

**NEW QUESTION 10**

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 and wishes to deploy new infrastructure
- C. The team is asked to build a reusable code base 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 10**

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 13**

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 14**

Which configuration consistency errors does terraform validate report?

- A. Terraform module isn't the latest version
- B. Differences between local and remote state
- C. Declaring a resource identifier more than once
- D. A mix of spaces and tabs in configuration files

**Answer: C**

**Explanation:**

Terraform validate reports configuration consistency errors, such as declaring a resource identifier more than once. This means that the same resource type and name combination is used for multiple resource blocks, which is not allowed in Terraform. For example, resource "aws\_instance" "example" {...} cannot be used more than once in the same configuration. Terraform validate does not report errors related to module versions, state differences, or formatting issues, as these are not relevant for checking the configuration syntax and structure. References = [Validate Configuration], [Resource Syntax]

**NEW QUESTION 15**

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 20

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 23

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 26

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 28

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 33

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 38

Which of the following is not a valid source path for specifying a module?



- A. source - "github.com/hashicorp/examplePref-ul.0.8M
- B. source = "./module?version=vl.6.0"
- C. source - "hashicorp/consul/aws"
- D. source - "./module"

**Answer:** B

**Explanation:**

Terraform modules are referenced by specifying a source location. This location can be a URL or a file path. However, specifying query parameters such as ?version=vl.6.0 directly within the source path is not a valid or supported method for specifying a module version in Terraform. Instead, version constraints are specified using the version argument within the module block, not as part of the source string.

References

= This clarification is based on Terraform's official documentation regarding module usage, which outlines the correct methods for specifying module sources and versions.

**NEW QUESTION 39**

Which of these actions will prevent two Terraform runs from changing the same state file at the same time?

- A. Refresh the state after running Terraform
- B. Delete the state before running Terraform
- C. Configure state locking for your state backend
- D. Run Terraform with parallelism set to 1

**Answer:** B

**Explanation:**

To prevent two Terraform runs from changing the same state file simultaneously, state locking is used. State locking ensures that when one Terraform operation is running, others will be blocked from making changes to the same state, thus preventing conflicts and data corruption. This is achieved by configuring the state backend to support locking, which will lock the state for all operations that could write to the state.

References = This information is supported by Terraform's official documentation, which explains the importance of state locking and how it can be configured for different backends to prevent concurrent state modifications .

**NEW QUESTION 44**

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 48**

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 51**

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 56**

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 59**



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 61**

When does Sentinel enforce policy logic during a Terraform Cloud run?

- A. Before the plan phase
- B. During the plan phase
- C. Before the apply phase
- D. After the apply phase

**Answer:** C

**Explanation:**

Sentinel policies are checked after the plan stage of a Terraform run, but before it can be confirmed or the terraform apply is executed<sup>3</sup>. This allows you to enforce rules on your infrastructure before it is created or modified.

**NEW QUESTION 65**

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 70**

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 72**

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 75

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. References = 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 76

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 77

Which of the following arguments are required when declaring a Terraform output?

- A. value
- B. description
- C. default
- D. sensitive

**Answer:** A

#### Explanation:

When declaring a Terraform output, the value argument is required. Outputs are a way to extract information from Terraform-managed infrastructure, and the value argument specifies what data will be outputted. While other arguments like description and sensitive can provide additional context or security around the output, value is the only mandatory argument needed to define an output. References = The requirement of the value argument for outputs is specified in Terraform's official documentation, which provides guidelines on defining and using outputs in Terraform configurations.

#### NEW QUESTION 78

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 83**

Which of the following is not a valid Terraform variable type?

- A. list
- B. array
- C. nap
- D. string

**Answer:** B

**Explanation:**

This is not a valid Terraform variable type. The other options are valid variable types that can store different kinds of values<sup>2</sup>.

**NEW QUESTION 87**

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

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

**Answer:** BC

**Explanation:**

These are two ways to produce a list of the IDs from a list of objects that have an attribute id, using either a for expression or a splat expression syntax.

**NEW QUESTION 91**

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 96**

One remote backend configuration always maps to a single remote workspace.

- A. True
- B. False

**Answer:** A

**Explanation:**

The remote backend can work with either a single remote Terraform Cloud workspace, or with multiple similarly-named remote workspaces (like networking-dev and networking-prod). The workspaces block of the backend configuration determines which mode it uses. To use a single remote Terraform Cloud workspace, set workspaces.name to the remote workspace's full name (like networking-prod). To use multiple remote workspaces, set workspaces.prefix to a prefix used in all of the desired remote workspace names. For example, set prefix = ??networking-?? to use Terraform cloud workspaces with names like networking-dev and networking-prod. This is helpful when mapping multiple Terraform CLI workspaces used in a single Terraform configuration to multiple Terraform Cloud workspaces<sup>3</sup>. However, one remote backend configuration always maps to a single remote workspace, either by name or by prefix. You cannot use both name and prefix in the same backend configuration, or omit both. Doing so will result in a configuration error<sup>3</sup>. References = [Backend Type: remote]<sup>3</sup>

#### NEW QUESTION 97

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 100

How would you output returned values from a child module in the Terraform CLI output?

- A. Declare the output in the root configuration
- B. Declare the output in the child module
- C. Declare the output in both the root and child module
- D. None of the above

**Answer:** C

#### Explanation:

To output returned values from a child module in the Terraform CLI output, you need to declare the output in both the child module and the root module. The child module output will return the value to the root module, and the root module output will display the value in the CLI. References = [Terraform Outputs]

#### NEW QUESTION 105

.....

## THANKS FOR TRYING THE DEMO OF OUR PRODUCT

Visit Our Site to Purchase the Full Set of Actual Terraform-Associate-003 Exam Questions With Answers.

We Also Provide Practice Exam Software That Simulates Real Exam Environment And Has Many Self-Assessment Features. Order the Terraform-Associate-003 Product From:

<https://www.2passeasy.com/dumps/Terraform-Associate-003/>

## Money Back Guarantee

### **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