

HashiCorp

Exam Questions Terraform-Associate-003

HashiCorp Certified: Terraform Associate (003)



NEW QUESTION 1

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 2

HashiCorp Configuration Language (HCL) supports user-denned functions.

- A. True
- B. False

Answer: B

Explanation:

HashiCorp Configuration Language (HCL) does not support user-defined functions. You can only use the built-in functions that are provided by the language. The built-in functions allow you to perform various operations and transformations on values within expressions. The general syntax for function calls is a function name followed by comma-separated arguments in parentheses, such as max(5, 12, 9). You can find the documentation for all of the available built-in functions in the Terraform Registry or the

Packer Documentation, depending on which tool you are using. References = : Functions - Configuration Language | Terraform : Functions - Configuration Language | Packer

NEW QUESTION 3

Terraform providers are always installed from the Internet.

- A. True
- B. False

Answer: B

Explanation:

Terraform providers are not always installed from the Internet. There are other ways to install provider plugins, such as from a local mirror or cache, from a local filesystem directory, or from a network filesystem. These methods can be useful for offline or air-gapped environments, or for customizing the installation process. You can configure the provider installation methods using the provider_installation block in the CLI configuration file.

NEW QUESTION 4

You much 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 5

Why would you use the -replace flag for terraform apply?

- A. You want Terraform to ignore a resource on the next apply
- B. You want Terraform to destroy all the infrastructure in your workspace
- C. You want to force Terraform to destroy a resource on the next apply
- D. You want to force Terraform to destroy and recreate a resource on the next apply

Answer: D

Explanation:

The -replace flag is used with the terraform apply command when there is a need to explicitly force Terraform to destroy and then recreate a specific resource during the next apply. This can be necessary in situations where a simple update is insufficient or when a resource must be re-provisioned to pick up certain changes.

NEW QUESTION 6

While attempting to deploy resources into your cloud provider using Terraform, you begin to see some odd behavior and experience slow responses. In order to troubleshoot you decide to turn on Terraform debugging. Which environment variables must be configured to make Terraform's logging more verbose?

- A. TF_LOG_PAIRH
- B. TF_LOG
- C. TF_VAR_log_path
- D. TF_VAR_log_level

Answer: B

Explanation:

To make Terraform's logging more verbose for troubleshooting purposes, you must configure the TF_LOG environment variable. This variable controls the level of logging and can be set to TRACE, DEBUG, INFO, WARN, or ERROR, with TRACE providing the most verbose output. References = Detailed debugging instructions and the use of environment variables like TF_LOG for increasing verbosity are part of Terraform's standard debugging practices

NEW QUESTION 7

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

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

Answer: CD

Explanation:

These are features of Terraform Cloud, which is a hosted service that provides a web-based UI, remote state storage, remote operations, collaboration features, and more for managing your Terraform infrastructure.

NEW QUESTION 8

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 AMIs

Answer: ADE

Explanation:

Sentinel is a policy-as-code framework that allows you to define and enforce rules on your Terraform configurations, states, and plans¹. 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 infrastructure².
- 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 requirements³.
- You can enforce a list of approved AWS AMIs, 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 infrastructure⁴. 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 9

Which method for sharing Terraform configurations fulfills the following criteria:

- * 1. Keeps the configurations confidential within your organization
- * 2. Support Terraform's semantic version constraints
- * 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 10

As a developer, you want to ensure your plugins are up to date with the latest versions. Which Terraform command should you use?

- A. terraform refresh -upgrade
- B. terraform apply -upgrade
- C. terraform init -upgrade
- D. terraform providers -upgrade

Answer: C

Explanation:

This command will upgrade the plugins to the latest acceptable version within the version constraints specified in the configuration. The other commands do not have an - upgrade option.

NEW QUESTION 10

Running terraform fmt without any flags in a directory with Terraform configuration files check the formatting of those files without changing their contents.

- A. True
- B. False

Answer: B

Explanation:

Running terraform fmt without any flags in a directory with Terraform configuration files will not check the formatting of those files without changing their contents, but will actually rewrite them to a canonical format and style. If you want to check the formatting without making changes, you need to use the -check flag.

NEW QUESTION 15

What are some benefits of using Sentinel with Terraform Cloud/Terra form 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 17

What value does the Terraform Cloud private registry provide over the public Terraform Module Registry?

- A. The ability to share modules publicly with any user of Terraform
- B. The ability to restrict modules to members of Terraform Cloud or Enterprise organizations
- C. The ability to tag modules by version or release
- D. The ability to share modules with public Terraform users and members of Terraform Cloud Organizations

Answer: B

Explanation:

The Terraform Cloud private registry provides the ability to restrict modules to members of Terraform Cloud or Enterprise organizations. This allows you to share modules within your organization without exposing them to the public. The private registry also supports importing modules from your private VCS repositories. The public Terraform Module Registry, on the other hand, publishes modules from public Git repositories and makes them available to any user of Terraform. References = : Private Registry - Terraform Cloud : Terraform Registry - Provider Documentation

NEW QUESTION 18

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 23

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 25

You are writing a child Terraform module that provisions an AWS instance. You want to reference the IP address returned by the child module in the root configuration. You name the instance resource "main".

Which of these is the correct way to define the output value?

A)

```
output "instance_ip_addr" {  
    return aws_instance.main.private_ip  
}
```

B)

```
output "aws_instance.instance_ip_addr" {  
    return aws_instance.main.private_ip  
}
```

C)

```
output "aws_instance.instance_ip_addr" {  
    value = ${main.private_ip}  
}
```

D)

```
output "instance_ip_addr" {  
    value = aws_instance.main.private_ip  
}
```

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

Answer: D

NEW QUESTION 30

Which of these are secure options for storing secrets for connecting to a Terraform remote backend? Choose two correct answers.

- A. A variable file
- B. Defined in Environment variables
- C. Inside the backend block within the Terraform configuration
- D. Defined in a connection configuration outside of Terraform

Answer: BD

Explanation:

Environment variables and connection configurations outside of Terraform are secure options for storing secrets for connecting to a Terraform remote backend. Environment variables can be used to set values for input variables that contain secrets, such as backend access keys or tokens. Terraform will read environment variables that start with TF_VAR_ and match the name of an input variable. For example, if you have an input variable called backend_token, you can set its value with the environment variable TF_VAR_backend_token1. Connection configurations outside of Terraform are files or scripts that provide credentials or other information for Terraform to connect to a remote backend. For example, you can use a credentials file for the S3 backend2, or a shell script for the HTTP backend3. These files or scripts are not part of the Terraform configuration and can be stored securely in a separate location. The other options are not secure for storing secrets. A variable file is a file that contains values for input variables. Variable files are usually stored in the same directory as the Terraform configuration or in a version control system. This exposes the secrets to anyone who can access the files or the repository. You should not store secrets in variable files1. Inside the backend block within the Terraform configuration is where you specify the type and settings of the remote backend. The backend block is part of the Terraform configuration and is usually stored in a version control system. This exposes the secrets to anyone who can access the configuration or the repository. You should not store secrets in the backend block4. References = [Terraform Input Variables]1, [Backend Type: s3]2, [Backend Type: http]3, [Backend Configuration]4

NEW QUESTION 31

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 35

You are using a networking module in your Terraform configuration with the name label my-network. In your main configuration you have the following code:

```
output "net_id" {  
  value = module.my_network.vnet_id  
}
```

When you run terraform validate, you get the following error:

```
Error: Reference to undeclared output value  
  
on main.tf line 12, in output "net_id":  
12:   value = module.my_network.vnet_id
```

What must you do to successfully retrieve this value from your networking module?

- A. Change the reference value to my-network,outputs,vmet_id
- B. Define the attribute vmet_id as a variable in the networking modeule
- C. Define the attribute vnet_id as an output in the networking module
- D. Change the reference value module.my,network,outputs,vnet_id

Answer: C

Explanation:

This is what you must do to successfully retrieve this value from your networking module, as it will expose the attribute as an output value that can be referenced by other modules or resources. The error message indicates that the networking module does not have an output value named vnet_id, which causes the reference to fail.

NEW QUESTION 36

What does state locking accomplish?

- A. Prevent accidental Prevent accident deletion of the state file
- B. Blocks Terraform commands from modifying, the state file
- C. Copies the state file from memory to disk
- D. Encrypts any credentials stored within the state file

Answer: B

Explanation:

This is what state locking accomplishes, by preventing other users from modifying the state file while a Terraform operation is in progress. This prevents conflicts and data loss.

NEW QUESTION 38

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 42

Which Terraform command checks that your configuration syntax is correct?

- A. terraform validate
- B. terraform init
- C. terraform show
- D. terraform fmt

Answer: A

Explanation:

The terraform validate command is used to check that your Terraform configuration files are syntactically valid and internally consistent. It is a useful command for ensuring your Terraform code is error-free before applying any changes to your infrastructure.

NEW QUESTION 47

A provider configuration block is required in every Terraform configuration.

Example:

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

- A. True
- B. False

Answer: B

Explanation:

A provider configuration block is not required in every Terraform configuration. A provider configuration block can be omitted if its contents would otherwise be empty. Terraform assumes an empty default configuration for any provider that is not explicitly configured. However, some providers may require some configuration arguments (such as endpoint URLs or cloud regions) before they can be used. A provider's documentation should list which configuration arguments it expects. For providers distributed on the Terraform Registry, versioned documentation is available on each provider's page, via the Documentation link in the provider's header¹. References = [Provider Configuration]¹

NEW QUESTION 49

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 location⁴. The other commands are not relevant for this task.

NEW QUESTION 51

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 52

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 53

Which Terraform collection type should you use to store key/value pairs?

- A. Set
- B. Map
- C. Tuple
- D. list

Answer: B

Explanation:

The Terraform collection type that should be used to store key/value pairs is map. A map is a collection of values that are accessed by arbitrary labels, called keys.

The keys and values can be of any type, but the keys must be unique within a map. For example, var = { key1 = "value1", key2 = "value2" } is a map with two key/value pairs. Maps are useful for grouping related values together, such as configuration options or metadata. References = [Collection Types], [Map Type Constraints]

NEW QUESTION 58

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

- A. When you change a Terraform-managed resource via the Azure Cloud Console, Terraform updates the state file to reflect the change during the next plan or apply
- B. Changing resources via the Azure Cloud Console records the change in the current state file
- C. When you change a resource via the Azure Cloud Console, Terraform records the changes in a new state file
- D. Changing resources via the Azure Cloud Console does not update current state file

Answer: AD

Explanation:

Terraform state is a representation of the infrastructure that Terraform manages. Terraform uses state to track the current status of the resources it creates and to plan future changes. However, Terraform state is not aware of any changes made to the resources outside of Terraform, such as through the Azure Cloud Console, the Azure CLI, or the Azure API. Therefore, changing resources via the Azure Cloud Console does not update the current state file, and it may cause inconsistencies or conflicts with Terraform's desired configuration. To avoid this, it is recommended to manage resources exclusively through Terraform or to use the terraform import command to bring existing resources under Terraform's control.

When you change a Terraform-managed resource via the Azure Cloud Console, Terraform does not immediately update the state file to reflect the change.

However, the next time you run terraform plan or terraform apply, Terraform will compare the state file with the actual state of the resources in Azure and detect any drifts or differences. Terraform will

then update the state file to match the current state of the resources and show you the proposed changes in the execution plan. Depending on the configuration and the change, Terraform may try to undo the change, modify the resource further, or recreate the resource entirely. To avoid unexpected or destructive changes,

it is recommended to review the execution plan carefully before applying it or to use the terraform

refresh command to update the state file without applying any changes.

References = Purpose of Terraform State, Terraform State, Managing State, Importing Infrastructure, [Command: plan], [Command: apply], [Command: refresh]

NEW QUESTION 59

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 63

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 65

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 66

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 configures². The other options are either missing the name attribute or using an invalid syntax.

NEW QUESTION 67

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 68

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 71

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 76

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 executed³. This allows you to enforce rules on your infrastructure before it is created or modified.

NEW QUESTION 78

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 80

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 82

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 83

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 88

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 90

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 91

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 93

How would you reference the "name" value of the second instance of this resource?

```
resource "aws_instance" "web" {  
  count = 2  
  name = "terraform-${count.index}"  
}
```

- A. aws_instance.web(2),name
- B. element(aws_instance.web, 2)
- C. aws_instance-web(1)
- D. aws_instance_web(1),name
- E. Aws_instance,web,* , name

Answer: D

Explanation:

In Terraform, when you use the count meta-argument, you can reference individual instances using an index. The indexing starts at 0, so to reference the "name" value of the second instance, you would use aws_instance.web[1].name. This syntax allows you to access the properties of specific instances in a list generated by the count argument.

References:

? Terraform documentation on count and accessing resource instances: Terraform Count

NEW QUESTION 97

When using multiple configuration of the same Terraform provider, what meta-argument must you include in any non-default provider configurations?

- A. Alias
- B. Id
- C. Depends_on
- D. name

Answer: A

Explanation:

This is the meta-argument that you must include in any non-default provider configurations, as it allows you to give a friendly name to the configuration and reference it in other parts of your code. The other options are either invalid or irrelevant for this purpose.

NEW QUESTION 98

What does the default "local" Terraform backend store?

- A. tfplan files
- B. State file
- C. Provider plugins
- D. Terraform binary

Answer: B

Explanation:

The default "local" Terraform backend stores the state file in a local file named terraform.tfstate, which can be used to track and manage the state of your infrastructure.

NEW QUESTION 103

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 VM
- 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 105

Which parameters does terraform import require? Choose two correct answers.

- A. Provider
- B. Resource ID
- C. Resource address
- D. Path

Answer: BC

Explanation:

These are the parameters that terraform import requires, as they allow Terraform to identify the existing resource that you want to import into your state file, and match it with the corresponding configuration block in your files.

NEW QUESTION 108

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 have created a main.tf Terraform configuration consisting of an application server, a database and a load balancer. You ran terraform apply and Terraform created all of the resources successfully.

Now you realize that you do not actually need the load balancer, so you run terraform destroy without any flags. What will happen?

- A. Terraform will prompt you to pick which resource you want to destroy
- B. Terraform will destroy the application server because it is listed first in the code
- C. Terraform will prompt you to confirm that you want to destroy all the infrastructure
- D. Terraform will destroy the main.tf file
- E. Terraform will immediately destroy all the infrastructure

Answer: C

Explanation:

This is what will happen if you run terraform destroy without any flags, as it will attempt to delete all the resources that are associated with your current working directory or workspace. You can use the -target flag to specify a particular resource that you want to destroy.

NEW QUESTION 112

Which of the following is not a benefit of adopting infrastructure as code?

- A. Versioning
- B. A Graphical User Interface
- C. Reusability of code
- D. Automation

Answer: B

Explanation:

Infrastructure as Code (IaC) provides several benefits, including the ability to version control infrastructure, reuse code, and automate infrastructure management. However, IaC is typically associated with declarative configuration files and does not inherently provide a graphical user interface (GUI). A GUI is a feature that may be provided by specific tools or platforms built on top of IaC principles but is not a direct benefit of IaC itself¹.

References = The benefits of IaC can be verified from the official HashiCorp documentation on ??What is Infrastructure as Code with Terraform??? provided by HashiCorp Developer¹.

NEW QUESTION 117

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 them¹.

NEW QUESTION 119

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 120

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 125

Which provider authentication method prevents credentials from being stored in the state file?

- A. Using environment variables
- B. Specifying the login credentials in the provider block
- C. Setting credentials as Terraform variables
- D. None of the above

Answer: D

Explanation:

None of the above methods prevent credentials from being stored in the state file. Terraform stores the provider configuration in the state file, which may include sensitive information such as credentials. This is a potential security risk and should be avoided if possible. To prevent credentials from being stored in the state file, you can use one of the following methods:

? Use environment variables to pass credentials to the provider. This way, the credentials are not part of the provider configuration and are not stored in the state file. However, this method may not work for some providers that require credentials to be set in the provider block.

? Use dynamic credentials to authenticate with your cloud provider. This way, Terraform Cloud or Enterprise will request temporary credentials from your cloud provider for each run and use them to provision your resources. The credentials are not stored in the state file and are revoked after the run is completed. This method is supported for AWS, Google Cloud Platform, Azure, and Vault. References = : [Sensitive Values in State] : Authenticate providers with dynamic credentials

NEW QUESTION 126

FILL IN THE BLANK

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

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

- A. Mastered
- B. Not Mastered

Answer: A

Explanation:

The name of the default file where Terraform stores the state is terraform.tfstate. This file contains a JSON representation of the current state of the infrastructure managed by Terraform. Terraform uses this file to track the metadata and attributes of the resources, and to plan and apply changes. By default, Terraform stores the state file locally in the same directory as the configuration files, but it can also be configured to store the state remotely in a backend. References = [Terraform State], [State File Format]

NEW QUESTION 127

How do you specify a module's version when publishing it to the public terraform Module Registry?

- A. Configuration it in the module's Terraform code
- B. Mention it on the module's configuration page on the Terraform Module Registry
- C. The Terraform Module Registry does not support versioning modules
- D. Tag a release in the associated repo

Answer: D

Explanation:

This is how you specify a module's version when publishing it to the public Terraform Module Registry, as it uses the tags from your version control system (such as GitHub or GitLab) to identify module versions. You need to use semantic versioning for your tags, such as v1.0.0.

NEW QUESTION 128

A Terraform provider is NOT responsible for:

- A. Exposing resources and data sources based on an API
- B. Managing actions to take based on resources differences
- C. Understanding API interactions with some service
- D. Provisioning infrastructure in multiple

Answer: D

Explanation:

This is not a responsibility of a Terraform provider, as it does not make sense grammatically or logically. A Terraform provider is responsible for exposing resources and data sources based on an API, managing actions to take based on resource differences, and understanding API interactions with some service.

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](#)