DUMPS PLANET

2024

HashiCorp

Terraform-Associate-003 Exam

HashiCorp Infrastructure Automation

Questions & Answers

(Full Version)

Thank you for Purchasing Terraform-Associate-003 Exam

➢ TOTAL QUESTIONS : 138

Question: 1

Terraform providers are part of the Terraform core binary.

A. True
B. False

Answer: B

Explanation:

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

Question: 2

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.

Question: 3

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

Question: 4

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

Question: 5

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.

Question: 6

Your DevOps team is currently using the local backend for your Terraform configuration. You would like to move to a remote backend to store the state file in a central location. Which of the following backends would not work?

A. Artifactory
B. Amazon S3
C. Terraform Cloud
D. Git

Answer: D

Explanation:

This is not a valid backend for Terraform, as it does not support locking or versioning of state files4. The other options are valid backends that can store state files in a central location.

Question: 7

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.

Question: 8

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]

Question: 9

How does Terraform determine dependencies between resources?

A. Terraform requires resource dependencies to be defined as modules and sourced in order
B. Terraform automatically builds a resource graph based on resources provisioners, special meta-parameters, and the stale file (if present}
C. Terraform requires resources in a configuration to be listed m the order they will be created to determine dependencies
D. Terraform requires all dependencies between resources to be specified using the depends_on parameter

Answer: B

Explanation:

This is how Terraform determines dependencies between resources, by using the references between them in the configuration files and other factors that affect the order of operations.

Question: 10

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.

Question: 11

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 infrastructure3.

Question: 12

_____backends support state locking.

A. All
B. No
C. Some
D. Only local

Answer: C
Explanation:

Some backends support state locking, which prevents other users from modifying the state file while a Terraform operation is in progress. This prevents conflicts and data loss. Not all backends support this feature, and you can check the documentation for each backend type to see if it does.

Question: 13

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.

Question: 14

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.

Question: 15

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 consistent3. To confirm that your infrastructure matches the Terraform state file, you need to use terraform plan or terraform apply with the -refresh-only option.

Question: 16

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.

Question: 17

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

A. The team is asked to manage a new application stack built on AWS-native services
B. The organization decides to expand into Azure wishes to deploy new infrastructure
C. The team is asked to build a reusable code based that can deploy resources into any AWS region
D. The DevOps team is tasked with automating a manual, web console-based provisioning.
Answer: B
Explanation:

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

Question: 18

Which of these ate 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: B, D

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

Question: 19

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

Question: 20

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.

Question: 21

A developer accidentally launched a VM (virtual machine) outside of the Terraform workflow and ended up with two servers with the same name. They don't know which VM Terraform manages but do have a list of all active VM IDs.
Which of the following methods could you use to discover which instance Terraform manages?

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

Answer: A

Explanation:

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

Question: 22

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.

Question: 23

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.

Question: 24

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.

Question: 25

What Terraform command always causes a state file to be updated with changes that might have been made outside of Terraform?

A. Terraform plan –refresh-only
B. Terraform show –json
C. Terraform apply –lock-false
D. Terraform plan target-state

Answer: A

Explanation:

This is the command that always causes a state file to be updated with changes that might have been made outside of Terraform, as it will only refresh the state file with the current status of the real resources, without making any changes to them or creating a plan.

Question: 26

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 workspaces3. 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 error3. References = [Backend Type: remote]3

Question: 27

Which of these statements about Terraform Cloud workspaces is false?

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

Answer: B

Explanation:

The statement that you must use the CLI to switch between workspaces is false. Terraform Cloud workspaces are different from Terraform CLI workspaces. Terraform Cloud workspaces are required and represent all of the collections of infrastructure in an organization. They are also a major component of role-based access in Terraform Cloud. You can grant individual users and user groups permissions for one or more workspaces that dictate whether they can manage variables, perform runs, etc. You can create, view, and switch between Terraform Cloud workspaces using the Terraform Cloud UI, the Workspaces API, or the Terraform Enterprise Provider5. Terraform CLI workspaces are optional and allow you to create multiple distinct instances of a single configuration within one working directory. They are useful for creating disposable environments for testing or experimenting without affecting your main or production environment. You can create, view, and switch between Terraform CLI workspaces using the terraform workspace command6. The other statements about Terraform Cloud workspaces are true. They have role-based access controls that allow you to assign permissions to users and teams based on their roles and responsibilities. You can create and manage roles using the Teams API or the Terraform Enterprise Provider7. Plans and applies can be triggered via version control system integrations that allow you to link your Terraform Cloud workspaces to your VCS repositories. You can configure VCS settings, webhooks, and branch tracking to automate your Terraform Cloud workflow8. They can securely store cloud credentials as sensitive variables that are encrypted at rest and only decrypted when needed. You can manage variables using the Terraform Cloud UI, the Variables API, or the Terraform Enterprise Provider9. References = [Workspaces]5, [Terraform CLI Workspaces]6, [Teams and Organizations]7, [VCS Integration]8, [Variables]9

Question: 28

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

A. Dynamic
B. For_each
C. Nesting
D. repeated.

Answer: A

Explanation:

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

Question: 29

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

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

Answer: A

Explanation:

The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style. This command applies a subset of the Terraform language style conventions, along with other minor adjustments for readability. Running this command on your configuration files before committing them to source control can help ensure consistency of style between different Terraform codebases, and can also make diffs easier to read. You can also use the -check and -diff options to check if the files are formatted and display the formatting changes respectively2. Running the terraform fmt command during the code linting phase of your CI/CD process can help automate this process and enforce the formatting standards for your team. References = [Command: fmt]2

Question: 30

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]

Question: 31

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.

Question: 32

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]

Question: 33

A terraform apply can not _____infrastructure.

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

Answer: D

Explanation:

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

Question: 34

How does Terraform manage most dependencies between resources?

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

Answer: A

Explanation:

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

Question: 35

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

A. True
B. False

Answer: B

Explanation:

It is not a best practice to store secret data in the same version control repository as your Terraform

configuration, as it could expose your sensitive information to unauthorized parties or compromise your security. You should use environment variables, vaults, or other mechanisms to store and provide secret data to Terraform.

Question: 36

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

A. terraform state list 'provider_type.name'
B. terraform state show 'provider_type.name'
C. terraform get 'provider_type.name'
D. terraform state list

Answer: B

Explanation:

The terraform state show command allows you to access all of the attributes and details of a resource managed by Terraform. You can use the resource address (e.g. provider_type.name) as an argument to show the information about a specific resource. The terraform state list command only shows the list of resources in the state, not their attributes. The terraform get command downloads and installs modules needed for the configuration. It does not show any information about resources. References = [Command: state show] and [Command: state list]

Question: 37

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.

Question: 38

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]

Question: 39

In a Terraform Cloud workpace linked to a version control repository speculative plan rum 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:

Question: 40

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: A, D

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]

Question: 41

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.

Question: 42

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]

Question: 43

You have used Terraform lo create an ephemeral development environment in the (loud 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 mil 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: C, D

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

Question: 44

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]

Question: 45

You should run terraform fnt 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

Question: 46

Where does the Terraform local backend store its state?

A. In the terraform file
B. In the /tmp directory
C. In the terraform,tfstate file
D. In the user's terraform,state file

Answer: C

Explanation:

This is where the Terraform local backend stores its state, by default, unless you specify a different file name or location in your configuration. The local backend is the simplest backend type that stores the state file on your local disk.

Question: 47

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

Question: 48

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

Question: 49

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.

Question: 50

You ate making changes to existing Terraform code to add some new infrastructure. When is the best time to run terraform validate?

A. After you run terraform apply so you can validate your infrastructure
B. Before you run terraform apply so you can validate your provider credentials
C. Before you run terraform plan so you can validate your code syntax
D. After you run terraform plan so you can validate that your state file is consistent with your infrastructure

Answer: C

Explanation:

This is the best time to run terraform validate, as it will check your code for syntax errors, typos, and missing arguments before you attempt to create a plan. The other options are either incorrect or unnecessary.

Question: 51

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

Question: 52

Terraform providers are part of the Terraform core binary.

A. True
B. False

Answer: B

Explanation:

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

Question: 53

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

Question: 54

You have deployed a new webapp with a public IP address on a cloud provider. However, you did not create any outputs for your code. What is the best method to quickly find the IP address of the resource you deployed?

A. In a new folder, use the terraform_remote_state data source to load in the state file, then write an output for each resource that you find the state file
B. Run terraform state list to find the name of the resource, then terraform state show to find the attributes including public IP address
C. Run terraform output ip_address to view the result
D. Run terraform destroy then terraform apply and look for the IP address in stdout

Answer: B

Explanation:

This is a quick way to inspect the state file and find the information you need without modifying anything5. The other options are either incorrect or inefficient.

Question: 55

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.

Question: 56

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

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

Answer: D

Explanation:

This is a secure way to protect sensitive data in the state file, as it will be encrypted at rest and in transit2. The other options are not recommended, as they could lead to data loss, errors, or security breaches.

Question: 57

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.

Question: 58

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.

Question: 59

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.

Answer: Terraform.tfstat

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]

Question: 60

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

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

Answer: E

Explanation:

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

Question: 61

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

Question: 62

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 loss1.

Question: 63

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

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

Answer: A

Explanation:

This is not a valid string function in Terraform. The other options are valid string functions that can manipulate strings in various ways2.

Question: 64

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: B, C

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.

Question: 65

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.

Question: 66

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

A. True
B. False

Answer: B

Explanation:

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

Question: 67

Which command must you first run before performing further Terraform operations in a working directory?

A. terraform import
B. terraform workspace
C. terraform plan
D. terraform init

Answer: D

Explanation:

terraform init is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It initializes a working directory containing Terraform configuration files and downloads any required providers and modules. The other commands are used for different purposes, such as importing existing resources, switching between workspaces, generating execution plans, etc.

Question: 68

Where can Terraform not load a provider from?

A. Plugins directory

B. Provider plugin chance
C. Official HashCrop Distribution on releases.hashcrop.com
D. Source code

Answer: D

Explanation:

This is where Terraform cannot load a provider from, as it requires a compiled binary file that implements the provider protocol. You can load a provider from a plugins directory, a provider plugin cache, or the official HashiCorp distribution on releases.hashicorp.com.

Question: 69

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]

Question: 70

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.

Question: 71

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.

Question: 72

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.Â ReferencesÂ = [AWS Systems Manager Parameter Store], [Terraform AWS Provider Issue #55], [Terraform Providers], [Terraform Input Variables]

Question: 73

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.

Question: 74

Which method for sharing Terraform configurations fulfills the following criteria:

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

A. Subfolder within a workspace
B. Generic git repository
C. Terraform Cloud private registry
D. Public Terraform module registry

Answer: C

Explanation:

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

• Keeps the configurations confidential within your organization

• Supports Terraform's semantic version constraints

• Provides a browsable directory

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

Question: 75

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.

Question: 76

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.

Question: 77

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

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

Answer: A

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]

Question: 79

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: B, C

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.

Question: 80

The_____determines how Terraform creates, updates, or delete resources.

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

Answer: C

Explanation:

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

Question: 81

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.

Question: 82

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.

Question: 83

What is the Terraform style convention for indenting a nesting level compared to the one above it?

A. With a tab
B. With two spaces
C. With four spaces
D. With three spaces

Answer: B

Explanation:

This is the Terraform style convention for indenting a nesting level compared to the one above it. The other options are not consistent with the Terraform style guide.

Question: 84

You add a new provider to your configuration and immediately run terraform apply in the CD using the local backend. Why does the apply fail?

A. The Terraform CD needs you to log into Terraform Cloud first
B. Terraform requires you to manually run terraform plan first
C. Terraform needs to install the necessary plugins first
D. Terraform needs you to format your code according to best practices first

Answer: C

Explanation:

The reason why the apply fails after adding a new provider to the configuration and immediately running terraform apply in the CD using the local backend is because Terraform needs to install the necessary plugins first. Terraform providers are plugins that Terraform uses to interact with various cloud services and other APIs. Each provider has a source address that determines where to download it from. When Terraform encounters a new provider in the configuration, it needs to run terraform init first to install the provider plugins in a local directory. Without the plugins, Terraform cannot communicate with the provider and perform the desired actions. References = [Provider Requirements], [Provider Installation]

Question: 85

A Terraform provider is NOT responsible for:

A. Exposing resources and data sources based on an APUI
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.

Question: 86

You want to define a single input variable to capture configuration values for a server. The values must represent memory as a number, and the server name as a string.
Which variable type could you use for this input?

A. List
B. Object
C. Map
D. Terraform does not support complex input variables of different types

Answer: B

Explanation:

This is the variable type that you could use for this input, as it can store multiple attributes of different types within a single value. The other options are either invalid or incorrect for this use case.

Question: 87

Which of these commands makes your code more human readable?

A. Terraform validate
B. Terraform output
C. Terraform show
D. Terraform fmt

Answer: D

Explanation:

The command that makes your code more human readable is terraform fmt. This command is used to rewrite Terraform configuration files to a canonical format and style, following the Terraform language style conventions and other minor adjustments for readability. The command is optional, opinionated, and has no customization options, but it is recommended to ensure consistency of style across different Terraform codebases. Consistency can help your team understand the code more quickly and easily, making the use of terraform fmt very important. You can run this command on your configuration files before committing them to source control or as part of your CI/CD pipeline. References = : Command: fmt : Using Terraform fmt Command to Format Your Terraform Code

Question: 88

If you update the version constraint in your Terraform configuration, Terraform will update your lock file the next time you run terraform Init.

A. True
B. False

Answer: A

Explanation:

If you update the version constraint in your Terraform configuration, Terraform will update your lock file the next time you runÂ terraform init3. This will ensure that you use the same provider versions across different machines and runs.

Question: 89

Terraform variable names are saved in the state file.

A. True
B. False

Answer: B

Explanation:

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

Question: 90

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.

Question: 91

What does state looking 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.

Question: 92

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.

Question: 93

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

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

Answer: D

Explanation:

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

Question: 94

What kind of configuration block will create an infrastructure object with settings specified within the block?

A. provider
B. state
C. data
D. resource

Answer: D

Explanation:

This is the kind of configuration block that will create an infrastructure object with settings specified within the block. The other options are not used for creating infrastructure objects, but for configuring providers, accessing state data, or querying data sources.

Question: 95

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 â€œenvironmentsâ€• (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.Â ReferencesÂ =Â WorkspacesÂ andÂ How to Use Terraform Workspaces

Question: 96

You cannot install third party plugins using terraform init.

A. True
B. False

Answer: B

Explanation:

You can install third party plugins using terraform init, as long as you specify the plugin directory in your configuration or as a command-line argument. You can also use the terraform providers mirror command to create a local mirror of providers from any source.

Question: 97

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 values2.

Question: 98

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. Execute terraform destroy and select the newly-created VM.
B. Delete the Terraform state file and execute terraform apply.
C. The Terraform state file only contains the one new VM. Execute terraform destroy.
D. 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.

Question: 99

Which are forbidden actions when the terraform state file is locked? Choose three correct answers.

A. Terraform state list
B. Terraform destroy
C. Terraform validate
D. Terraform validate
E. Terraform for
F. Terraform apply

Answer: B, C, F

Explanation:

The terraform state file is locked when a Terraform operation that could write state is in progress. This prevents concurrent state operations that could corrupt the state. The forbidden actions when the state file is locked are those that could write state, such as terraform apply, terraform destroy, terraform refresh, terraform taint, terraform untaint, terraform import, and terraform state *. The terraform validate command is also forbidden, because it requires an initialized working directory with the state file. The allowed actions when the state file is locked are those that only read state, such as terraform plan, terraform show, terraform output, and terraform console. References = [State Locking] and [Command: validate]

Question: 100

Which of the following ate advantages of using infrastructure as code (IaC) instead of provisioning with a graphical user interface (GUI)? Choose two correct answers.

A. Lets you version, reuse, and share infrastructure configuration
B. Provisions the same resources at a lower cost
C. Secures your credentials
D. Reduces risk of operator error
E. Prevents manual modifications to your resources

Answer: A, D

Explanation:

• It lets you version, reuse, and share infrastructure configuration as code files, which can be stored in a source control system and integrated with your CI/CD pipeline.

• It reduces risk of operator error by automating repetitive tasks and ensuring consistency across environments. IaC does not necessarily provision resources at a lower cost, secure your credentials, or prevent manual modifications to your resources - these depend on other factors such as your cloud provider, your security practices, and your access policies.

Question: 101

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.

Question: 102

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.

Question: 103

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.

Question: 104

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]

Question: 105

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

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

Answer: B, C

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.

Question: 106

A developer on your team is going lo leaf down an existing deployment managed by Terraform and deploy a new one. However, there is a server resource named aws instant.ubuntu[l] 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 destory 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
, where
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

Question: 107

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.

Question: 108

Which command lets you experiment with terraform expressions?

A. Terraform console
B. Terraform validate
C. Terraform env
D. Terraform test

Answer: A

Explanation:

This is the command that lets you experiment with Terraform expressions, by providing an interactive console that allows you to evaluate expressions and see their results. You can use this command to test your expressions before using them in your configuration files.

Question: 109

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]

Question: 110

When using a remote backend or terraform Cloud integration, where does Terraform save resource sate?

A. In an environment variable
B. On the disk
C. In the remote backend or Terraform Cloud
D. In memory

Answer: C

Explanation:

This is where Terraform saves resource state when using a remote backend or Terraform Cloud integration, as it allows you to store and manage your state file in a remote location, such as a cloud storage service or Terraform Cloud's servers. This enables collaboration, security, and scalability for your Terraform infrastructure.

Question: 111

A module can always refer to all variables declared in its parent module.

A. True
B. False

Answer: B

Explanation:

A module cannot always refer to all variables declared in its parent module, as it needs to explicitly declare input variables and assign values to them from the parent module's arguments. A module cannot access the parent module's variables directly, unless they are passed as input arguments.

Question: 112

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: A, D

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

Question: 113

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.

Question: 114

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 location3. The other commands are not relevant for this task.

Question: 115

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

A. Tru
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]

Question: 116

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]

Question: 117

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: A, B, D

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.

Question: 118

What is the workflow for deploying new infrastructure with Terraform?

A. Write Terraform configuration, run terraform init to initialize the working directory or workspace, and run terraform apply
B. Write Terraform configuration, run terraform show to view proposed changes, and terraform apply to create new infrastructure
C. Write Terraform configuration, run terraform apply to create infrastructure, use terraform validate to confirm Terraform deployed resources correctly

D. Write Terraform configuration, run terraform plan to initialize the working directory or workspace, and terraform apply to create the infrastructure

Answer: A

Explanation:

This is the workflow for deploying new infrastructure with Terraform, as it will create a plan and apply it to the target environment. The other options are either incorrect or incomplete.

Question: 119

In Terraform HCL, an object type of object({name=string, age-number}) would match this value.

A)

```
{
  name = "John"
  age = fifty two
}
```

B)

```
{
  name = "John"
  age = 52
}
```

C)

```
{
  name =  John
  age = fifty two
}
```

D)

```
{
  name =  John
  age = 52
}
```

A. Option A

B. Option B
C. Option C
D. Option D

Answer: B

Question: 120

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

Question: 121

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]

Question: 122

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

A. Source = “module/consul’’
B. Source = ‘’githhub.comicrop/example’’
C. Source =’’git@github.com:hasicrop/example.git’’
D. Source = ‘’hasicrop/consul/aws’’

Answer: A

Explanation:

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

Question: 123

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

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

Answer: C

Explanation:

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

Question: 124

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.

Question: 125

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.

Question: 126

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 release. like 4.1
D. 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.

Question: 127

You have created a main.tf Terraform configuration consisting of an application server, a database and a

load balanced. 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.

Question: 128

You ate 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
 }

 "datadog" {
   api_key = var.datadog_api_key
   app_key = var.datadog_app_key
 }
}
```

A. Option A
B. Option B
C. Option C

Answer: B

Explanation:

Option B 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.

Question: 129

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

Question: 130

Terraform providers are part of the Terraform core binary.

A. True
B. False

Answer: B

Explanation:

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

Question: 131

Which of these ate 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: C, D

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.

Question: 132

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

A. Plan
B. Apply
C. Import
D. Init
E. Validate

Answer: B, D

Explanation:

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

Question: 133

The terraform.tfstate file always matches your currently built infrastructure.

A. True
B. False

Answer: B

Question: 134

You can configure Terraform to log to a file using the TF_LOG environment variable.

A. True
B. False

Answer: A

Explanation:

You can configure Terraform to log to a file using the TF_LOG environment variable. This variable can be set to one of the log levels: TRACE, DEBUG, INFO, WARN or ERROR. You can also use the TF_LOG_PATH environment variable to specify a custom log file location. References = : Debugging Terraform

Question: 135

Which is the best way to specify a tag of v1.0.0 when referencing a module stored in Git (for example. Git::https://example.com/vpc.git)?

A. Append pref=v1.0.0 argument to the source path
B. Add version = “1.0.0” parameter to module block
C. Nothing modules stored on GitHub always default to version 1.0.0

Answer: A

Explanation:

The best way to specify a tag of v1.0.0 when referencing a module stored in Git is to append ?ref=v1.0.0 argument to the source path. This tells Terraform to use a specific Git reference, such as a branch, tag, or commit, when fetching the module source code. For example, source = "git::https://example.com/vpc.git?ref=v1.0.0". This ensures that the module version is consistent and reproducible across different environments. References = [Module Sources], [Module Versions]

Question: 136

What does Terraform use the .terraform.lock.hc1 file for?

A. There is no such file
B. Tracking specific provider dependencies
C. Preventing Terraform runs from occurring
D. Storing references to workspaces which are locked

Answer: B

Explanation:

The .terraform.lock.hcl file is a new feature in Terraform 0.14 that records the exact versions of each provider used in your configuration. This helps ensure consistent and reproducible behavior across different machines and runs.

Question: 137

When should you use the force-unlock command?

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

Answer: B

Explanation:

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

Question: 138

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.

Question: 138

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

A. remote-exec
B. null-exec
C. local-exec
D. file

Answer: A

Explanation:

"The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource."
https://www.terraform.io/language/resources/provisioners/local-exec
"The remote-exec provisioner invokes a script on a remote resource after it is created."

Thank You for Being Our Valued Customer
We Hope You Enjoy Your Purchase
HashiCorp Terraform-Associate-003 Exam Question & Answers
HashiCorp Certified: Terraform Associate (003) Exam