#5 Terraform Associate Certification 003 - Results

Back to result overview

Attempt 1

All domains

- **57 all**
- **40 correct**
- **17 incorrect**
- **0 skipped**
- **0 marked**

Collapse all questions

Question 1Correct

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

terraform state list ‘provider_type.name’

Explanation

The command 'terraform state list' is used to list all the resources managed by Terraform, not to access the attributes and details of a specific resource. It provides a list of resource instances, not the detailed information of a specific resource.

Your answer is correct

terraform state show ‘provider_type.name’

Explanation

The correct command to access all the attributes and details of a specific resource managed by Terraform is 'terraform state show 'provider_type.name''. This command displays all the details and attributes of the specified resource, allowing you to view the current state of that resource.

terraform get ‘provider_type.name’

Explanation

The command 'terraform get 'provider_type.name'' is used to download and update modules from the Terraform registry or other sources. It is not used to access the attributes and details of a specific resource managed by Terraform.

terraform state list

Explanation

The command 'terraform state list' is used to list all the resources managed by Terraform, not to access the attributes and details of a specific resource. It provides a list of resource instances, not the detailed information of a specific resource.

Overall explanation

Correct Answer: terraform state show ‘provider_type.name’

Explanation:

The command `terraform state show` is used to access all the attributes and details of a resource managed by Terraform. This command provides detailed information about the current state of the specified resource, including its properties and metadata, as tracked in the Terraform state file.

- `terraform state list` only lists the resources in the state file, without providing detailed information about them.
- `terraform get` is used to download modules into your working directory, not to show details of resources.
- `terraform state list 'provider_type.name'` is incorrect because `state list` does not accept specific resource names as arguments for detailed display.

Therefore, `terraform state show 'provider_type.name'` is the correct command to get detailed information about a specific resource managed by Terraform.

Resources

Terraform state show

Question 2Correct

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

True

Explanation

This statement is incorrect. The `terraform validate` command is used to check the syntax and validity of the Terraform configuration files, ensuring that they are properly formatted and can be successfully processed by Terraform. It does not compare the infrastructure to the Terraform state file.

Your answer is correct
False
Explanation
This statement is correct. The `terraform validate` command does not confirm that the infrastructure matches the Terraform state file. It is specifically used to validate the syntax and structure of the Terraform configuration files, helping to catch errors before applying changes to the infrastructure.
Overall explanation
Correct Answer: False

Explanation:

`terraform validate` is used to check the syntax and structure of the Terraform configuration files (.tf files). It ensures that the configuration is syntactically valid and that the configuration can be interpreted by Terraform, but it does **not** verify whether the infrastructure matches the Terraform state file.

To check if the infrastructure matches the state file, you would use commands like `terraform plan` or `terraform refresh`. These commands compare the current infrastructure (as known by Terraform state) with the configuration and show any differences.

Thus, `terraform validate` does not compare the infrastructure with the state file; it only validates the configuration files.

Question 3Correct
A senior admin accidentally deleted some of your cloud instances. What does Terraform do when you run terraform apply?

Build a completely brand new set of infrastructure
Explanation
Terraform does not build a completely brand new set of infrastructure when you run terraform apply. It aims to make the current state match the desired state defined in the configuration files, so it will only make necessary changes to achieve that.

Tear down the entire workspace infrastructure and rebuild it
Explanation
Terraform does not tear down the entire workspace infrastructure and rebuild it when you run terraform apply. It focuses on making incremental changes to the infrastructure based on the configuration files, rather than starting from scratch.

Your answer is correct
Rebuild only the instances that were deleted
Explanation
When you run terraform apply after some cloud instances have been accidentally deleted, Terraform will rebuild only the instances that were deleted. It compares the current state with the desired state and only makes changes to bring the infrastructure back to the desired state.

Stop and generate an error message about the missing instances
Explanation
Terraform does not stop and generate an error message about the missing instances when you run terraform apply. Instead, it will identify the differences between the current state and the desired state, and take actions to reconcile those differences, such as recreating the deleted instances.
Overall explanation
Correct Answer: Rebuild only the instances that were deleted

Explanation:

When you run `terraform apply` after a resource has been manually deleted outside of Terraform, Terraform will notice that the resource is missing from the cloud provider and will attempt to recreate it to bring the infrastructure back in sync with the configuration and state. It will only rebuild the missing resources and not the entire infrastructure.

Terraform keeps track of the current state of the infrastructure in the state file. If a resource (like a cloud instance) is deleted manually, it will be detected as "missing" when you run `terraform apply`, and Terraform will recreate just that resource without touching the rest of the infrastructure.

Thus, Terraform doesn't tear down the entire infrastructure or build new resources unnecessarily. It specifically targets the missing resources and attempts to rebuild them.

Question 4Correct

`terraform init` creates an example main.tf file in the current directory.

True

Explanation

This statement is incorrect. The `terraform init` command initializes a Terraform working directory by downloading provider plugins and modules as specified in the configuration files. It does not create an example `main.tf` file in the current directory.

Your answer is correct

False

Explanation

This statement is correct. The `terraform init` command does not create an example `main.tf` file in the current directory. Its primary purpose is to initialize a Terraform working directory by downloading necessary plugins and modules.

Overall explanation

Correct Answer: False

Explanation:

The `terraform init` command does not create an example `main.tf` file in the current directory. Instead, `terraform init` initializes a Terraform working directory by downloading the necessary provider plugins and setting up the backend configuration if defined. It prepares the directory for Terraform commands to be run but does not generate any example configuration files like `main.tf`.

The `main.tf` file, if required, needs to be manually created or can be created by the user as part of the infrastructure configuration.

Resources

[Terraform init]

Question 5Incorrect

Which argument helps prevent unexpected updates when calling Terraform Registry modules?

Your answer is incorrect

count

Explanation

The `count` argument in Terraform is used to create multiple instances of a resource or module based on a specified number. It does not directly help prevent unexpected updates when calling Terraform Registry modules.

source

Explanation

The `source` argument in Terraform is used to specify the location of the module or provider. While it is important for defining where to fetch the module from, it does not specifically help prevent unexpected updates when calling Terraform Registry modules.

Correct answer

version

Explanation

The `version` argument in Terraform is crucial for specifying the version of the module to use. By pinning the version, you can prevent unexpected updates when calling Terraform Registry modules, ensuring that the module remains consistent and predictable.

lifecycle

Explanation

The `lifecycle` argument in Terraform is used to control the behavior of resources during Terraform operations. While it can be used to manage resource lifecycle events, it does not directly help prevent unexpected updates when calling Terraform Registry modules.

Overall explanation

Correct Answer: version

Explanation:

The `version` argument is used to specify which version of a module to use when calling Terraform Registry modules. By specifying the version, you can prevent unexpected updates or changes to your infrastructure due to new module releases. Without a version constraint, Terraform might automatically pull the latest version of a module, which could introduce breaking changes or unexpected behavior. Specifying the version ensures consistency and stability across environments.

Other arguments like `count` or `source` are unrelated to controlling module versions, while `lifecycle` controls the behavior of resources, not the module versioning.

Question 6Correct

What is the Terraform resource name of the following resource block?

```
    resource "azurerm_resource_group" "dev" {
        name = "test"
        location = "westus"
    }
```

azurerm_resource_group

Explanation

The resource name in the given resource block is "azurerm_resource_group". This is the name of the Terraform resource being defined in the block, not the resource group name itself.

azurerm

Explanation

"azurerm" is not the resource name in the given resource block. It is the provider name specified at the beginning of the resource block to indicate that the resource is managed by the Azure provider in Terraform.

test

Explanation

"test" is not the resource name in the given resource block. It is the value assigned to the "name" attribute of the resource group being created, not the resource name itself.

Your answer is correct

dev

Explanation

"dev" is the correct Terraform resource name in the given resource block. In Terraform, the resource name is specified after the provider and separated by a space, as shown in the resource block provided.

Overall explanation

Correct Answer: dev

Explanation:

In the given resource block, the Terraform resource name is specified after the resource type. The syntax for defining a resource in Terraform is:

```
        resource "resource_type" "resource_name" {
         # resource properties
        }
```

In this case, `"azurerm_resource_group"` is the resource type, and `"dev"` is the resource name. The resource name is used to uniquely identify the instance of that resource within your Terraform configuration. Therefore, `dev` is the correct resource name.

`azurerm_resource_group` is the resource type, not the name. `test` is the value assigned to the `name` property, and `azurerm` is the provider.

Resources

[Resource block syntax]()

Question 7Correct

When do you need to explicitly execute `terraform refresh-only`

Before every terraform plan

Explanation

Executing `terraform refresh-only` before every `terraform plan` is not necessary. The `terraform plan` command already automatically refreshes the state to reflect the current infrastructure before generating an execution plan.

Before every terraform apply

Explanation

Running `terraform refresh-only` before every `terraform apply` is not required. The `terraform apply` command automatically refreshes the state to ensure it has the latest information before making any changes to the infrastructure.

Before every terraform import

Explanation

There is no need to explicitly run `terraform refresh-only` before every `terraform import`. The `terraform import` command already refreshes the state to reflect the imported resource's current state.

Your answer is correct

None of the above

Explanation

The correct choice is "None of the above" because there is no specific requirement to execute `terraform refresh-only` before any of the mentioned commands (`terraform plan`, `terraform apply`, `terraform import`). The `terraform refresh-only` command is used independently to update the state file without making any changes to the infrastructure.

Overall explanation

Correct Answer: None of the above

Explanation: `terraform refresh` is used to update the Terraform state with the latest information about the infrastructure. This is generally not something that needs to be executed manually unless you specifically want to refresh the state outside of a normal plan or apply process.

Terraform automatically refreshes the state when running `terraform plan` and `terraform apply`. So, you typically don't need to explicitly run `terraform refresh` or `terraform refresh-only` before those commands. `terraform refresh-only` is specifically used when you want to manually refresh the state without making any changes to the infrastructure or applying any updates.

Resources

Terraform refresh

Question 8Incorrect

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

Secures your credentials

Explanation

Securing your credentials is not a direct advantage of using infrastructure as code (IaC) over a graphical user interface (GUI). While IaC can help manage and secure credentials more effectively, this specific advantage does not differentiate between IaC and GUI provisioning methods.

Correct selection

Lets your version, reuse, and share infrastructure configuration

Explanation

Allowing versioning, reusability, and sharing of infrastructure configurations is a key advantage of using infrastructure as code (IaC) over a graphical user interface (GUI). This enables teams to collaborate more efficiently, track changes, and maintain consistency across environments.

Provisions the same resources at a lower cost

Explanation

Provisioning the same resources at a lower cost is not a direct advantage of using infrastructure as code (IaC) over a graphical user interface (GUI). While IaC can help optimize resource usage and reduce manual effort, cost savings are not inherently tied to the choice of provisioning method.

Your selection is correct

Reduces risk of operator error

Explanation

Reducing the risk of operator error is a significant advantage of using infrastructure as code (IaC) over a graphical user interface (GUI). By defining infrastructure configurations in code, you can avoid manual mistakes, ensure consistency, and automate deployment processes, leading to fewer errors.

Your selection is incorrect

Prevents manual modifications to your resources

Explanation

Preventing manual modifications to your resources is not a direct advantage of using infrastructure as code (IaC) over a graphical user interface (GUI). While IaC can help enforce consistency and prevent ad-hoc changes, this specific advantage does not solely apply to the choice of provisioning method.

Overall explanation

Correct Answer: **Lets you version, reuse, and share infrastructure configuration** and **Reduces risk of operator error**

Explanation:

- **Lets you version, reuse, and share infrastructure configuration**: One of the main advantages of Infrastructure as Code (IaC) is that it allows you to define your infrastructure configuration in code, making it possible to version control, reuse, and share the configuration easily. This is not possible when using a GUI-based approach, where changes are often manual and can't be easily replicated or tracked.
- **Reduces risk of operator error**: With IaC, configurations are written as code, so the process becomes automated and reproducible. This reduces the likelihood of human errors that might occur when manually provisioning infrastructure through a GUI. Since the infrastructure is defined and executed in code, there is less room for mistakes related to inconsistent configurations, missed steps, or incorrect inputs.

Other options:

- **Secures your credentials**: This is not a direct advantage of IaC itself. While IaC allows you to manage and securely handle credentials (for example, through environment variables or secrets management), securing credentials is a separate concern, not automatically resolved by IaC.
- **Provisions the same resources at a lower cost**: While IaC can help automate resource provisioning, it does not necessarily lead to lower costs. Cost savings come from better resource management and optimization, which might be a result of using IaC but not a guaranteed outcome.
- **Prevents manual modifications to your resources**: While IaC reduces the risk of manual changes, it doesn't inherently prevent all manual modifications. However, you can enforce policies that prevent manual changes by integrating IaC with workflows that restrict direct access to infrastructure resources.

Question 9Correct

One cloud configuration always maps to a single remote workspace.

True

Explanation

This statement is incorrect. In Terraform, a single cloud configuration can map to multiple remote workspaces. This allows for better organization and management of infrastructure resources across different environments or projects.

Your answer is correct

False

Explanation

This statement is correct. In Terraform, a single cloud configuration does not always map to a single remote workspace. Multiple remote workspaces can be associated with a single cloud configuration, providing flexibility and scalability in managing infrastructure resources.

Overall explanation

Correct Answer: **False**

Explanation:

In Terraform, a workspace is essentially an isolated environment for managing infrastructure. A single Terraform configuration can be applied to multiple workspaces. Workspaces allow you to manage different environments (such as development, staging, and production) with the same configuration but separate state files. Therefore, a single cloud configuration can be associated with multiple remote workspaces, and they do not need to map to just one.

This flexibility allows teams to manage infrastructure across multiple environments while keeping the configuration consistent, reducing the need to maintain separate configuration files for each environment. Each workspace can have its own separate state, enabling environment-specific configurations and resources.

Resources

Workspaces

Question 10Correct

Multiple team members are collaborating on infrastructure using Terraform and want to format their Terraform code following standard Terraform-style convention.

How could they automatically ensure the code satisfies conventions?

Replace all tabs with spaces

Explanation

Replacing all tabs with spaces is a good practice for code readability, but it does not automatically ensure that the Terraform code follows standard Terraform-style conventions. This choice focuses on code formatting rather than enforcing conventions.

Terraform automatically formats configuration on terraform apply

Explanation

Terraform automatically formats configuration on terraform apply, but this does not specifically ensure that the code adheres to standard Terraform-style conventions. While it helps with code formatting, it does not guarantee that the code meets all the conventions.

Run terraform validate prior to executing terraform plan or terraform apply

Explanation

Running terraform validate prior to executing terraform plan or terraform apply is a good practice to check for syntax errors and validate the configuration. However, it does not enforce standard Terraform-style conventions. This choice focuses on validating the syntax rather than enforcing conventions.

Your answer is correct

Use terraform fmt

Explanation

Using terraform fmt is the correct choice because it automatically formats the Terraform code to adhere to standard Terraform-style conventions. This command ensures that the code follows the recommended formatting guidelines, making it easier for multiple team members to collaborate and maintain consistent code style.

Overall explanation

Correct Answer: Use terraform fmt

Explanation:

To ensure that Terraform configuration files are formatted consistently according to Terraform's standard style conventions, the **terraform fmt** command is used. This command automatically formats the code, ensuring that it adheres to the standard Terraform formatting guidelines, such as using two spaces for indentation and aligning blocks consistently.

- **Replace all tabs with spaces** is not sufficient by itself to enforce Terraform's style conventions. While Terraform requires spaces instead of tabs for indentation, **terraform fmt** is the proper tool to ensure the entire code is formatted correctly.
- **Terraform automatically formats configuration on terraform apply** is incorrect. Terraform does not automatically format your configuration files when running commands like `terraform apply`. You must manually run `terraform fmt` to format the code.
- **Run terraform validate prior to executing terraform plan or terraform apply** is not directly related to formatting. `terraform validate` checks the syntax and validity of the Terraform code, but it does not enforce style conventions or formatting.

By using `terraform fmt`, teams can ensure that all Terraform code is formatted consistently, which is essential when multiple people are collaborating on the same infrastructure.

Resources

terraform fmt

Question 11 Correct

Which backend does the Terraform CLI use by default?

Depends on the cloud provider configured

Explanation

The backend used by the Terraform CLI does not depend on the cloud provider configured. The default backend is determined by the Terraform configuration and can be overridden by explicitly specifying a different backend.

Remote

Explanation

The default backend used by the Terraform CLI is not a remote backend. While remote backends can be configured for storing state files remotely, the default backend is different.

Terraform Cloud

Explanation

Terraform Cloud is a specific type of remote backend that can be configured for storing state files and collaborating on Terraform projects. However, it is not the default backend used by the Terraform CLI.

Your answer is correct

Local

Explanation

The Terraform CLI uses the local backend by default. This means that the state file is stored on the local filesystem of the machine running Terraform. This is the default behavior unless a different backend is explicitly configured.

HTTP

Explanation

The Terraform CLI does not use an HTTP backend by default. While HTTP backends can be configured for storing state files remotely, the default backend used by the Terraform CLI is the local backend.

Overall explanation

Correct Answer: Local

Explanation:

By default, Terraform uses the **local** backend, which stores the Terraform state file (`terraform.tfstate`) on the local filesystem. This is the default behavior when no explicit backend configuration is provided in the Terraform configuration.

- **Depends on the cloud provider configured** is incorrect because the backend is not determined by the cloud provider but is instead specified by the user (e.g., using S3 for AWS, or GCS for Google Cloud).
- **Remote** is incorrect because while Terraform does support remote backends (like AWS S3, Azure Blob Storage, or Terraform Cloud), this is not the default backend; you have to explicitly configure remote backends.
- **Terraform Cloud** is also incorrect because while Terraform Cloud provides a powerful remote backend, it's not the default option. It must be set up separately by specifying a backend configuration.
- **HTTP** is incorrect as it is not a default backend and would require explicit configuration for any remote backend to interact over HTTP.

Thus, the **local** backend is used by default when Terraform is initialized and no other backend is specified.

Resources

Backend block configuration overview

Question 12 Correct

The Terraform CLI will print output values from a child module after running terraform apply.

True

Explanation

This statement is incorrect. The Terraform CLI does not automatically print output values from a child module after running terraform apply. Output values must be explicitly defined in the root module and accessed using the terraform output command.

Your answer is correct
False
Explanation
This statement is correct. The Terraform CLI does not automatically print output values from a child module after running terraform apply. Output values must be explicitly defined in the root module and accessed using the terraform output command.
Overall explanation
Correct Answer: False

Explanation:

By default, Terraform does not automatically print output values from a child module after running `terraform apply`. The `terraform apply` command will apply the changes and output the overall results, but the output values from a child module need to be explicitly referenced or called in the root module if you want them displayed.

To print the output values from a child module, you need to use `output` blocks in the root module that reference the child module's outputs. Here's an example:

```
hcl
Copy codeoutput "child_module_output" {
 value = module.child_module.some_output
}
```

This output block in the root module will allow you to see the output of the child module after running `terraform apply`. Without this, the outputs from the child module are not displayed.

- **Option "True"** is incorrect because child module outputs are not automatically printed unless explicitly referenced in the root module.

Question 13Correct
What does terraform refresh-only modify?
Your cloud infrastructure
Explanation
Terraform refresh-only does not modify your cloud infrastructure. It only updates the state file with the latest information from the actual resources in the cloud.
Your Terraform plan
Explanation
Terraform refresh-only does not modify your Terraform plan. It is specifically designed to update the state file without making any changes to the infrastructure.
Your Terraform configuration
Explanation
Terraform refresh-only does not modify your Terraform configuration. It focuses solely on updating the state file based on the current state of the resources in the cloud.
Your answer is correct
Your state file
Explanation
Terraform refresh-only modifies your state file by updating it with the latest information from the actual resources in the cloud. It does not make any changes to the infrastructure itself, but ensures that the state file accurately reflects the current state of the resources.
Overall explanation
Correct Answer: Your state file

Explanation: The `terraform refresh` command is used to reconcile the state file with the real-world infrastructure. It queries the infrastructure providers to determine the current state of resources and updates the Terraform state file accordingly. This operation does not modify any cloud resources, Terraform configurations, or plans; it only updates the state file to reflect the most recent resource statuses.

- "Your cloud infrastructure" is incorrect because `terraform refresh` does not create, modify, or destroy cloud resources.
- "Your Terraform plan" is incorrect because the `terraform refresh` command does not affect the execution plan directly. It simply updates the state file with current data, which can be used later when generating a plan.
- "Your Terraform configuration" is incorrect because the command does not modify the actual Terraform code; it only impacts the state file.

Question 14 Correct

What does terraform import do?

Your answer is correct

Imports existing resources into the state file

Explanation

Terraform import is used to bring existing resources under Terraform management by importing them into the state file. This allows Terraform to track and manage the existing resources as part of its configuration.

Imports all infrastructure from a given cloud provider

Explanation

Terraform import does not import all infrastructure from a cloud provider. It specifically focuses on importing existing resources into the state file for management, rather than importing all infrastructure.

Imports a new Terraform module

Explanation

Terraform import does not import a new Terraform module. It is used to bring existing resources into the state file, not to import new modules.

Imports clean copies of tainted resources

Explanation

Terraform import does not import clean copies of tainted resources. It is used to import existing resources into the state file, regardless of their current state.

None of the above

Explanation

None of the above choices accurately describe what terraform import does. The correct answer is that it imports existing resources into the state file for management.

Overall explanation

Correct Answer: Imports existing resources into the state file

Explanation: The `terraform import` command is used to bring existing resources into Terraform's state file. This command allows Terraform to manage resources that were created outside of Terraform, such as resources provisioned manually or by other tools. Importing a resource does not modify the resource itself, it simply allows Terraform to start managing it from that point forward.

- "Imports all infrastructure from a given cloud provider" is incorrect because `terraform import` only imports a specific resource, not all resources in an entire cloud provider.
- "Imports a new Terraform module" is incorrect because Terraform modules are added by referencing them in your configuration, not through `terraform import`.
- "Imports clean copies of tainted resources" is incorrect because `terraform import` is used for existing resources; tainted resources can be managed or recreated using `terraform taint`, not `terraform import`.

Question 15 Correct

Which of the following is the correct way to pass the value in the variable num_servers into a module with the input server?

servers = var(num_servers)

Explanation

This choice is incorrect because the correct syntax to reference a variable in Terraform is by using the `var.` prefix before the variable name. In this case, the variable `num_servers` should be referenced as `var.num_servers`.

$(var.num_servers)

Explanation

This choice is incorrect because the syntax `$(var.num_servers)` is not valid in Terraform for passing variable values into modules. The correct way to reference a variable is by using the `var.` prefix before the variable name.

servers = num_servers

Explanation

This choice is incorrect because simply using the variable name `num_servers` without the `var.` prefix is not the correct way to reference and pass the value of the variable into a module in Terraform. The correct syntax requires using `var.num_servers`.

Your answer is correct

servers = var.num_servers

Explanation

This choice is correct because it uses the correct syntax to reference and pass the value of the variable `num_servers` into a module with the input `server`. In Terraform, variables are referenced using the `var.` prefix followed by the variable name, so `servers = var.num_servers` is the appropriate way to pass the value.

Overall explanation

Correct Answer: servers = var.num_servers

Explanation:

In Terraform, when you pass values to a module, you use the syntax `var.<variable_name>`, where `<variable_name>` is the name of the variable defined in the root configuration or the parent module.

- "servers = var(num_servers)" is incorrect because the correct syntax to reference a variable is `var.<variable_name>`, not using parentheses.
- "$(var.num_servers)" is incorrect because the correct syntax for referencing variables in Terraform is `var.num_servers`, without the `$` sign.
- "servers = num_servers" is incorrect because you need to prefix the variable name with `var.` to access the value of a variable.

Question 16Correct

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

terraform destroy aws_instance.ubuntu[l]

Explanation

The command "terraform destroy" is used to destroy all the resources managed by Terraform, not to stop managing a specific resource. Using this command would result in the deletion of the aws_instance.ubuntu[1] resource along with all other resources.

terraform apply rm aws_instance.ubuntu[l]

Explanation

The command "terraform apply" is used to apply the changes specified in the Terraform configuration. Adding "rm" after "apply" does not represent a valid Terraform command. This command would not stop Terraform from managing the aws_instance.ubuntu[1] resource.

Your answer is correct

terraform state rm aws_instance.ubuntu[l]

Explanation

The command "terraform state rm" is used to remove a resource from the Terraform state, effectively telling Terraform to stop managing that specific resource. This is the correct command to use if the developer wants to keep the aws_instance.ubuntu[1] resource while tearing down the rest of the deployment.

terraform plan rm aws_instance.ubuntu[l]

Explanation

The command "terraform plan" is used to generate an execution plan for Terraform, showing what actions Terraform will take when applying the configuration. Adding "rm" after "plan" does not represent a valid Terraform command. This command would not stop Terraform from managing the aws_instance.ubuntu[1] resource.

Overall explanation

Correct Answer: terraform state rm aws_instance.ubuntu[1]

Explanation:

To tell Terraform to stop managing a specific resource, you use the `terraform state rm` command. This removes the resource from Terraform's state file, effectively telling Terraform to "forget" about the resource without actually destroying it in the cloud provider.

- "terraform destroy aws_instance.ubuntu[1]" is incorrect because `terraform destroy` would destroy the resource rather than stop managing it.
- "terraform apply rm aws_instance.ubuntu[1]" is incorrect because there is no `rm` command used in conjunction with `terraform apply`.
- "terraform plan rm aws_instance.ubuntu[1]" is incorrect because `terraform plan` is used to preview changes, not to modify the state directly. You cannot remove a resource using `terraform plan`.

Resources
state rm command

Question 17 Correct
Before you can use a remote backend, you must first execute terraform init.

Your answer is correct
True
Explanation
True. Before utilizing a remote backend in Terraform, it is necessary to initialize the configuration with `terraform init`. This command sets up the working directory and downloads any necessary plugins and modules, including the backend configuration for remote operations.

False
Explanation
False. This statement is incorrect. In order to use a remote backend in Terraform, the first step is indeed to execute `terraform init`. This command is essential for configuring the backend and preparing the workspace for remote operations.

Overall explanation
Correct Answer: True

Explanation:

Before you can use a remote backend, you must execute `terraform init`. This command initializes your Terraform working directory, downloads the necessary provider plugins, and configures the backend for storing the state remotely. The `terraform init` process sets up the backend configuration, whether it's for local or remote state storage, and prepares Terraform to manage your infrastructure.

- "False" is incorrect because initializing Terraform with `terraform init` is essential to configure the backend before any operations can be performed.

Question 18 Correct
What does running a terraform plan do?

Compares your Terraform code and local state file to the remote state file in a cloud provider and determines if any changes need to be made
Explanation
Running a terraform plan does not compare your Terraform code and local state file to the remote state file in a cloud provider. It compares the state file to your Terraform code to determine if any changes need to be made.

Imports all of your existing cloud provider resources to the state file
Explanation
Running a terraform plan does not import all of your existing cloud provider resources to the state file. It is a command used to preview the changes that Terraform will make to your infrastructure.

Installs all providers and modules referenced by configuration
Explanation
Running a terraform plan does not install all providers and modules referenced by configuration. It is a command that helps you understand the changes that Terraform will apply to your infrastructure.

Your answer is correct
Compares the state file to your Terraform code and determines if any changes need to be made

Explanation
Running a terraform plan compares the state file to your Terraform code and determines if any changes need to be made. It provides a preview of the actions that Terraform will take when you apply the configuration.
Overall explanation
Correct Answer: Compares the state file to your Terraform code and determines if any changes need to be made

Explanation:

When you run `terraform plan`, Terraform compares the current infrastructure (represented by the state file) with the desired infrastructure (defined in your Terraform configuration files). This comparison results in an execution plan that outlines what actions Terraform will take to align the state with the configuration—whether it needs to create, modify, or delete resources.

- The answer "Compares your Terraform code and local state file to the remote state file in a cloud provider and determines if any changes need to be made" is incorrect because Terraform does not compare local and remote state files during the `terraform plan` command. The comparison is between the state file (local or remote) and the configuration.
- The answer "Imports all of your existing cloud provider resources to the state file" is incorrect because `terraform plan` does not import resources. Importing resources into the state file is done with the `terraform import` command.
- The answer "Installs all providers and modules referenced by configuration" is incorrect because installing providers and modules is handled by `terraform init`, not `terraform plan`.

Resources
Terraform plan
Question 19 Correct
Which of the following statements about Terraform modules is not true?
Your answer is correct
Modules must be publicly accessible
Explanation
This statement is not true. Modules in Terraform do not need to be publicly accessible. Modules are self-contained units of Terraform configurations that can be stored locally or in version control systems without the need for public access.
You can call the same module multiple times
Explanation
This statement is true. In Terraform, you can call the same module multiple times within your configuration to create multiple instances of the resources defined in that module.
A module is a container for one or more resources
Explanation
This statement is true. A module in Terraform is indeed a container for one or more resources. It allows you to encapsulate related resources, variables, and outputs into a reusable and shareable unit.
Modules can call other modules
Explanation
This statement is true. In Terraform, modules can call other modules, allowing for modular and reusable configurations. This feature enables you to create complex infrastructure setups by composing smaller, reusable modules.
Overall explanation
Correct Answer: Modules must be publicly accessible

Explanation:

- **Modules must be publicly accessible** is not true. Terraform modules can be either public or private. You can create private modules and store them in a private repository (e.g., within your organization's version control system), or in private cloud storage such as Amazon S3. Public accessibility is not a requirement for modules.
- **You can call the same module multiple times** is true. Terraform allows you to reuse the same module in your configuration multiple times with different configurations (by passing different variables or using `count` or `for_each`).
- **A module is a container for one or more resources** is true. A module typically encapsulates resources, outputs, input variables, and any other Terraform code to provide reusable infrastructure components.
- **Modules can call other modules** is true. Terraform modules can reference other modules by calling them from within their configuration, which allows for modular and organized infrastructure management.

Resources
Modules

Question 20 Correct
How can a ticket-based system slow down infrastructure provisioning and limit the ability to scale? (Choose two.)

Your selection is correct
End-users have to request infrastructure changes
Explanation
End-users having to request infrastructure changes through a ticket-based system introduces a manual step in the provisioning process, which can slow down the overall speed of infrastructure deployment. This manual intervention can lead to delays as tickets are processed and approved before the actual provisioning takes place.

Ticket based systems generate a full audit trail of the request and fulfillment process
Explanation
Ticket-based systems generate a full audit trail of the request and fulfillment process, which is beneficial for tracking and compliance purposes. However, this detailed audit trail can also add complexity to the infrastructure provisioning process, potentially slowing it down and limiting scalability as each step must be meticulously documented.

Users can access a catalog of approved resources from drop down lists m a request form
Explanation
Users being able to access a catalog of approved resources from drop-down lists in a request form can streamline the request process and make it more user-friendly. While this feature can improve user experience, it does not directly impact the speed of infrastructure provisioning or scalability limitations.

Your selection is correct
The more resources your organization needs, the more tickets your infrastructure team has to process
Explanation
As the organization requires more resources, the number of tickets that the infrastructure team has to process also increases. This can lead to a bottleneck in the provisioning process, causing delays and limiting the ability to scale efficiently. The manual handling of each ticket can become overwhelming as the demand for resources grows.

Overall explanation
Correct Answer:

- **End-users have to request infrastructure changes**
- **The more resources your organization needs, the more tickets your infrastructure team has to process**

Explanation:

- **End-users have to request infrastructure changes**: In a ticket-based system, end-users are required to submit requests for infrastructure changes, which creates a manual step that can slow down the process. It introduces delays as requests need to be reviewed, approved, and fulfilled by the operations team.
- **The more resources your organization needs, the more tickets your infrastructure team has to process**: As the need for infrastructure scales, the number of tickets submitted by different teams or departments increases. This can overwhelm the infrastructure team, leading to longer processing times and a bottleneck in scaling, reducing the ability to quickly adapt to new requirements.
- **Ticket-based systems generate a full audit trail of the request and fulfillment process** is actually an advantage, as it helps maintain a record for compliance and accountability. However, it doesn't inherently slow down the process unless the audit system is inefficient, which is not implied by this option.
- **Users can access a catalog of approved resources from drop-down lists in a request form** is actually a feature that helps speed up the process, as it provides predefined resources for users to choose from. This reduces friction and simplifies the process, rather than slowing it down.

Question 21 Incorrect
How do you specify a module's version when publishing it to the public Terraform Module Registry?

Your answer is incorrect
Configure it in the module's Terraform code
Explanation

Configuring the module's version in the Terraform code is a common practice to specify the version of the module when publishing it to the public Terraform Module Registry. This allows users to easily identify and use the specific version of the module they need for their infrastructure.

Mention it on the module s configuration page on the Terraform Module Registry

Explanation

Mentioning the module's version on the module's configuration page on the Terraform Module Registry is not the standard way to specify a module's version. While additional information can be provided on the configuration page, the versioning is typically managed through the associated repository.

The Terraform Module Registry does not support versioning modules

Explanation

The Terraform Module Registry does support versioning modules, allowing users to specify and use different versions of a module based on their requirements. Versioning is an essential feature of the registry to ensure compatibility and consistency in infrastructure deployments.

Correct answer

Tag a release in the associated repo

Explanation

Tagging a release in the associated repository is the correct way to specify a module's version when publishing it to the public Terraform Module Registry. By tagging a release, users can easily identify and access the specific version of the module they need for their Terraform configurations.

Overall explanation

Correct Answer: Tag a release in the associated repo

Explanation:

- **Tag a release in the associated repo**: In the public Terraform Module Registry, the version of a module is determined by the release tags in the associated GitHub repository. When you tag a release in the GitHub repository (e.g., `v1.0.0`), the Terraform Module Registry automatically associates that version tag with the module. This allows users to reference specific versions of the module when they use it in their configurations.
- **Configure it in the module's Terraform code** is incorrect because versioning is not specified within the module's actual Terraform code itself. Versioning happens at the repository level (through tags), not within the code files.
- **Mention it on the module's configuration page on the Terraform Module Registry** is incorrect because versioning isn't directly configured on the Terraform Module Registry page. Instead, it is managed through release tags in the repository.
- **The Terraform Module Registry does not support versioning modules** is incorrect. The Terraform Module Registry does indeed support versioning, and it uses the release tags from the associated GitHub repository to track and manage module versions.

Resources

Publish modules

Question 22 Correct

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

Your answer is correct

terraform plan -refresh-only

Explanation

The `terraform plan -refresh-only` command is used to update the state file with changes that might have been made outside of Terraform. It refreshes the state without making any changes to the infrastructure, ensuring that the state file accurately reflects the current state of the resources.

terraform show -json

Explanation

The `terraform show -json` command is used to display the Terraform state or plan in a machine-readable JSON format. It does not update the state file with changes made outside of Terraform, so it is not the correct choice for this scenario.

terraform apply -lock-false

Explanation

The `terraform apply -lock-false` command is used to apply changes to the infrastructure without locking the state file. While it can be useful in certain situations, it does not specifically update the state file with changes made outside of Terraform, so it is not the correct choice for this scenario.

terraform plan -target-state

Explanation

The `terraform plan -target-state` command is used to generate a new execution plan that includes only the specified resource as the target state. It does not update the state file with changes made outside of Terraform, so it is not the correct choice for this scenario.

Overall explanation

Correct Answer: terraform plan -refresh-only

Explanation:

- **terraform plan -refresh-only**: This command forces Terraform to refresh the state file by querying the cloud provider for the current state of the resources. It updates the state file to reflect any changes that might have been made outside of Terraform, such as manual modifications through the cloud provider's console or APIs.
- **terraform show -json** is incorrect because this command is used to display the state file in JSON format. It does not cause any updates to the state file itself.
- **terraform apply -lock-false** is incorrect because the `-lock-false` flag disables state file locking during `terraform apply`. While `terraform apply` does modify the state file by applying changes, it does not specifically refresh the state based on external modifications unless explicitly instructed to do so.
- **terraform plan -target-state** is incorrect because this command doesn't exist in Terraform. The `-target` flag is used to apply changes to specific resources, not for refreshing or updating the state file based on external changes.

Resources

Terraform state refresh

Question 23Correct

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

terraform plan

Explanation

Running `terraform plan` is used to generate an execution plan, showing what actions Terraform will take to change the infrastructure. However, it is not necessary to run `terraform plan` before performing further Terraform operations in a working directory.

terraform workspace

Explanation

`terraform workspace` is used to manage Terraform workspaces, allowing you to switch between different states of your infrastructure. While workspaces are useful for managing multiple environments, it is not a command that must be run before performing further Terraform operations in a working directory.

Your answer is correct

terraform init

Explanation

Running `terraform init` is the correct command to run before performing further Terraform operations in a working directory. This command initializes a working directory containing Terraform configuration files, downloading any necessary plugins and modules for the configuration.

terraform import

Explanation

`terraform import` is used to import existing infrastructure into Terraform. While this command can be useful for incorporating existing resources into your Terraform configuration, it is not a command that must be run before performing further Terraform operations in a working directory.

Overall explanation

Correct Answer: terraform init

Explanation:

The `terraform init` command is always the first command you must run before performing any other Terraform operations in a working directory. It initializes the working directory by downloading necessary provider plugins, configuring the backend (if specified), and preparing the directory for further operations like `terraform plan` or `terraform apply`.

- **terraform plan** is incorrect because it generates an execution plan but requires the working directory to be initialized first.
- **terraform workspace** is incorrect because it is used to manage workspaces, but the directory must be initialized before you can use this command.
- **terraform import** is incorrect because it imports existing resources into Terraform's state file, but this also requires the directory to be initialized first.

Resources

terraform init

Question 24 Incorrect

Which command lets you experiment with Terraform expressions?

Correct answer

terraform console

Explanation

The 'terraform console' command allows you to experiment with Terraform expressions interactively. It provides a way to test and evaluate different Terraform expressions before incorporating them into your infrastructure code.

terraform validate

Explanation

The 'terraform validate' command is used to validate the configuration files for syntax errors and other issues. It does not provide an interactive environment to experiment with Terraform expressions like the 'terraform console' command does.

Your answer is incorrect

terraform env

Explanation

The 'terraform env' command is used to manage Terraform execution environments, such as switching between different workspaces or managing environment variables. It does not provide a way to experiment with Terraform expressions.

terraform test

Explanation

The 'terraform test' command is not a standard Terraform command. Terraform does not have a built-in testing command, so this choice is incorrect in the context of experimenting with Terraform expressions.

Overall explanation

Correct Answer: terraform console

Explanation:

The `terraform console` command allows you to experiment with and evaluate Terraform expressions interactively. You can use it to test expressions, functions, and resource outputs within the context of your Terraform configuration, making it a helpful tool for debugging and understanding your setup.

- **terraform validate** is incorrect because it checks the syntax and validity of your Terraform configuration but does not allow interactive experimentation.
- **terraform env** is incorrect as it is used to manage workspaces (previously referred to as environments), not for experimenting with expressions.
- **terraform test** is incorrect because Terraform does not have a `test` command.

Resources

terraform console

Question 25 Incorrect

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

Your answer is incorrect

provider

Explanation

The provider block is used to configure the provider that will be used to interact with the infrastructure. It specifies details such as the provider type, version, and authentication credentials, but it does not create infrastructure objects with settings specified within the block.

state

Explanation

The state block is used to configure the state management settings for Terraform, such as the backend type, location, and encryption settings. It is not used to create infrastructure objects with settings specified within the block.

data

Explanation

The data block is used to define data sources that Terraform can use to fetch information from external systems. It allows Terraform to query and retrieve information, but it does not create infrastructure objects with settings specified within the block.

Correct answer

resource

Explanation

The resource block is used to define and create infrastructure objects with settings specified within the block. It specifies details such as the resource type, name, and configuration settings, allowing Terraform to create and manage the desired infrastructure resources.

Overall explanation

Correct Answer: resource

Explanation:

A `resource` block in Terraform defines an infrastructure object, such as a virtual machine, storage bucket, or database, and specifies its settings within the block. This block tells Terraform what to create and how to configure it in the target infrastructure provider.

- **provider** is incorrect because it configures the plugins that allow Terraform to interact with cloud providers, but it does not create infrastructure objects.
- **state** is incorrect because it refers to the Terraform state file, which tracks resources managed by Terraform, but it is not a configuration block.
- **data** is incorrect because it is used to retrieve information about existing resources but does not create infrastructure objects.

Resources

Resource Behavior

Question 26Incorrect

When do changes invoked by terraform apply take effect?

After Terraform has updated the state file

Explanation

After Terraform has updated the state file, the changes are recorded and stored for future reference. However, the actual implementation of these changes does not occur until the resource provider has fulfilled the request, making this choice incorrect in terms of when changes take effect.

Correct answer

Once the resource provider has fulfilled the request

Explanation

Once the resource provider has fulfilled the request, the changes invoked by terraform apply take effect. This means that the actual provisioning, modification, or deletion of resources occurs at this stage, making this choice the correct option for when changes become active.

Your answer is incorrect

Immediately

Explanation

While changes may appear to take effect immediately after running terraform apply, the actual implementation of these changes is dependent on the resource provider fulfilling the request. Therefore, stating that changes take effect immediately is not entirely accurate in the context of Terraform operations.

None of the above are correct

Explanation

None of the above are correct as the correct choice is option B, which states that changes take effect once the resource provider has fulfilled the request. The other options do not accurately describe the timing of when changes become active in the Terraform workflow.

Overall explanation

Correct Answer: Once the resource provider has fulfilled the request

Explanation:

When you run `terraform apply`, Terraform submits requests to the resource provider (e.g., AWS, Azure, or Google Cloud) to create, update, or delete resources as specified in the configuration. The changes take effect only after the provider fulfills those requests, which can sometimes take a few seconds or minutes, depending on the resource type and provider.

- **After Terraform has updated the state file** is incorrect because updating the state file occurs after the resource provider has fulfilled the changes, not before.
- **Immediately** is incorrect because changes depend on the provider's processing time and do not happen instantaneously.
- **None of the above are correct** is incorrect because the changes take effect when the provider fulfills the request.

Question 27 Correct

What is the workflow for deploying new infrastructure with Terraform?

Your answer is correct

Write Terraform configuration, run terraform init to initialize the working directory or workspace, and run terraform apply

Explanation

This choice correctly outlines the standard workflow for deploying new infrastructure with Terraform. Writing Terraform configuration files, initializing the working directory or workspace with terraform init, and applying the changes with terraform apply is the recommended sequence of steps to create new infrastructure.

Write Terraform configuration, run terraform show to view proposed changes, and terraform apply to create new infrastructure

Explanation

This choice is incorrect because it suggests using terraform show to view proposed changes before applying them. While terraform show can be used to inspect the current state, it is not typically used as part of the workflow for deploying new infrastructure.

Write Terraform configuration, run terraform apply to create infrastructure, use terraform validate to confirm Terraform deployed resources correctly

Explanation

This choice is incorrect because it suggests running terraform apply before validating the Terraform deployment with terraform validate. The recommended order is to validate the configuration with terraform validate before applying changes with terraform apply.

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

Explanation

This choice is incorrect because it suggests using terraform plan to initialize the working directory or workspace, which is not the correct usage of the terraform plan command. Terraform plan is typically used to generate an execution plan before applying changes with terraform apply.

Overall explanation

Correct Answer: Write Terraform configuration, run terraform init to initialize the working directory or workspace, and run terraform apply

Explanation:

The correct workflow for deploying new infrastructure with Terraform involves:

1. **Write Terraform configuration**: Define the desired infrastructure as code in `.tf` files.
2. **Run `terraform init`**: This initializes the working directory, downloads the necessary provider plugins, and prepares the configuration for deployment.
3. **Run `terraform apply`**: This creates or updates infrastructure as defined in the configuration after confirming the proposed changes.

- **Write Terraform configuration, run terraform show to view proposed changes, and terraform apply to create new infrastructure** is incorrect because `terraform show` displays details of the current state or the results of a previous plan or apply operation, not proposed changes.
- **Write Terraform configuration, run terraform apply to create infrastructure, use terraform validate to confirm Terraform deployed resources correctly** is incorrect because `terraform validate` checks the configuration syntax but does not validate deployed resources.
- **Write Terraform configuration, run terraform plan to initialize the working directory or workspace, and terraform apply to create the infrastructure** is incorrect because `terraform plan` does not initialize the directory or workspace; that is done by `terraform init`.

Resources

The Core Terraform Workflow

Question 28 Correct

Which of these are features of Terraform Cloud? (Choose two.)

Your selection is correct

Remote state storage

Explanation

Remote state storage is a key feature of Terraform Cloud, allowing users to store their Terraform state files securely in the cloud. This enables collaboration among team members and ensures consistency in infrastructure management.

Your selection is correct

A web-based user interface (UI)

Explanation

Terraform Cloud provides a web-based user interface (UI) that allows users to manage their Terraform configurations, workspaces, variables, and state files in a centralized and user-friendly manner. This UI simplifies the process of infrastructure provisioning and management.

Automatic backups

Explanation

Automatic backups are not a feature of Terraform Cloud. While backups are important for data protection and disaster recovery, Terraform Cloud focuses on providing features related to infrastructure provisioning, collaboration, and state management.

Automated infrastructure deployment visualization

Explanation

Automated infrastructure deployment visualization is not a feature of Terraform Cloud. While visualization tools can be helpful for understanding complex infrastructure configurations, Terraform Cloud primarily focuses on managing Terraform configurations, state files, and workspaces in a centralized platform.

Overall explanation

Correct Answers: Remote state storage and A web-based user interface (UI)

Explanation:

Terraform Cloud provides several features to streamline infrastructure management:

1. **Remote state storage**: Terraform Cloud securely stores the state file in a central location, eliminating the need for local state files and enabling team collaboration.
2. **A web-based user interface (UI)**: Terraform Cloud includes a web UI where users can manage workspaces, view plans, apply changes, and monitor states.

- **Automatic backups** is incorrect because while Terraform Cloud ensures state file availability and reliability, it does not explicitly offer a backup feature; state files are managed and stored centrally.
- **Automated infrastructure deployment visualization** is incorrect because Terraform Cloud does not natively provide visual diagrams of infrastructure. This would require additional tools like Terraform Graph or third-party services.

Question 29 Incorrect

Which option can not keep secrets out of Terraform configuration files?

A shared credential file

Explanation

A shared credential file can be used to store sensitive information such as access keys or passwords outside of Terraform configuration files. This helps keep secrets secure and separate from the main configuration.

Correct answer

Mark the variable as sensitive

Explanation

Marking a variable as sensitive in Terraform ensures that its value is not displayed in the console output or stored in the state file. This helps prevent sensitive information from being exposed, making it a suitable option for keeping secrets out of configuration files.

Your answer is incorrect

Environment Variables

Explanation

Environment variables can be used to pass sensitive information to Terraform without storing them directly in configuration files. By setting environment variables, you can keep secrets separate from the main configuration and prevent them from being exposed.

A -var flag

Explanation

The -var flag in Terraform allows you to pass variable values directly from the command line. While this can be useful for providing dynamic input, it does not inherently keep secrets out of configuration files. It is important to be cautious when using this method with sensitive information.

Overall explanation

Correct Answer: Mark the variable as sensitive

Explanation:

Marking a variable as sensitive in Terraform only ensures that its value is not displayed in logs or CLI output. However, it **does not prevent the secret from being hardcoded in the Terraform configuration files** or from being visible in the state file. Sensitive variables are helpful for obfuscation during execution but do not address how secrets are stored in configuration files or the state file.

- **A shared credential file** is a secure method to store credentials externally from Terraform code and reference them when needed.
- **Environment Variables** can also be used to keep secrets out of configuration files by storing them securely in the system and referencing them during runtime.
- **A -var flag** allows you to pass secret values directly during runtime, avoiding the need to include them in configuration files.

Sensitive marking alone is insufficient to fully protect secrets.

Question 30Correct

Which Terraform command checks that your configuration syntax is correct?

terraform fmt

Explanation

The `terraform fmt` command is used to automatically format your Terraform configuration files to follow the standard style guidelines. It does not check the syntax of your configuration for correctness.

Your answer is correct

terraform validate

Explanation

The `terraform validate` command is used to check that your Terraform configuration files are syntactically valid and internally consistent. It ensures that the configuration can be successfully parsed and validated by Terraform.

terraform init

Explanation

The `terraform init` command is used to initialize a Terraform working directory and download any necessary plugins or modules. It does not specifically check the syntax of your configuration files for correctness.

terraform show

Explanation

The `terraform show` command is used to generate a human-readable summary of the current state of your infrastructure as managed by Terraform. It does not perform syntax validation on your configuration files.

Overall explanation

Correct Answer: terraform validate

Explanation:

The `terraform validate` command checks the syntax and structure of your Terraform configuration files to ensure they are correct and valid. It does not check for runtime errors or infrastructure changes but is a critical step in verifying that the configuration is syntactically valid before running further commands like `terraform plan` or `terraform apply`.

- **terraform fmt** is used to format Terraform configuration files according to standard conventions but does not validate syntax.
- **terraform init** initializes a Terraform working directory, downloading provider plugins and setting up the backend but does not validate configuration syntax.
- **terraform show** displays information about the current state or plan but is not used for validation.

Question 31Incorrect
terraform validate uses provider APIs to verify your infrastructure settings.

Your answer is incorrect
True
Explanation
This statement is incorrect. The `terraform validate` command does not use provider APIs to verify infrastructure settings. Instead, it checks the configuration files for syntax errors, attribute references, and other potential issues without actually interacting with the provider APIs.

Correct answer
False
Explanation
This statement is correct. The `terraform validate` command does not rely on provider APIs to verify infrastructure settings. It focuses on checking the Terraform configuration files for any errors or issues that may prevent successful deployment of the infrastructure.

Overall explanation
Correct Answer: False

Explanation:

The `terraform validate` command only checks the syntax and structure of the Terraform configuration files. It does **not** interact with provider APIs or verify infrastructure settings. Its purpose is to ensure that the configuration files are syntactically correct and follow Terraform's rules for defining resources, variables, and outputs.

To validate infrastructure settings with the provider APIs, you would need to use commands like `terraform plan`, which compares the desired state in your configuration with the actual infrastructure state by interacting with the provider APIs.

Question 32Correct
You add a new provider to your configuration and immediately run terraform apply in the CLI using the local backend. Why does the apply fail?

Terraform needs you to format your code according to best practices first
Explanation
Formatting your code according to best practices is not a prerequisite for running terraform apply. While it is recommended to follow best practices for readability and maintainability, it is not a reason for the apply to fail immediately after adding a new provider.

Terraform requires you to manually run terraform plan first
Explanation
While running terraform plan before apply is a best practice to preview the changes that will be made, it is not a requirement for the apply to succeed. The apply command can be run directly after adding a new provider without manually running plan first.

The Terraform CLI needs you to log into Terraform Cloud first
Explanation

Logging into Terraform Cloud is not necessary when using the local backend to run terraform apply. The local backend allows you to apply changes locally without the need for an external service like Terraform Cloud.

Your answer is correct

Terraform needs to install the necessary plugins first

Explanation

When a new provider is added to the configuration, Terraform needs to download and install the necessary plugins to communicate with the provider's API. If the plugins are not installed, the apply command will fail because Terraform cannot execute the necessary actions without the required plugins.

Overall explanation

Correct Answer: Terraform needs to install the necessary plugins first

Explanation:

When you add a new provider to your Terraform configuration, Terraform needs to download the corresponding provider plugin before executing any operations. This happens during the `terraform init` command, which initializes the working directory and installs all required plugins and modules.

If you skip running `terraform init` and directly run `terraform apply`, the operation will fail because the required provider plugin is not yet installed.

- **"Terraform needs you to format your code according to best practices first"** is incorrect because code formatting is optional and handled by `terraform fmt`, not a prerequisite for running `terraform apply`.
- **"Terraform requires you to manually run terraform plan first"** is incorrect because running `terraform plan` is not mandatory; it is recommended but not required.
- **"The Terraform CLI needs you to log into Terraform Cloud first"** is incorrect when using the local backend, as it does not involve Terraform Cloud.

Always run `terraform init` when adding new providers to avoid this issue.

Question 33Incorrect

Which of these statements about Terraform Cloud workspaces is false?

They can securely store cloud credentials

Explanation

Terraform Cloud workspaces can securely store cloud credentials, allowing users to manage and access their cloud resources securely without exposing sensitive information. This feature enhances the overall security of infrastructure management processes.

Your answer is incorrect

They have role-based access controls

Explanation

Terraform Cloud workspaces have role-based access controls, which enable administrators to define and manage user permissions within the workspace. This ensures that only authorized users can perform specific actions, enhancing security and governance.

Correct answer

You must use the CLI to switch between workspaces

Explanation

The statement that you must use the CLI to switch between workspaces is false. In Terraform Cloud, users can easily switch between workspaces within the web interface without the need to use the CLI. This functionality simplifies workspace management and improves user experience.

Plans and applies can be triggered via version control system integrations

Explanation

Plans and applies can be triggered via version control system integrations in Terraform Cloud workspaces. This feature allows users to automate the execution of Terraform workflows based on changes in the version control system, streamlining the infrastructure deployment process.

Overall explanation

Correct Answer: You must use the CLI to switch between workspaces

Explanation:

- "**They can securely store cloud credentials**" is true. Terraform Cloud workspaces allow you to securely store cloud credentials such as API tokens, which are needed for interacting with cloud providers.
- "**They have role-based access controls**" is true. Terraform Cloud supports role-based access control (RBAC), allowing you to set permissions for users in the workspace.
- "**Plans and applies can be triggered via version control system integrations**" is true. Terraform Cloud integrates with version control systems (VCS) like GitHub, GitLab, and Bitbucket, allowing for automatic triggers of Terraform plans and applies when changes are pushed to the repository.
- "**You must use the CLI to switch between workspaces**" is false. In Terraform Cloud, you can switch between workspaces using the web interface, so using the CLI is not required. You can manage workspaces through the Terraform Cloud UI, making it convenient to interact with different workspaces without needing the CLI.

Therefore, the false statement is that you must use the CLI to switch between workspaces.

Question 34Correct

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

Your answer is correct

The ability to restrict modules to members of Terraform Cloud or Enterprise organizations

Explanation

The Terraform Cloud private registry allows users to restrict access to modules to only members of Terraform Cloud or Enterprise organizations. This provides a higher level of security and control over who can view and use the modules, which is not possible with the public Terraform Module Registry.

The ability to share modules publicly with any user of Terraform

Explanation

The ability to share modules publicly with any user of Terraform is not a feature of the Terraform Cloud private registry. This functionality is typically associated with the public Terraform Module Registry, where modules are available to all Terraform users, regardless of their organization or membership status.

The ability to tag modules by version or release

Explanation

The ability to tag modules by version or release is a common feature of both the public Terraform Module Registry and the Terraform Cloud private registry. This feature allows users to track and manage different versions of modules for better version control and module management.

The ability to share modules with public Terraform users and members of Terraform Cloud Organizations

Explanation

The Terraform Cloud private registry does not provide the ability to share modules with public Terraform users. It is specifically designed to restrict module access to members of Terraform Cloud or Enterprise organizations, ensuring that modules are only available to authorized users within the organization. Sharing with public Terraform users is a feature of the public Terraform Module Registry.

Overall explanation

Correct Answer: The ability to restrict modules to members of Terraform Cloud or Enterprise organizations

Explanation:

- "**The ability to restrict modules to members of Terraform Cloud or Enterprise organizations**" is the correct value provided by the Terraform Cloud private registry. It allows organizations to control access to modules by restricting them to members of the organization or specific teams within Terraform Cloud or Terraform Enterprise. This is especially useful for organizations that want to maintain private, internal modules and avoid exposing them to the public.
- "**The ability to share modules publicly with any user of Terraform**" is not specific to the private registry but applies to the public Terraform Module Registry. The public registry allows anyone to access and use modules.
- "**The ability to tag modules by version or release**" is a feature supported in both public and private registries. You can tag modules by version in both registries to manage and organize releases.
- "**The ability to share modules with public Terraform users and members of Terraform Cloud Organizations**" is not a unique feature of the private registry. The public registry also allows sharing modules with all users, and the private registry is specifically for controlling access within organizations.

Hence, the primary advantage of the Terraform Cloud private registry is the ability to restrict module access to members of your Terraform Cloud or Enterprise organizations.

Question 35 Incorrect

Terraform providers are part of the Terraform core binary.

Your answer is incorrect

True

Explanation

This statement is incorrect. Terraform providers are not part of the Terraform core binary. Providers are separate plugins that extend Terraform's capabilities to interact with specific APIs or services.

Correct answer

False

Explanation

This statement is correct. Terraform providers are not included in the Terraform core binary. Providers are external plugins that need to be downloaded and installed separately to interact with different infrastructure platforms or services.

Overall explanation

Correct Answer: False

Explanation:

Terraform providers are not part of the Terraform core binary. Instead, they are separate plugins that Terraform dynamically downloads and uses during execution. These providers interact with the APIs of cloud services or other infrastructure platforms to manage resources.

- **Terraform Core** is responsible for managing the Terraform workflow, including the execution plan and applying configurations, but the actual interactions with cloud services or systems are handled by the providers, which are separate from the core.
- Providers are typically installed and managed by Terraform automatically when you run commands like `terraform init`, which downloads the necessary provider plugins.

Question 36 Correct

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

Reusability of code

Explanation

Reusability of code is a significant benefit of adopting infrastructure as code. By defining infrastructure in code, it can be reused across different environments, projects, and teams, leading to increased efficiency and consistency in deployments.

Automation

Explanation

Automation is a key advantage of infrastructure as code. By automating the provisioning, configuration, and management of infrastructure through code, manual tasks are reduced, errors are minimized, and deployments become more reliable and repeatable.

Your answer is correct

Graphical User Interface

Explanation

Graphical User Interface (GUI) is not typically considered a direct benefit of adopting infrastructure as code. Infrastructure as code focuses on defining and managing infrastructure through code files, scripts, or configuration files, rather than relying on GUI-based tools for provisioning and configuration.

Versioning

Explanation

Versioning is an essential benefit of infrastructure as code. By storing infrastructure configurations in version control systems, changes can be tracked, rollback to previous versions is possible, and collaboration among team members is facilitated.

Overall explanation

Correct Answer: Graphical User Interface

Explanation:

The primary benefits of adopting infrastructure as code (IaC) include:

- **Reusability of code**: Infrastructure code can be reused across different environments, reducing duplication and ensuring consistency.
- **Automation**: IaC enables the automatic creation, modification, and deletion of infrastructure, making it easier to manage at scale.
- **Versioning**: Infrastructure configurations can be versioned, similar to application code, allowing teams to track changes and revert to previous versions if necessary.

On the other hand, **Graphical User Interface (GUI)** is not a benefit of IaC. In fact, IaC typically replaces GUI-based interactions with code-driven management. While GUIs might be available for managing infrastructure, IaC focuses on defining infrastructure through code rather than through graphical interfaces.

Question 37Correct

Where does the Terraform local backend store its state?

Your answer is correct

In the terraform.tfstate file

Explanation

The Terraform local backend stores its state in the terraform.tfstate file. This file contains the current state of the infrastructure managed by Terraform and is used to determine what changes need to be applied during subsequent runs.

In the .terraform directory

Explanation

The .terraform directory is used by Terraform to store internal files and plugins, but it does not store the state of the infrastructure. The state file is typically stored separately from this directory.

In the terraform.tfstate directory

Explanation

There is no specific terraform.tfstate directory where the Terraform local backend stores its state. The state is typically stored in the terraform.tfstate file itself, rather than in a separate directory.

In the .terraform.lock.hcl file

Explanation

The .terraform.lock.hcl file is used to lock the versions of the providers and modules used in a Terraform configuration. It does not store the state of the infrastructure, which is instead stored in the terraform.tfstate file.

Overall explanation

The Terraform local backend stores its state in the `terraform.tfstate` file.

Explanation: The local backend, which is the default backend for Terraform, stores the state file in the `terraform.tfstate` file within the working directory. This file contains the latest state of the infrastructure managed by Terraform, allowing it to track the resources, dependencies, and attributes of the managed infrastructure.

- `terraform.tfstate` is where the state is kept by default in the local backend.
- `.terraform` directory is used by Terraform for storing modules and provider plugins, not for state files.
- `.terraform.lock.hcl` is used for locking provider versions, not for state storage.
- `terraform.tfstate` directory does not exist by default unless explicitly created for organizing state files in custom configurations.

Question 38Incorrect

Which of these is true about Terraform's plugin-based architecture?

Terraform can only source providers from the internet

Explanation

This statement is incorrect. While Terraform can automatically download providers from the internet if they are not already installed, it is also possible to manually install providers from local files or custom sources.

Correct answer

You can create a provider for your API if none exists

Explanation

This statement is correct. Terraform's plugin-based architecture allows users to create custom providers for their own APIs if a pre-existing provider does not already exist. This flexibility enables users to extend Terraform's capabilities to manage resources specific to their infrastructure.

Every provider in a configuration has its own state file for its resources

Explanation

This statement is incorrect. In Terraform, all resources managed by providers in a configuration are stored in a single state file. Each resource is identified by a unique resource address that includes the provider name, type, and resource name.

Your answer is incorrect

All providers are part of the Terraform core binary

Explanation

This statement is incorrect. While Terraform's core binary includes essential functionality, providers are separate plugins that extend Terraform's capabilities to manage resources from various cloud providers, services, and APIs. Providers are not part of the Terraform core binary.

Overall explanation

Correct Answer: You can create a provider for your API if none exists

Explanation: Terraform uses a plugin-based architecture where providers are separate executable binaries that enable Terraform to interact with different APIs and services. If a provider for a specific API or service does not exist, you can create your own custom provider to manage resources for that service.

- Terraform can source providers from the internet through the Terraform Registry, but it can also use providers that are locally available or custom-built.
- Providers in a configuration share a common state file rather than each having their own individual state file. The state is maintained in a single file or backend.
- Not all providers are part of the Terraform core binary. Terraform core provides a basic set of functionalities, but most providers are separate plugins that extend Terraform's capabilities.

Question 39Correct

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?

Auditing cloud storage buckets with a vulnerability scanning tool

Explanation

Auditing cloud storage buckets with a vulnerability scanning tool is a reactive approach to security control and does not automatically enforce the requirement for new AWS S3 buckets to be private and encrypted at rest. It does not proactively ensure that the security control is always in place.

Your answer is correct

With a Sentinel policy, which runs before every apply

Explanation

Using a Sentinel policy that runs before every apply in Terraform Cloud allows for automatic and proactive enforcement of the security control requirement. By defining a policy that checks for private and encrypted at rest settings for new S3 buckets, Terraform Cloud can ensure that these requirements are always met before any changes are applied.

With an S3 module with proper settings for buckets

Explanation

Creating an S3 module with proper settings for buckets is a good practice for standardizing configurations, but it does not automatically enforce the security control requirement. Without a mechanism to ensure that the module is always used and configured correctly, there is no guarantee that new S3 buckets will always be private and encrypted at rest.

By adding variables to each Terraform Cloud workspace to ensure these settings are always enabled

Explanation

Adding variables to each Terraform Cloud workspace to ensure these settings are always enabled is a manual approach to enforcing the security control requirement. While variables can help streamline the configuration process, they do not provide an automated and proactive way to ensure that new S3 buckets are always private and encrypted at rest.

Overall explanation

Correct Answer: With a Sentinel policy, which runs before every apply

Explanation: Terraform Cloud allows the enforcement of security and compliance controls using Sentinel policies. Sentinel is a policy-as-code framework that integrates with Terraform to define, enforce, and monitor policies for infrastructure. By using a Sentinel policy, you can ensure that S3 buckets are created with the required settings for privacy and encryption at rest before any `terraform apply` is executed, ensuring compliance with the organization's security requirements.

- Auditing cloud storage buckets with a vulnerability scanning tool is a post-deployment activity and does not prevent issues before they occur.
- Using a module with proper settings for S3 buckets can help ensure the correct configuration, but it is not a proactive enforcement mechanism like Sentinel.
- Adding variables to workspaces ensures the variables are set but does not enforce policy-based checks or ensure compliance with specific controls like encryption at rest and bucket privacy.

Question 40Incorrect

If you don't use the local backend, where does Terraform save resource state?

Correct answer

In the remote backend or Terraform Cloud

Explanation

When the local backend is not used, Terraform saves the resource state in the remote backend or Terraform Cloud. This allows for centralized management of state files, ensuring consistency and collaboration among team members working on the same infrastructure.

Your answer is incorrect

On the disk

Explanation

Terraform does not save resource state on the disk when the local backend is not used. Storing state on the disk can lead to inconsistencies and conflicts when multiple team members are working on the same infrastructure, making it essential to use a remote backend or Terraform Cloud for state management.

In memory

Explanation

Terraform does not store resource state in memory when the local backend is not used. Storing state in memory would result in the loss of state information when the Terraform process is terminated, making it necessary to use a persistent storage solution like a remote backend or Terraform Cloud.

In an environment variable

Explanation

Terraform does not save resource state in an environment variable when the local backend is not used. Environment variables are typically used for configuration settings and not for storing and managing the state of infrastructure resources. It is important to use a dedicated state management solution like a remote backend or Terraform Cloud for effective state handling.

Overall explanation

Correct Answer: In the remote backend or Terraform Cloud

Explanation: When Terraform is configured to use a remote backend (such as Terraform Cloud or a custom remote backend), it stores the state of resources in that remote location. The state file is stored remotely instead of locally, allowing for centralized management, sharing, and team collaboration. Terraform Cloud and other remote backends also provide features like state locking and versioning, which are critical for collaboration in a multi-team environment.

- Storing state on disk happens only when using the local backend, not when using a remote backend.
- In-memory state storage occurs temporarily during a Terraform run but is not persistent across sessions.
- Environment variables do not store Terraform state; they are typically used to configure credentials and other settings for the execution.

Question 41Correct

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?

Your answer is correct

output "instance_ip_addr" {

value = aws_instance.main.private_ip

}

Explanation

This choice correctly defines an output named "instance_ip_addr" that references the private IP address of the AWS instance named "main" in the child module. The syntax used to access the private IP address is valid for Terraform and will allow the root configuration to access this value.

output "aws_instance.instance_ip_addr" {

value = ${main.private_ip}

}

Explanation

This choice incorrectly defines the output by using an incorrect syntax for referencing the private IP address of the AWS instance. The "${}" syntax is not valid for accessing attributes of resources in Terraform. Additionally, the naming convention for the output does not match the correct way to reference the resource.

output "instance_ip_addr" {

return aws_instance.main.private_ip

}

Explanation

This choice incorrectly defines the output by using the "return" keyword, which is not valid for defining outputs in Terraform. Outputs are defined using the "output" keyword. Additionally, the syntax used to access the private IP address is incorrect and will not provide the desired result in the root configuration.

output "aws_instance.instance_ip_addr" {

return aws_instance.main.private_ip

}

Explanation

This choice incorrectly defines the output by using the "return" keyword, which is not valid for defining outputs in Terraform. Outputs are defined using the "output" keyword. Additionally, the naming convention for the output does not match the correct way to reference the resource, and the syntax used to access the private IP address is incorrect.

Overall explanation

Correct Answer: output "instance_ip_addr" { value = aws_instance.main.private_ip }

Explanation:

- This is the correct way to define an output value in Terraform. The `output` block allows you to export values from a module to be used in the root configuration or other modules. The correct syntax is to use `value = aws_instance.main.private_ip` to reference the IP address of the `aws_instance.main` resource.
- The other options are incorrect because:
  - The syntax `return` is not used in Terraform's `output` block.
  - The syntax `${main.private_ip}` and `aws_instance.instance_ip_addr` are not valid ways to reference the output value.

Resources

[Output Values]()

Question 42Correct

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

Before the plan phase

Explanation

Sentinel does not enforce policy logic before the plan phase in Terraform Cloud runs. It is not involved in the initial planning process of the infrastructure deployment.

During the plan phase

Explanation

Sentinel does not enforce policy logic during the plan phase in Terraform Cloud runs. The plan phase is focused on generating an execution plan for the infrastructure changes to be applied.

Your answer is correct

Before the apply phase

Explanation

Sentinel enforces policy logic before the apply phase in Terraform Cloud runs. This ensures that policy checks are performed before any changes are actually applied to the infrastructure, allowing for the prevention of non-compliant configurations.

After the apply phase

Explanation

Sentinel does not enforce policy logic after the apply phase in Terraform Cloud runs. Once the changes have been applied to the infrastructure, Sentinel's role in enforcing policies is completed.

Overall explanation

Correct Answer: Before the apply phase

Explanation:

- Sentinel enforces policy logic **before the apply phase**. It ensures that the proposed changes in the Terraform plan meet the required policies before any changes are made to the infrastructure. This allows you to validate configurations against organizational standards, security requirements, and compliance regulations before Terraform executes any actions.
- Sentinel does not enforce policies during the plan phase; it analyzes the plan after it has been generated and before the apply phase starts.

Question 43Incorrect

What is terraform refresh-only intended to detect?

Empty state files

Explanation

Terraform refresh-only is not intended to detect empty state files. It is used to refresh the state of the infrastructure without making any changes to the resources.

Corrupt state files

Explanation

Terraform refresh-only is not designed to detect corrupt state files. It is used to update the state of the infrastructure without applying any changes.

Your answer is incorrect

Terraform configuration code changes

Explanation

Terraform refresh-only does not detect Terraform configuration code changes. It is specifically used to refresh the state of the infrastructure without modifying any resources.

Correct answer

State file drift

Explanation

Terraform refresh-only is intended to detect state file drift, which occurs when the actual state of the infrastructure resources differs from the state recorded in the Terraform state file. This command helps to identify any variances between the actual state and the expected state defined in the configuration.

Overall explanation

Correct Answer: State file drift

Explanation:

- The **terraform refresh-only** command is specifically designed to detect **state file drift**. Drift occurs when the actual state of the resources differs from the state recorded in the Terraform state file, often due to changes made outside of Terraform (such as manual changes in the cloud provider's console).
- This command updates the state file to match the current state of the infrastructure without making any changes to the actual infrastructure itself.

Resources

terraform refresh

Question 44 Correct

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

Your answer is correct

True

Explanation

Running `terraform fmt` is indeed true as it is a command used to automatically format all Terraform configurations within the current working directory to adhere to Terraform-style conventions. This helps maintain consistency in the codebase and makes it easier for team members to read and understand the configurations.

False

Explanation

The statement is false. Running `terraform fmt` is a valid command in Terraform that helps in formatting Terraform configurations to follow the Terraform-style conventions. It is a recommended practice to run this command to ensure consistency and readability in the codebase.

Overall explanation

Correct Answer: True

Explanation:

- Running **terraform fmt** automatically formats all Terraform configuration files in the current working directory according to Terraform's standard style conventions. This helps ensure consistency and readability in the code, making it easier for teams to collaborate.

Resources

terraform fmt

Question 45 Correct

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

You want to force Terraform to destroy a resource on the next apply

Explanation

Using the -replace flag in Terraform apply does not specifically force Terraform to destroy a resource on the next apply. It is used for a different purpose related to resource management.

You want Terraform to ignore a resource on the next apply

Explanation

The -replace flag in Terraform apply is not used to ignore a resource on the next apply. It serves a different function in the Terraform workflow.

Your answer is correct

You want to force Terraform to destroy and recreate a resource on the next apply

Explanation

The correct use of the -replace flag in Terraform apply is to force Terraform to destroy and recreate a resource on the next apply. This can be useful when you need to make significant changes to a resource that cannot be updated in place.

You want Terraform to destroy all the infrastructure in your workspace

Explanation

The -replace flag in Terraform apply does not automatically destroy all the infrastructure in your workspace. It is specifically used to manage the destruction and recreation of individual resources during the apply process.

Overall explanation

Correct Answer: You want to force Terraform to destroy and recreate a resource on the next apply

Explanation:

- The **-replace** flag forces the destruction and recreation of a specific resource during the next `terraform apply`, even if Terraform doesn't detect any changes to it. This can be necessary if a resource needs to be recreated due to configuration changes or issues with the existing instance.
- **You want to force Terraform to destroy a resource on the next apply**: This is incorrect because the **-replace** flag doesn't just destroy the resource; it also recreates it. The **-destroy** flag would be used if you just want to destroy a resource, without recreating it.
- **You want Terraform to ignore a resource on the next apply**: This is incorrect because the **-replace** flag explicitly tells Terraform to handle the resource by destroying and recreating it, not to ignore it. To ignore a resource, you would use the `lifecycle` block with the `ignore_changes` argument.
- **You want Terraform to destroy all the infrastructure in your workspace**: This is incorrect because the **-replace** flag only applies to specific resources, not the entire infrastructure. To destroy all infrastructure, you would use `terraform destroy`, which is a command that destroys all the resources in the workspace.

Resources
Planning Options

Question 46Incorrect
When does Terraform create the **.terraform.lock.hcl** file?

After your first terraform plan
Explanation
Terraform does not create the .terraform.lock.hcl file after the first terraform plan. The lock file is used to record the versions of the provider plugins and modules required for a specific configuration, and it is generated at a different stage of the Terraform workflow.

After your first terraform apply
Explanation
The .terraform.lock.hcl file is not created after the first terraform apply. Applying changes to the infrastructure does not trigger the creation of the lock file, as its purpose is to manage dependencies and versions of providers and modules.

Correct answer
After your first terraform init
Explanation
The .terraform.lock.hcl file is created after the first terraform init command. The init command initializes a working directory containing Terraform configuration files, downloads necessary plugins, and generates the lock file to ensure consistent provider and module versions across environments.

Your answer is incorrect
Whenever you enable state locking
Explanation
Enabling state locking does not directly trigger the creation of the .terraform.lock.hcl file. State locking is a separate mechanism used to prevent concurrent modifications to the Terraform state file, while the lock file manages dependencies and versions of providers and modules.

Overall explanation
Correct Answer: After your first terraform init

Explanation:

- The **.terraform.lock.hcl** file is created after running the `terraform init` command. This file locks the provider versions in your Terraform project, ensuring that the same provider versions are used consistently across different runs and by other collaborators. It is a part of the initialization process, which retrieves and installs the necessary provider plugins.
- **After your first terraform plan**: This is incorrect. The `.terraform.lock.hcl` file is not created during the `terraform plan` phase but during the initialization process (i.e., `terraform init`).
- **After your first terraform apply**: This is also incorrect. The `.terraform.lock.hcl` file is not created during the `terraform apply` phase but rather during the initialization phase when Terraform checks dependencies and providers.
- **Whenever you enable state locking**: This is incorrect. State locking is related to the state file and does not trigger the creation of the `.terraform.lock.hcl` file, which is specifically for provider version locking.

Resources
Dependency Lock File

Question 47 Correct

You have been working in a Cloud provider account that is shared with other team members. You previously used Terraform to create a load balancer that is listening on port 80. After some application changes, you updated the Terraform code to change the port to 443.

You run terraform plan and see that the execution plan shows the port changing from 80 to 443 like you intended, and step away to grab some coffee.

In the meantime, another team member manually changes the load balancer port to 443 through the Cloud provider console before you get back to your desk.

What will happen when you terraform apply upon returning to your desk?

Terraform will fail with an error because the state file is no longer accurate.

Explanation

Terraform uses a state file to track the current state of the infrastructure. If the state file does not match the actual state of the resources in the Cloud provider account, Terraform will detect the inconsistency and fail with an error to prevent unintended changes.

Terraform will change the load balancer port to 80, and then change it back to 443.

Explanation

Terraform follows the desired state configuration, which means it will attempt to change the load balancer port to 80 as per the Terraform code. However, since the port is already set to 443 manually in the Cloud provider console, Terraform will then revert the change back to 443 to match the desired state.

Your answer is correct

Terraform will not make any changes to the Load Balancer and will update the state file to reflect any changes made.

Explanation

When Terraform detects a discrepancy between the desired state in the Terraform code and the actual state in the Cloud provider account, it will not make any changes to the resources. Instead, Terraform will update the state file to reflect the changes made outside of Terraform, ensuring that the state file remains accurate.

Terraform will recreate the load balancer.

Explanation

If Terraform detects that the actual state of the resources does not match the desired state in the Terraform code, it may attempt to recreate the resources to align with the desired configuration. However, in this scenario, since the load balancer port was manually changed to 443, Terraform will not recreate the load balancer.

Overall explanation

Correct Answer: Terraform will not make any changes to the Load Balancer and will update the state file to reflect any changes made.

Explanation:

- When you run `terraform apply`, Terraform checks the current state of resources by comparing the state file with the configuration. In this case, the load balancer's port has been manually updated to 443 by another team member. Terraform detects that the actual state matches the intended state (because the port is now 443, which matches your updated configuration), so no changes are needed.
- As a result, **Terraform will not make any changes** and will update the state file to reflect the change made manually by the team member, ensuring the state file is now accurate.
- **Terraform will fail with an error because the state file is no longer accurate**: This is incorrect. Terraform uses a mechanism like `terraform refresh` to detect the real state of resources and reconcile it with the state file. Terraform does not throw an error in this scenario as it can reconcile changes.
- **Terraform will change the load balancer port to 80, and then change it back to 443**: This is incorrect because Terraform sees that the port is already 443 (matching your intended configuration), so it does not try to change it back to 80.
- **Terraform will recreate the load balancer**: This is incorrect. Since Terraform detects that the load balancer's current state (port 443) matches the desired state, it will not recreate the resource.

Question 48 Incorrect

Which of the following is not an action performed by terraform init?

Correct answer

Create template configuration files

Explanation

Terraform init does not create template configuration files. It initializes the working directory containing Terraform configuration files, but it does not generate any new configuration files.

Initialize a configured backend

Explanation

Initializing a configured backend is indeed an action performed by terraform init. This step sets up the backend configuration specified in the Terraform configuration files.

Your answer is incorrect

Retrieve the source code for all referenced modules

Explanation

Retrieving the source code for all referenced modules is an action performed by terraform init. It ensures that all necessary module dependencies are downloaded and available for use during the Terraform execution.

Load required provider plugins

Explanation

Loading required provider plugins is another action performed by terraform init. It ensures that all necessary provider plugins are downloaded and available for use in the Terraform configuration.

Overall explanation

The action that is not performed by terraform init is Create template configuration files.

Here's why:

- **Initialize a configured backend**: `terraform init` sets up the backend configuration for storing state.
- **Retrieve the source code for all referenced modules**: This is part of what terraform init does when it installs modules defined in the configuration.
- **Load required provider plugins**: `terraform init` downloads the necessary plugins for the providers used in the configuration.

However, **Create template configuration files** is not something `terraform init` does. Template configuration files are typically manually created by users or generated outside of Terraform operations.

Resources

[terraform init]()

Question 49Correct

Before you can use a new backend or HCP Terraform/Terraform Cloud integration, you must first execute `terraform init`.

Your answer is correct

True

Explanation

True. Before you can start using a new backend or integrate with HCP Terraform/Terraform Cloud, you must run the terraform init command. This command initializes the working directory and downloads any necessary plugins and modules to ensure that Terraform can interact with the selected backend or integration properly.

False

Explanation

False. This statement is incorrect. In order to use a new backend or integrate with HCP Terraform/Terraform Cloud, you must first execute terraform init. This command is essential to set up the necessary environment for Terraform to work with the selected backend or integration effectively.

Overall explanation

The correct answer is **True**.

Explanation:

When you introduce a new backend, such as Terraform Cloud or HCP Terraform, or make changes to an existing one, you must first run `terraform init`. This command initializes your working directory by setting up the backend configuration, downloading the required provider plugins, and preparing your workspace for Terraform operations. Without running `terraform init`, Terraform won't be able to interact with the new backend or cloud integration, and you won't be able to manage your infrastructure as expected. Think of it like setting up all the necessary tools and configurations before you start working on a project.

Question 50Correct

```
    resource “google _compute_instance” “main” {
         name = “test”
    }
```

A resource block is shown in the Exhibit space of this page. What is the Terraform resource name of the resource block?

test

Explanation

The name "test" in the resource block represents the specific name assigned to the Google Compute Engine instance being created, not the Terraform resource name itself.

Your answer is correct

main

Explanation

The Terraform resource name in this case is "main," as indicated in the resource block declaration. This name is used to uniquely identify the resource within the Terraform configuration.

google

Explanation

The word "google" in the resource block is part of the resource type, specifying that the resource being created is a Google Cloud Platform resource. It is not the Terraform resource name.

compute_instance

Explanation

"compute_instance" in the resource block represents the specific resource type being created, which is a Google Compute Engine instance. It is not the Terraform resource name, but rather the resource type within the Google Cloud Platform provider.

Overall explanation

The correct answer is main.

Explanation:

In Terraform, the resource name consists of two parts: the resource type (in this case, `google_compute_instance`) and the resource name (here, `main`). The resource name (`main`) is a user-defined identifier used to uniquely reference this specific resource within the Terraform configuration. While `test` is the name of the instance being created in the cloud provider, it is not the Terraform resource name. Similarly, `google_compute_instance` is the resource type, not the resource name.

Therefore, the Terraform resource name for this block is `main`.

Question 51Correct

```
    module “consul” {
         source = “hashicorp/consul/aws”
    }
```

When you use a module block to reference a module from the Terraform Registry such as the one in the example, how do you specify version 1.0.0 of the module?

You cannot. Modules stored on the public Terraform Registry do not support versioning.

Explanation

This statement is incorrect. Modules stored on the public Terraform Registry do support versioning, and you can specify a specific version when referencing them in your Terraform configuration.

Your answer is correct

Add a version = “1.0.0” attribute to the module block.

Explanation

To specify version 1.0.0 of the module when using a module block to reference a module from the Terraform Registry, you need to add a version = "1.0.0" attribute to the module block. This ensures that your configuration uses the desired version of the module.

Nothing. Modules stored on the public Terraform module Registry always default to version 1.0.0.

Explanation
This statement is incorrect. Modules stored on the public Terraform module Registry do not default to version 1.0.0. You need to explicitly specify the version you want to use in your Terraform configuration.
Append ?ref=v1.0.0 argument to the source path.
Explanation
Appending ?ref=v1.0.0 to the source path is not the correct way to specify version 1.0.0 of a module from the Terraform Registry. Instead, you should add a version = "1.0.0" attribute to the module block in your Terraform configuration to ensure the correct version is used.
Overall explanation
The correct answer is Add a version = "1.0.0" attribute to the module block.

Explanation:
When referencing a module from the public Terraform Registry, you can specify a particular version of the module by adding the `version` attribute to the module block. For example:
```
module "consul" {
 source  = "hashicorp/consul/aws"
 version = "1.0.0"
}
```
This approach allows you to lock the module to a specific version, ensuring consistency and preventing unexpected changes from newer versions.

The other options are incorrect:

- Modules on the public Terraform Registry **do support versioning**, making the statement in "You cannot" false.
- Modules **do not default to version 1.0.0** unless explicitly specified, so "Nothing" is incorrect.
- Adding `?ref=v1.0.0` is not a valid syntax for specifying a version in Terraform. Versions are handled through the `version` attribute in the module block.

Resources
Module version constraints
Question 52Incorrect
Terraform encrypts sensitive values stored in your state file.
Your answer is incorrect
True
Explanation
Terraform does not automatically encrypt sensitive values stored in the state file. It is the responsibility of the user to implement encryption mechanisms or use external tools to secure sensitive information within the state file.
Correct answer
False
Explanation
The choice is correct because Terraform does not inherently encrypt sensitive values stored in the state file. It is recommended to use external encryption tools or implement secure practices to protect sensitive information within the state file.
Overall explanation
The correct answer is False.

Explanation:
Terraform does not encrypt sensitive values stored in the state file by default. The state file contains all the information Terraform uses to manage your infrastructure, including sensitive values like secrets or access keys. It is stored as plain text unless additional measures are taken.

To protect sensitive data in the state file, you should:

1. **Use secure storage for the state file**, such as a remote backend that supports encryption (e.g., AWS S3 with server-side encryption enabled).

2. **Limit access** to the state file using appropriate IAM or access controls.
3. Use tools like HashiCorp Vault or other secret management systems to securely manage sensitive data.

Without these precautions, sensitive data in the state file could be exposed if unauthorized individuals gain access to it.

Question 53Incorrect

When should you run terraform init?

Every time you run terraform apply

Explanation

Running `terraform init` every time you run `terraform apply` is not necessary. `terraform init` is used to initialize a Terraform working directory, and it only needs to be run once in a new Terraform project or when the configuration changes significantly.

After you run terraform plan for the first time in a new Terraform project and before you run terraform apply

Explanation

Running `terraform init` after the first `terraform plan` in a new Terraform project and before `terraform apply` is not the correct timing. `terraform init` should be run before the first `terraform plan` to initialize the working directory and download any necessary plugins.

Correct answer

After you start coding a new Terraform project and before you run terraform plan for the first time

Explanation

The correct timing to run `terraform init` is after you start coding a new Terraform project and before you run `terraform plan` for the first time. This ensures that the working directory is properly initialized and any required plugins are downloaded before the execution of the plan.

Your answer is incorrect

Before you start coding a new Terraform project

Explanation

Running `terraform init` before you start coding a new Terraform project is not necessary. `terraform init` is typically run after the project structure and configuration files are set up, to initialize the working directory and prepare it for further operations like `terraform plan` and `terraform apply`.

Overall explanation

The correct answer is **After you start coding a new Terraform project and before you run terraform plan for the first time**.

Explanation:

You should run `terraform init` after creating a new Terraform configuration file or starting a new Terraform project. This command initializes the working directory, sets up the backend (if configured), and downloads the necessary provider plugins and modules specified in your configuration.

Running `terraform init` ensures that Terraform has everything it needs to validate and execute the configuration. It should always be run before using commands like `terraform plan` or `terraform apply` for the first time in a new project.

- **Every time you run terraform apply** is incorrect because `terraform init` is only required when setting up or reinitializing the environment (e.g., after adding a new provider). It does not need to be run every time.
- **After you run terraform plan for the first time** is incorrect because Terraform needs initialization before running `terraform plan`. Without initialization, Terraform cannot validate or plan the configuration.
- **Before you start coding a new Terraform project** is incorrect because there is no configuration to initialize before coding begins. Initialization requires at least one `.tf` configuration file.

Question 54Correct

You are making changes to existing Terraform code to add some new infrastructure.

When is the best time to run `terraform validate`?

After you run terraform plan so you can validate that your state file is consistent with your infrastructure

Explanation

Running `terraform validate` after `terraform plan` is not the best approach because `terraform validate` is specifically designed to check the syntax and configuration of your Terraform code before any actual execution. It helps catch errors early in the development process and ensures that your code is valid before proceeding with planning or applying changes.

Your answer is correct

Before you run terraform plan so you can validate your code syntax

Explanation

The best time to run `terraform validate` is before you run `terraform plan` because it allows you to validate the syntax and configuration of your Terraform code before generating an execution plan. This helps identify any errors or issues in your code early on, making the planning and applying stages more efficient and error-free.

Before you run terraform apply so you can validate your infrastructure

Explanation

While running `terraform validate` before `terraform apply` is also important to ensure that your infrastructure configuration is valid, the primary purpose of `terraform validate` is to check the syntax and configuration of your Terraform code. Running it before `terraform plan` helps catch any code errors before proceeding with the planning and applying stages.

After you run terraform apply so you can validate your provider credentials

Explanation

Running `terraform validate` after `terraform apply` is not necessary for validating your provider credentials. `terraform validate` is focused on checking the syntax and configuration of your Terraform code, not on validating external credentials or resources. It is more appropriate to run validation checks before planning or applying changes to ensure the integrity of your Terraform code.

Overall explanation

The correct answer is **Before you run terraform plan so you can validate your code syntax**.

Explanation:

The `terraform validate` command checks the syntax and basic validity of your Terraform configuration files. It does not check if resources already exist or if your state file is consistent with your infrastructure, but it ensures that your code is free from syntax errors and is well-formed. Running `terraform validate` before `terraform plan` allows you to catch any configuration mistakes early, before attempting to generate a plan or apply changes.

- **After you run terraform plan so you can validate that your state file is consistent with your infrastructure** is incorrect because `terraform validate` does not check your state file or infrastructure; it only checks your configuration syntax.
- **Before you run terraform apply so you can validate your infrastructure** is incorrect because `terraform validate` focuses on the syntax of the configuration, not the infrastructure or state.
- **After you run terraform apply so you can validate your provider credentials** is incorrect because `terraform validate` does not check provider credentials or run after the apply phase.

Resources

terraform validate

Question 55Correct

Which of these commands makes your code more human readable?

terraform validate

Explanation

The `terraform validate` command is used to check whether the configuration files are valid and can be processed by Terraform. While it ensures the syntax is correct, it does not directly impact the human readability of the code.

terraform output

Explanation

The `terraform output` command is used to extract the values of output variables from the state file. It helps in retrieving specific information from the Terraform state but does not directly improve the human readability of the code.

terraform show

Explanation

The `terraform show` command is used to display the current state or a specific resource in a human-readable format. While it helps in understanding the current state of the infrastructure, it does not inherently make the code itself more human-readable.

Your answer is correct

terraform fmt

Explanation

The `terraform fmt` command is used to automatically format the Terraform configuration files according to a standard style. It helps in maintaining consistent formatting, indentation, and structure, making the code more readable and easier to understand for humans.

Overall explanation

The correct answer is **terraform fmt**.

Explanation:

The `terraform fmt` command automatically formats Terraform configuration files to follow consistent coding style conventions. This improves the readability of your code for humans by fixing indentation, aligning key-value pairs, and ensuring proper syntax. It is especially helpful in collaborative environments where multiple people work on the same codebase.

- **terraform validate** is incorrect because it only checks for syntax errors and the validity of the configuration but does not alter or format the code.
- **terraform output** is incorrect because it displays output values from the Terraform state, which is not related to formatting or making the code more readable.
- **terraform show** is incorrect because it displays the current state or plan in a human-readable format but does not improve the formatting of the configuration files themselves.

Resources

terraform fmt

Question 56Correct

Terraform configuration can only import from the public registry.

True

Explanation

This statement is incorrect. Terraform configuration can import modules not only from the public registry but also from private repositories, local files, and other sources. This flexibility allows users to customize their configurations and import modules from various locations based on their requirements.

Your answer is correct

False

Explanation

This statement is correct. Terraform configuration is not limited to importing modules only from the public registry. Users have the flexibility to import modules from private repositories, local files, and other sources as well, making the statement false.

Overall explanation

The correct answer is **False**.

Explanation:

Terraform configuration is not limited to importing only from the public Terraform Registry. While the public Terraform Registry provides many reusable modules, Terraform can also import modules and providers from other sources, including:

1. **Private Module Registry**: You can use private registries, such as those provided by Terraform Cloud or an enterprise setup, to host and manage your custom modules.
2. **Local Filesystem**: Terraform allows you to reference modules stored locally on your machine.
3. **Git Repositories**: You can pull modules directly from Git repositories (e.g., GitHub, GitLab, Bitbucket) by specifying the repository URL in the `source` argument.
4. **Object Storage Services**: Modules can be retrieved from cloud object storage, such as Amazon S3 or Google Cloud Storage.

These options make Terraform flexible in sourcing configurations, enabling both public and private usage.

Question 57Correct

What is the Terraform resource name of the following resource block?

```
mainresource “google_compute_instance” “main” {
           name = “test”
    }
```

test

Explanation

The resource name in Terraform is defined as the second argument in the resource block, which in this case is "main". Therefore, "test" is not the Terraform resource name in this scenario.

google

Explanation

The resource type in Terraform is defined as the first argument in the resource block, which in this case is "google_compute_instance". Therefore, "google" is not the Terraform resource name in this scenario.

Your answer is correct

main

Explanation

The Terraform resource name in this scenario is "main" as it is the second argument in the resource block. This is the correct choice as it accurately represents the resource name in the given code snippet.

compute_instance

Explanation

The resource type and instance name are combined to form the Terraform resource name. In this case, "compute_instance" is part of the resource type and not the resource name itself. Therefore, "compute_instance" is not the Terraform resource name in this scenario.

Overall explanation

The correct answer is main.

Explanation:

In a Terraform resource block, the **resource name** refers to the second string in the declaration, which is used as an identifier for the resource within the configuration.

The syntax of a resource block is as follows:

```
resource "<resource type>" "<resource name>" { ... }
```

In this example:

- `"google_compute_instance"` is the **resource type**, which defines the type of infrastructure resource (in this case, a Google Compute Instance).
- `"main"` is the **resource name**, used to uniquely identify this specific resource within the Terraform configuration.

Therefore, the resource name of this block is **main**.