#3 Terraform Associate Certification 003 - Results

Back to result overview

Attempt 1

All domains

- **57 all**
- **34 correct**
- **23 incorrect**
- **0 skipped**
- **0 marked**

Collapse all questions

Question 1 Incorrect

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

Which command should you use to show all of the resources that will be deleted? **(Choose two.)**

Correct selection

Run terraform plan -destroy

Explanation

Running `terraform plan -destroy` will show you a detailed execution plan of all the resources that Terraform will destroy. This command is specifically designed to provide visibility into the resources that will be deleted before actually executing the destruction process.

Correct selection

Run terraform destroy and it will first output all the resources that will be deleted before prompting for approval

Explanation

Running `terraform destroy` will indeed output a list of all the resources that will be deleted before prompting for approval. This command is useful for reviewing the resources that Terraform will remove and ensuring that you are aware of the impact of the destruction process.

Your selection is incorrect

Run terraform show -destroy

Explanation

Running `terraform show -destroy` is not a valid command in Terraform. The `terraform show` command is used to display the current state of the Terraform-managed infrastructure, but it does not have an option to specifically show resources that will be deleted.

Your selection is incorrect

Run terraform show -destroy

Explanation

Running `terraform show -destroy` is not a valid command in Terraform. The `terraform show` command is used to display the current state of the Terraform-managed infrastructure, but it does not have an option to specifically show resources that will be deleted.

Overall explanation

Correct Answer: Run `terraform plan -destroy` and `terraform destroy`

Explanation:

Running `terraform plan -destroy` shows all resources that will be destroyed without actually deleting them, allowing you to review what will be impacted by the destroy operation. This command is useful for understanding the scope of changes without applying them.

Additionally, running `terraform destroy` will also list all the resources that are about to be destroyed before prompting for confirmation. This ensures you get a second chance to review the items marked for deletion before proceeding.

- `terraform show -destroy` is not a valid command, and there is no such flag for `terraform show`.

Question 2Correct
Which Terraform collection type should you use to store **key/value** pairs?

set
Explanation
Sets in Terraform are unordered collections of unique elements, and they do not store key/value pairs. They are used to ensure uniqueness and can be helpful in scenarios where you need to eliminate duplicates from a collection.

tuple
Explanation
Tuples in Terraform are ordered collections of elements that can contain different data types, but they do not store key/value pairs. They are more suitable for grouping related values together rather than storing key/value pairs.

Your answer is correct
map
Explanation
Maps in Terraform are key/value pairs where each key is unique. This collection type is specifically designed to store key/value pairs, making it the appropriate choice for storing key/value pairs in Terraform configurations.

list
Explanation
Lists in Terraform are ordered collections of elements, but they do not store key/value pairs. They are useful for maintaining a specific order of elements and accessing them by index, rather than storing key/value pairs.

Overall explanation
Correct Answer: map

Explanation:

- **map**: This is the correct choice. In Terraform, a map is used to store key/value pairs. It is a collection type where each value is associated with a unique key, and the key is typically a string. You can access the values in the map using their corresponding keys.
  Example:
  ```
  variable "instance_types" {
   type = map(string)
   default = {
     "web"   = "t2.micro"
     "db"    = "t2.medium"
     "cache" = "t2.nano"
   }
  }
  ```
- **tuple**: This type is used to store an ordered collection of values that can be of different types, but it does not store key/value pairs.
- **set**: A set is an unordered collection of unique elements. It does not allow duplicates but does not use keys, so it isn't suitable for storing key/value pairs.
- **list**: A list is an ordered collection of elements, but it also does not use key/value pairs. It stores values in a sequential order rather than associating each value with a unique key.

Resources
Types and Values

Question 3Correct
You cannot install third party plugins using terraform init.

Your answer is correct
False
Explanation
This statement is correct. Terraform does support the installation of third-party plugins using the `terraform init` command. Third-party plugins can be installed by specifying the source in the configuration files, allowing users to extend Terraform's capabilities with additional providers, provisioners, and modules.

True
Explanation

This statement is incorrect. Terraform allows users to install third-party plugins using the `terraform init` command. Third-party plugins can extend Terraform's functionality by providing additional providers, provisioners, and modules. Users can specify the source of the plugin in the configuration files, and Terraform will download and install the plugin during the initialization process.

Overall explanation

Correct Answer: False

Explanation:

You can install third-party plugins using `terraform init`. When you run `terraform init`, Terraform automatically installs the required provider plugins, which can include both official providers (like AWS, Azure, Google Cloud) and third-party plugins. If you reference a third-party provider module in your configuration, Terraform will download and install it when you run `terraform init`.

In addition to installing providers from the Terraform Registry, Terraform can also install plugins from custom or third-party sources, such as a private registry or a GitHub repository, as long as the appropriate source URL is specified. This capability is one of the core functions of `terraform init`.

Hence, the statement "You cannot install third-party plugins using terraform init" is **False**.

Resources

Terraform CLI Command: init

Question 4Correct

You have some Terraform code and a variable definitions file named **dev.auto.tfvars** that you tested successfully in the dev environment. You want to deploy the same code in the staging environment with a separate variable definition file and a separate state file.

Which two actions should you perform? **(Choose two.)**

Create a new Terraform provider for staging

Explanation

Creating a new Terraform provider for staging is not necessary for deploying the code in the staging environment with separate state and variable files. Providers are used to interact with different cloud platforms or services, and they do not need to be recreated for each environment.

Your selection is correct

Create a new Terraform workspace for staging

Explanation

Creating a new Terraform workspace for staging is essential to isolate the state and variables for the staging environment. Workspaces allow you to manage multiple environments within the same Terraform configuration, keeping the resources separate and organized.

Your selection is correct

Write a new staging.auto.tfvars variable definition file and run Terraform with the var-file=”staging.auto.tfvars” flag

Explanation

Writing a new staging.auto.tfvars variable definition file and running Terraform with the var-file=”staging.auto.tfvars” flag is the correct action to provide separate variable values for the staging environment. This ensures that the Terraform code uses the correct variables specific to the staging environment.

Copy the existing terraform.tfstate file and save it as staging.terraform.tfstate

Explanation

Copying the existing terraform.tfstate file and saving it as staging.terraform.tfstate is not necessary for deploying the code in the staging environment with a separate state file. Each environment should have its own state file to manage infrastructure resources independently.

Add new Terraform code (*.tf files) for staging in the same directory

Explanation

Adding new Terraform code (*.tf files) for staging in the same directory is not the recommended approach for managing separate environments. It is best practice to keep the code for each environment isolated in separate directories or repositories to avoid confusion and ensure clean separation of resources.

Overall explanation

Correct Answer:

- **Write a new staging.auto.tfvars variable definition file and run Terraform with the `-var-file="staging.auto.tfvars"` flag**
- **Create a new Terraform workspace for staging**

Explanation:

1. **Write a new staging.auto.tfvars variable definition file and run Terraform with the `-var-file="staging.auto.tfvars"` flag:**
    - The `staging.auto.tfvars` file will contain the environment-specific values for the staging environment. You will use the `-var-file="staging.auto.tfvars"` flag when running Terraform commands to specify this file for the staging environment. This allows Terraform to use different configurations from the ones used in the `dev.auto.tfvars` file.
2. **Create a new Terraform workspace for staging:**
    - Terraform workspaces allow you to maintain separate state files for different environments (e.g., dev, staging, production). By creating a new workspace for staging, you ensure that Terraform maintains a separate state file for the staging environment, avoiding potential conflicts with the state file used for the dev environment.

Why the others are incorrect:

- **Copy the existing terraform.tfstate file and save it as staging.terraform.tfstate:**
    - Copying the state file manually is not the best practice. Workspaces provide a more structured and organized way to manage state files for different environments. Manually copying the state file could lead to inconsistencies and confusion.
- **Create a new Terraform provider for staging:**
    - You don't need to create a new provider for staging unless you have a completely different provider configuration for the staging environment (e.g., different cloud credentials or API URLs). In most cases, you would manage different environments using separate workspaces and variable files.
- **Add new Terraform code (*.tf files) for staging in the same directory:**
    - Terraform code can be reused across environments, so there's no need to add new code for staging in the same directory unless you have specific configurations for staging. You typically use variable files and workspaces to manage differences between environments while keeping the codebase the same.

Question 5Incorrect

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

Correct answer

False

Explanation

The correct choice is False because any manual modifications made by your teammate to the infrastructure component you are working on will not be considered in the execution plan generated by terraform plan. Terraform plan only takes into account the current state of the infrastructure at the time of running the command, so the execution plan for terraform apply will not be the same if there are external changes made outside of Terraform.

Your answer is incorrect

True

Explanation

If your teammate manually modified the infrastructure component while you were reviewing the changes with terraform plan, the execution plan for terraform apply will not be the same as what you saw in your local environment. Terraform plan generates an execution plan based on the current state of the infrastructure at the time of running the command. Any manual changes made by your teammate will not be reflected in the plan generated by terraform plan.

Overall explanation

Correct Answer: False

Explanation:

When you run `terraform plan`, it generates an execution plan based on the current state of the infrastructure as recorded in the Terraform state file. If your teammate has manually modified the infrastructure, the state in Terraform no longer matches the actual infrastructure. This discrepancy means that the next time you run `terraform apply`, Terraform will detect that the infrastructure has changed and will update the plan accordingly to reflect the actual state, which could differ from what was previously planned.

- **True** is incorrect because the execution plan would likely differ after detecting manual changes made to the infrastructure.

Question 6 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 the state file to track the current state of the infrastructure. Since the load balancer port was manually changed outside of Terraform, the state file is no longer accurate, causing Terraform to fail with an error when attempting to apply changes.

Terraform will change the port back to 80 in your code.

Explanation

Terraform operates based on the desired state configuration defined in the Terraform code. Even if the load balancer port was manually changed to 443 before running terraform apply, Terraform will still enforce the desired state specified in the code, which is to have the port set to 443. Therefore, Terraform will not change the port back to 80 in your code.

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

Terraform is designed to manage infrastructure as code and enforce the desired state configuration. In this scenario, since the load balancer port was manually changed to 443, Terraform will detect that the current state does not match the desired state and will not make any changes to the Load Balancer. It will update the state file to reflect the changes made outside of Terraform.

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

Explanation

Terraform follows the desired state configuration, which in this case is to have the load balancer port set to 443. Even if the port was manually changed to 443 before running terraform apply, Terraform will still enforce the desired state and keep the port at 443.

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:

In this scenario, Terraform is designed to compare the current state (from the state file) with the actual infrastructure and the configuration in your Terraform code. Since your teammate manually updated the load balancer to port 443 through the cloud provider console, Terraform will detect that the port is already set to 443 and no changes are needed.

Here's the breakdown:

- **Terraform will not fail** with an error because the state file can still be synced with the actual infrastructure. Terraform can detect changes in the infrastructure and reconcile them by updating the state file.
- **Terraform will not change the port to 80 and back to 443** because it recognizes that the load balancer is already at port 443, matching the intended configuration.
- **Terraform will update the state file** to reflect that the load balancer is indeed at port 443, and no further changes will be made to the infrastructure.

Terraform is meant to work by maintaining the infrastructure's current state in the state file and will only make necessary changes when there's a mismatch between the state file and the current infrastructure configuration.

Question 7Correct

`terraform apply` will fail if you have not run `terraform plan` first to update the plan output.

Your answer is correct

False

Explanation

This statement is correct. Terraform apply does not necessarily require running terraform plan first to update the plan output. While it is a best practice to run terraform plan to review the changes before applying them, it is not a strict requirement for the apply command to function properly. Terraform apply can be used directly to apply the changes specified in the configuration without explicitly running terraform plan.

True

Explanation

This statement is incorrect. Terraform apply does not require running terraform plan first to update the plan output. While it is recommended to run terraform plan to preview the changes before applying them, it is not mandatory for the apply command to work successfully. Terraform apply can be used independently to apply the changes defined in the configuration.

Overall explanation

Correct Answer: False

Explanation:

Running `terraform apply` does not require you to execute `terraform plan` first. While `terraform plan` is a useful command that allows you to preview changes and ensure that the configuration aligns with your expectations, it is not mandatory to run it before `terraform apply`.

When you run `terraform apply`, Terraform automatically generates a plan on the spot and applies any necessary changes to achieve the desired infrastructure state defined in your configuration. Running `terraform plan` first is often a best practice, especially in production environments, as it allows you to review changes before they are applied, but it is not a strict requirement for `terraform apply` to succeed.

Question 8Incorrect

Terraform destroy is the only way to remove infrastructure.

Correct answer

False

Explanation

This statement is correct. Terraform destroy is not the only way to remove infrastructure. There are other methods and tools available to remove infrastructure provisioned by Terraform, such as manual deletion or using other automation tools.

Your answer is incorrect

True

Explanation

This statement is incorrect. While Terraform destroy is a common way to remove infrastructure provisioned by Terraform, it is not the only way. Infrastructure can also be removed manually or through other automation tools or scripts outside of Terraform.

Overall explanation

Correct Answer: False

Explanation:

While `terraform destroy` is the primary command used to destroy all resources defined in a Terraform configuration, it is not the only way to remove infrastructure.

- You can also manually delete resources directly from the cloud provider's console or via its API, but this would cause a mismatch with the state file. This is not recommended because Terraform would no longer be aware of the resources that were manually deleted.
- You can remove individual resources from the Terraform state using the `terraform state rm` command without actually destroying the resources. This allows Terraform to forget about the resource, but the resource itself will remain in the cloud environment.

Thus, `terraform destroy` is a convenient way to remove infrastructure, but it is not the *only* way to do so.

Question 9Incorrect

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

Your selection is incorrect

A full audit trail of the request and fulfillment process is generated

Explanation

Generating a full audit trail of the request and fulfillment process can actually help in tracking and monitoring infrastructure changes, ensuring compliance, and providing transparency. While this process may add some overhead, it is not directly related to slowing down infrastructure provisioning or limiting scalability.

Correct selection

As additional resources are required, more tickets are submitted

Explanation

As additional resources are required, more tickets need to be submitted, which can create bottlenecks in the infrastructure provisioning process. This can lead to delays, inefficiencies, and difficulties in scaling up or down based on demand.

Your selection is correct

A request must be submitted for infrastructure changes

Explanation

Requiring a request to be submitted for infrastructure changes can introduce delays in provisioning as each change needs to go through a ticketing system. This can slow down the process and limit the ability to scale quickly, especially in dynamic environments where rapid changes are needed.

A catalog of approved resources can be accessed from drop down lists in a request form

Explanation

Accessing a catalog of approved resources from drop-down lists in a request form can actually streamline the process of requesting infrastructure changes. While this may introduce some level of control and standardization, it is not directly related to the potential slowdown in provisioning or scalability limitations caused by a ticket-based system.

Overall explanation

Correct Answer:

- **A request must be submitted for infrastructure changes**
- **As additional resources are required, more tickets are submitted**

Explanation:

A ticket-based system, while useful for tracking and controlling infrastructure changes, can introduce delays and limitations in the provisioning process:

- **A request must be submitted for infrastructure changes**: Every infrastructure change or addition requires a manual request to be submitted through the ticketing system. This can slow down provisioning, as changes must wait for approval, and the ticketing system may introduce bottlenecks in the process. This is a significant limitation for teams that need to make fast changes in dynamic environments.
- **As additional resources are required, more tickets are submitted**: For each additional resource or infrastructure change, a new ticket is often submitted. As the need for resources grows, the number of tickets also increases, adding administrative overhead and slowing down the process, especially when dealing with large-scale or rapidly evolving infrastructure needs.

The other options are less relevant to the slowdown and scaling limitation:

- **A full audit trail of the request and fulfillment process is generated**: While this is a benefit for tracking changes, it does not inherently slow down the process. It provides visibility, which may actually help in governance and compliance, but does not directly impact speed or scalability.
- **A catalog of approved resources can be accessed from drop down lists in a request form**: This improves user experience and speeds up the request process, as it simplifies the selection of resources. It does not inherently slow down provisioning or limit scalability.

Question 10Correct

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

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

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

terraform state

Explanation

The `terraform state` command is used to perform operations on Terraform state, such as moving, deleting, or importing resources. However, it is not the command used to migrate the current state file to a new S3 remote backend.

terraform push

Explanation

The `terraform push` command is not a valid Terraform command. Migrating the state file to a new S3 remote backend requires using the `terraform init` command.

terraform refresh

Explanation

The `terraform refresh` command is used to update the state file with the real-world infrastructure. It does not migrate the state file to a new backend, so it is not the correct command for this scenario.

Your answer is correct

terraform init

Explanation

The `terraform init` command is used to initialize a working directory containing Terraform configuration files. When you have updated your backend configuration to use S3, running `terraform init` will migrate your current state file to the new S3 remote backend.

Overall explanation

The correct command is: terraform init

Explanation:

- The `terraform init` command is used to initialize a Terraform working directory. When you specify a new backend configuration (as in the `backend.tf` file), running `terraform init` will migrate the current state file to the new backend (in this case, S3).

- During this process, Terraform detects that the backend configuration has changed and prompts you to confirm the migration of the state file.

Why the others are incorrect:

- **terraform state**: This command is for managing the state file (e.g., listing, pulling, or moving resources within the state) but does not handle backend migrations.
- **terraform refresh**: This updates the Terraform state with the latest infrastructure changes but does not deal with backend configurations or state migrations.
- **terraform push**: This command is used for manually uploading a state file to a remote backend, but it is not commonly used and is deprecated in many cases.

Question 11 Correct

Which of the following is the correct way to pass the value in the variable **num_servers** into a module with the input servers in HCL2?

servers = num_servers

Explanation

This syntax "servers - num_servers" is incorrect for passing the value of a variable into a module in HCL2. The correct syntax includes the "var." prefix before the variable name to access its value.

Your answer is correct

servers = var.num_servers

Explanation

The correct way to pass the value of a variable into a module in HCL2 is by using the syntax "var.variable_name". In this case, "servers - var.num_servers" correctly passes the value of the variable num_servers into the input servers of the module.

servers = var(num_servers)

Explanation

The syntax "servers - var(num_servers)" is incorrect for passing the value of a variable into a module in HCL2. The correct way to reference a variable in HCL2 is by using the "var." prefix followed by the variable name.

$(var.num_servers)

Explanation

The syntax "$(var.num_servers)" is incorrect for passing the value of a variable into a module in HCL2. In HCL2, the correct way to reference a variable is by using the "var." prefix followed by the variable name.

Overall explanation

The correct way to pass the value in the variable `num_servers` into a module with the input `servers` in HCL2 is:

servers = var.num_servers

Explanation:

- The correct syntax to reference a variable in Terraform HCL2 is `var.<variable_name>`, so `var.num_servers` correctly references the value of the `num_servers` variable. This value can then be passed into the module using the `servers` parameter.
- The other options are incorrect because they do not follow the proper syntax for referencing variables in HCL2.

Question 12 Correct

Once a new Terraform backend is configured with a Terraform code block, which command(s) is (are) used to migrate the state file?

terraform destroy, then terraform apply

Explanation

The `terraform destroy` command is used to destroy the Terraform-managed infrastructure. Following it with `terraform apply` would recreate the infrastructure, but it does not specifically migrate the state file to a new backend, so this combination of commands is not the correct way to migrate the state file.

terraform push

Explanation

The `terraform push` command is used to push local state changes to the remote backend. It is not specifically designed for migrating the state file to a new backend, so it is not the correct command for this task.

terraform apply

Explanation

The `terraform apply` command is used to apply the changes described in the Terraform configuration files to the infrastructure. It does not specifically migrate the state file to a new backend, so it is not the correct command for this task.

Your answer is correct

terraform init

Explanation

The `terraform init` command is used to initialize a working directory containing Terraform configuration files. When a new backend is configured in the Terraform code block, running `terraform init` will migrate the state file to the new backend. This makes it the correct command for migrating the state file after configuring a new backend.

Overall explanation

Correct Answer: `terraform init`

Explanation:

When you configure a new backend in Terraform, running `terraform init` is the correct and only command required to initialize the backend and, if needed, migrate the state file to the new backend. During this initialization, Terraform will detect the backend change and prompt you to confirm if you want to migrate the existing state file from the current backend to the new backend. This ensures that all state information is safely moved without manual intervention.

Incorrect Options:

- **`terraform apply`**: This command applies the configuration and provisions resources but does not handle backend initialization or state migration.
- **`terraform push`**: This command is not commonly used for state migration and is related to pushing state information in specific scenarios, typically not for backend migration.
- **`terraform destroy` then `terraform apply`**: This would delete all managed infrastructure and then recreate it, which is unnecessary and would result in resource loss and downtime. It does not relate to backend initialization or migration.

Using `terraform init` is both secure and efficient for backend migration.

Question 13Correct

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

The DevOps team is tasked with automating a manual provisioning process

Explanation

Automating a manual provisioning process is a common use case for AWS CloudFormation and aligns with the tool's purpose of infrastructure as code. While this task may require some effort to convert manual processes into CloudFormation templates, it is not a specific challenge related to using CloudFormation as the provisioning method.

Your answer is correct

The organization decides to expand into Azure and wishes to deploy new infrastructure using their existing codebase

Explanation

The challenge arises when the organization decides to expand into Azure and wishes to deploy new infrastructure using their existing CloudFormation codebase. CloudFormation is specific to AWS and does not support provisioning resources in Azure. This scenario would require the DevOps team to adopt a different infrastructure as code tool that is compatible with Azure.

The team is asked to manage a new application stack built on AWS-native services

Explanation

Managing a new application stack built on AWS-native services is not inherently a challenge for a DevOps team using AWS CloudFormation. CloudFormation is designed to support the provisioning and management of AWS resources, including native services, making it a suitable tool for managing new application stacks.

The team is asked to build a reusable code base that can deploy resources into any AWS region

Explanation

Building a reusable code base that can deploy resources into any AWS region is not a challenge for a DevOps team using AWS CloudFormation. CloudFormation templates can be parameterized to support deployment in different regions, making it feasible to deploy resources across multiple regions with the same code base.

Overall explanation

Correct Answer: The organization decides to expand into Azure and wishes to deploy new infrastructure using their existing codebase.

Explanation:

AWS CloudFormation is a tool specifically designed for provisioning and managing infrastructure on AWS. It uses AWS-specific templates written in JSON or YAML to define resources.

- **The team is asked to build a reusable code base that can deploy resources into any AWS region**: This is possible with AWS CloudFormation, as it allows you to specify AWS regions within the configuration, making it flexible for deployment across different regions.
- **The team is asked to manage a new application stack built on AWS-native services**: This is exactly what AWS CloudFormation is designed for, as it works seamlessly with AWS services.
- **The organization decides to expand into Azure and wishes to deploy new infrastructure using their existing codebase**: AWS CloudFormation cannot be directly used for provisioning resources in Azure, as it is specifically built for AWS. This would be a challenge as the team would need to learn and adopt a new tool like Azure Resource Manager (ARM) templates, Terraform, or another multi-cloud provisioning tool to manage Azure infrastructure.
- **The DevOps team is tasked with automating a manual provisioning process**: This is one of the strengths of AWS CloudFormation, as it automates resource provisioning and management in AWS environments.

Thus, the biggest challenge is expanding to Azure using an AWS-specific tool.

Resources

Terraform vs. CloudFormation

Question 14Incorrect

What does terraform destroy do?

Your answer is incorrect

Destroy all infrastructure in the configured Terraform provider

Explanation

Terraform destroy does not target a specific provider to destroy infrastructure. It removes all the resources defined in the Terraform state file, regardless of which provider they belong to.

Destroy all Terraform code files in the current directory while leaving the state file intact

Explanation

This choice is incorrect because terraform destroy does not delete the Terraform code files themselves. It only removes the infrastructure resources that were created based on those code files.

Correct answer

Destroy all infrastructure in the Terraform state file

Explanation

Terraform destroy command removes all the infrastructure resources that are defined in the Terraform state file. It deletes all the resources that were created during the apply phase, effectively cleaning up the environment.

Destroy the Terraform state file while leaving infrastructure intact

Explanation

Terraform destroy does not delete the Terraform state file itself. It only removes the infrastructure resources that are managed by Terraform, leaving the state file intact for future use or reference.

Overall explanation

Correct Answer: Destroy all infrastructure in the Terraform state file

Explanation:

The `terraform destroy` command removes all infrastructure resources that are currently managed by the state file, based on the existing Terraform configuration. It doesn't affect the Terraform configuration files or the state file itself; rather, it uses the state file to identify the resources that were previously created and removes them from the infrastructure. This action helps ensure that any resources tracked by Terraform are destroyed as specified, allowing for a clean teardown of infrastructure.

- **Destroying Terraform code files** or **the state file** is not part of `terraform destroy`. It exclusively targets the infrastructure, not the Terraform files or state data.
- **Destroying all infrastructure in the configured Terraform provider** would potentially affect resources not managed by Terraform, which is not what `terraform destroy` does.

Resources

Destroy infrastructure

Question 15 Correct

Which of the following locations can Terraform use as a private source for modules? **(Choose two.)**

Your selection is correct

Private repository on GitHub

Explanation

Terraform can use a private repository on GitHub as a private source for modules. This allows organizations to store their Terraform modules privately on GitHub, restricting access to authorized users only. This provides a secure way to manage and share Terraform modules within a specific team or organization.

Public Terraform Module Registry

Explanation

The Public Terraform Module Registry is not a private source for modules. It is a public repository where users can share and discover Terraform modules created by the community. Terraform modules from the Public Terraform Module Registry are accessible to anyone.

Your selection is correct

Internally hosted SCM (Source Control Manager) platform

Explanation

Terraform can use an internally hosted SCM platform as a private source for modules. This allows organizations to securely store and manage their Terraform modules within their own infrastructure, ensuring control over access and versioning.

Public repository on GitHub

Explanation

A public repository on GitHub is not a private source for modules. Public repositories on GitHub are accessible to anyone, and the Terraform modules stored in public repositories are not restricted to specific users or organizations.

Overall explanation

Correct Answers:

1. **Internally hosted SCM (Source Control Manager) platform**
2. **Private repository on GitHub**

Explanation:

Terraform can use various private sources for modules, which is useful for accessing modules that aren't publicly available or are internally managed for an organization.

- **Internally hosted SCM (Source Control Manager) platform** can serve as a private source for modules by hosting Terraform code. Terraform can be configured to pull modules from SCM platforms like GitLab, Bitbucket, or self-hosted Git systems.
- **Private repository on GitHub** can also be used as a private source for modules. By specifying authentication credentials, Terraform can securely access private GitHub repositories to retrieve modules.

Other options:

- **Public Terraform Module Registry** and **Public repository on GitHub** are not considered private sources. They are publicly accessible and therefore do not require any special configuration for private access.

Question 16 Correct

What is the purpose of a Terraform workspace in either open source or enterprise?

Provides limited access to a cloud environment

Explanation

Workspaces in Terraform do not provide limited access to a cloud environment. They are primarily used for managing state files and configurations, not for controlling access levels to cloud environments. Access control in cloud environments is typically managed through IAM policies and permissions.

Your answer is correct

Workspaces allow you to manage collections of infrastructure in state files

Explanation

Workspaces in Terraform allow you to manage different collections of infrastructure resources within separate state files. This helps in organizing and isolating different environments or configurations, making it easier to manage and apply changes without affecting other resources.

A method of grouping multiple infrastructure security policies

Explanation

Terraform workspaces are not specifically designed for grouping multiple infrastructure security policies. While Terraform does support managing security configurations, workspaces are primarily used for managing infrastructure resources and state files.

A logical separation of business units

Explanation

A logical separation of business units is not the primary purpose of Terraform workspaces. While workspaces can help in organizing infrastructure resources, their main function is to manage state files and configurations, not to separate business units.

Overall explanation

Correct Answer: Workspaces allow you to manage collections of infrastructure in state files

Explanation:

In both Terraform Open Source and Terraform Enterprise, **workspaces** are used to manage different states for different environments or configurations. Each workspace can maintain a separate state file, allowing you to manage multiple environments (like development, staging, and production) without overlapping or interfering with each other. This provides a logical separation of infrastructure and enables managing different states without conflict.

- **A logical separation of business units**: While workspaces can provide some separation of infrastructure, they are not specifically designed for managing business units. Their main purpose is to handle different states and configurations for infrastructure.
- **A method of grouping multiple infrastructure security policies**: This is incorrect as workspaces do not directly manage security policies; they are primarily for managing state files and infrastructure configurations.
- **Provides limited access to a cloud environment**: Workspaces do not inherently control access to the cloud environment. Access is typically controlled through credentials and permissions, not workspaces.

Resources

Terraform Workspaces

Question 17 Incorrect

As a member of an operations team that uses **infrastructure as code** (IaC) practices, you are tasked with making a change to an infrastructure stack running in a public cloud.

Which pattern would follow IaC **best practice**s for making a change?

Make the change via the public cloud API endpoint

Explanation

Making changes directly via the public cloud API endpoint can be efficient and automated, but it may not align with IaC best practices. Changes made through APIs should still go through a version-controlled process to ensure proper documentation and collaboration.

Your answer is incorrect

Clone the repository containing your infrastructure code and then run the code

Explanation

Cloning the repository containing the infrastructure code and running the code locally may not follow best practices for IaC. Changes should be made in a version-controlled environment and go through a standardized process to ensure consistency and traceability.

Make the change programmatically via the public cloud CLI

Explanation
Making changes programmatically via the public cloud CLI is a step in the right direction as it allows for automation and repeatability. However, it may still lack the necessary review and approval processes that are essential in IaC best practices.
Correct answer
Submit a pull request and wait for an approved merge of the proposed changes
Explanation
Submitting a pull request and waiting for an approved merge of proposed changes is the best practice for making changes to infrastructure code in IaC. This process allows for peer review, documentation, and version control, ensuring that changes are properly tested and validated before being implemented.
Use the public cloud console to make the change after a database record has been approved
Explanation
Using the public cloud console to make changes directly bypasses the version control and review process typically associated with IaC. This method can lead to inconsistencies and potential issues with tracking changes over time.
Overall explanation
Correct Answer: Submit a pull request and wait for an approved merge of the proposed changes

Explanation:

In infrastructure as code (IaC) best practices, changes to infrastructure should be made through version-controlled code. The recommended workflow involves submitting a pull request (PR) with the proposed changes. This allows for peer review, testing, and approval before the code is merged and applied to the infrastructure. By following this process, you ensure that changes are tracked, auditable, and can be rolled back if needed.

- **Cloning the repository and running the code** is not ideal because it skips the review and approval process, which can lead to unintended or unreviewed changes.
- **Using the public cloud console** directly to make changes violates IaC principles because it introduces manual steps that are not tracked in version control and can cause discrepancies between the code and the actual infrastructure.
- **Making the change programmatically via the cloud CLI** also violates best practices because it bypasses the code review process and can lead to manual changes that aren't reflected in the version-controlled code.
- **Making the change via the public cloud API endpoint** is similar to using the CLI; it also bypasses the IaC workflow and lacks the benefits of version control, peer review, and testing.

By using the pull request model, you align with IaC best practices, ensuring that changes are reviewed, approved, and documented in a transparent and controlled manner.
Question 18 Correct
You have a Terraform configuration that defines a single virtual machine with no references to it. You have run terraform apply to create the resource, and then removed the resource definition from your Terraform configuration file.

What will happen when you run terraform apply in the working directory again?
Terraform will remove the virtual machine from the state file, but the resource will still exist
Explanation
Terraform manages the state of resources in a state file. If you remove the resource definition from the configuration file and run terraform apply again, Terraform will update the state file to reflect the absence of the virtual machine. However, the resource will still exist until Terraform is explicitly instructed to destroy it.
Your answer is correct
Terraform will destroy the virtual machine
Explanation
When you run terraform apply after removing the resource definition, Terraform will recognize that the virtual machine is no longer part of the configuration and will proceed to destroy the resource to match the desired state defined in the configuration.
Terraform will error
Explanation
Terraform will not error out in this scenario. Instead, it will recognize that the virtual machine is no longer part of the configuration and will take action accordingly, which in this case would be to destroy the resource.

Nothing

Explanation

If you have removed the resource definition from your Terraform configuration file, Terraform will not have any reference to the virtual machine. Therefore, running terraform apply again will not have any impact on the existing resource.

Overall explanation

Correct Answer: Terraform will destroy the virtual machine

Explanation:

When you remove a resource definition from your Terraform configuration file and then run `terraform apply`, Terraform identifies that the resource is no longer part of the desired state as defined by the configuration. Since the resource is still in the state file but no longer has a corresponding configuration, Terraform will determine that it should be destroyed to match the desired state. This is part of Terraform's declarative approach, which aims to ensure that the actual infrastructure matches what is defined in the configuration file.

- **Nothing** is incorrect because Terraform will detect the discrepancy and take action.
- **Terraform will error** is incorrect as Terraform will proceed with the apply process to resolve the difference between configuration and state.
- **Terraform will remove the virtual machine from the state file, but the resource will still exist** is incorrect because Terraform will actively destroy the resource rather than simply removing it from the state file.

Question 19 Incorrect

You need to write some Terraform code that adds 42 firewall rules to a security group as shown in the example.

```
    resource “aws_security_ group” "many_rules” {
       name = "many-rules”
       ingress {
         from_port = 443
         to_port = 443
         protocol = “tcp”
         cidr_blocks = "0.0.0.0/0"
       }
    }
```

What can you use to avoid writing 42 different nested ingress config blocks by hand?

A for block

Explanation

Terraform does not have a built-in "for" block for iterating over resources or configurations. Using a "for" block in this context would not be a valid Terraform syntax for dynamically generating multiple firewall rules within a security group.

A count loop

Explanation

Using a count loop in Terraform would require manually specifying each individual rule, which would not be efficient or scalable for adding 42 firewall rules. This approach would involve duplicating the same configuration block multiple times, leading to code duplication and potential errors.

Correct answer

A dynamic block

Explanation

A dynamic block in Terraform allows for generating multiple configurations dynamically based on a list or map of values. In this scenario, using a dynamic block would be the most appropriate approach to avoid manually writing 42 different nested ingress config blocks. It enables you to iterate over a list of firewall rules and generate the necessary configurations efficiently.

Your answer is incorrect

A for each block

Explanation

While Terraform does support the "for_each" meta-argument for iterating over a map or set of resources, it is not the most suitable option for dynamically generating multiple firewall rules within a security group. The "for_each" block is typically used for managing multiple instances of a resource with unique identifiers, rather than generating a large number of similar configurations.

Overall explanation
Correct Answer: A dynamic block

Explanation:

In Terraform, a **dynamic block** can be used to programmatically generate multiple nested blocks within a resource. In this case, you want to add multiple `ingress` blocks to an `aws_security_group` resource without manually writing each one. By using a dynamic block, you can create these ingress configurations based on a list of input values, simplifying the code significantly and avoiding repetitive block definitions.

Example Code:

Here's how you might use a dynamic block to generate multiple ingress rules:

```
resource "aws_security_group" "many_rules" {
 name = "many-rules"

 dynamic "ingress" {
   for_each = var.rules_list
   content {
     from_port   = ingress.value.from_port
     to_port     = ingress.value.to_port
     protocol    = ingress.value.protocol
     cidr_blocks = ingress.value.cidr_blocks
   }
 }
}
```

In this example:

- `var.rules_list` could be a list of objects with the `from_port`, `to_port`, `protocol`, and `cidr_blocks` values for each rule.
- Terraform will iterate through each element in `var.rules_list` to create multiple `ingress` blocks automatically.

Explanation of Other Options:

- **A count loop**: While `count` can duplicate resources, it does not work inside nested blocks within resources.
- **A for block**: `for` blocks are used mainly for generating lists or maps and are not used for creating nested blocks in resources.
- **A for_each block**: `for_each` is used to iterate over collections but, like `count`, does not work for creating nested blocks directly within a resource.

Thus, the **dynamic block** is the correct approach here for generating multiple nested blocks within a single resource.

Resources
[Dynamic Blocks]()

Question 20Correct
A Terraform output that sets the "sensitive" argument to true will not store that value in the state file.

True
Explanation
If a Terraform output sets the "sensitive" argument to true, it means that the value of that output will be marked as sensitive and will not be displayed in console output. However, this does not prevent the value from being stored in the state file. The sensitive flag only affects the visibility of the value during Terraform operations.

Your answer is correct
False
Explanation

The correct choice is False because even if a Terraform output sets the "sensitive" argument to true, the value will still be stored in the state file. The sensitive flag only controls the visibility of the value during Terraform operations and does not prevent it from being stored in the state file.

Overall explanation

Correct Answer: False

Explanation:

Setting the `sensitive` argument to `true` in a Terraform output does not prevent the value from being stored in the state file. The `sensitive` argument only ensures that the output value is not displayed in the Terraform CLI output (e.g., when running `terraform apply` or `terraform output`). It helps prevent the sensitive data from being shown to the user in the terminal, but it **does** get stored in the state file for use in the Terraform configuration and for state management purposes.

- If you want to keep sensitive data **out** of the state file, you would need to use a secure backend and ensure the appropriate encryption is in place (e.g., using an encrypted backend like Terraform Cloud or AWS S3 with encryption enabled).

Thus, the `sensitive = true` setting only affects visibility in the output, not in the state file.

Resources

Output Values

Question 21 Correct

Which configuration consistency errors does terraform validate report?

Differences between local and remote state

Explanation

Terraform validate does not specifically check for differences between local and remote state. This type of error would typically be identified during the terraform plan or terraform apply stages.

A mix of spaces and tabs in configuration files

Explanation

Terraform validate does not report on issues related to the mix of spaces and tabs in configuration files. This type of error is typically handled by code editors or linters, not by Terraform itself.

Your answer is correct

Declaring a resource identifier more than once

Explanation

Declaring a resource identifier more than once is a configuration consistency error that Terraform validate does report. This type of error can lead to conflicts and unexpected behavior in the infrastructure deployment process.

Terraform module isn't the latest version

Explanation

Terraform validate does not check if a Terraform module is the latest version. This is a separate concern that can be managed by tools like Terraform Cloud or Terraform Enterprise.

Overall explanation

Correct Answer: Declaring a resource identifier more than once

Explanation:

The `terraform validate` command checks for syntax or configuration errors within the Terraform configuration files, but it doesn't perform a full plan or check against the actual infrastructure or the state file. One of the key errors it will detect is if a resource identifier (like the name of a resource block) is declared more than once in a configuration file, as this creates ambiguity about which definition to use.

- **A mix of spaces and tabs in configuration files** is incorrect because `terraform validate` does not check for formatting issues; this would be handled by `terraform fmt`.

- **Differences between local and remote state** is incorrect, as `terraform validate` only checks the configuration files and does not interact with the state file. Differences between local and remote state would be identified during `terraform plan`.
- **Terraform module isn't the latest version** is incorrect, as `terraform validate` does not check if modules are up-to-date. This would need to be managed manually or with tools like `terraform get -update`.

Question 22Correct

How would you reference the attribute "name" of this fictitious resource in HCL?

```
    resource “kubernetes_namespace” “example” {
         name = “test”
    }
```

None of the above

Explanation

None of the above choices are correct. The correct way to reference the attribute "name" of the fictitious resource in HCL is "kubernetes_namespace.example.name". This format follows the proper syntax for referencing attributes of resources in HCL.

resource.kubernetes_namespace.example.name

Explanation

The format "resource.kubernetes_namespace.example.name" is incorrect as it does not follow the correct syntax for referencing attributes of a resource in HCL. The correct format should include the resource type, instance name, and attribute name separated by periods.

kubernetes_namespace.test.name

Explanation

The format "kubernetes_namespace.test.name" is incorrect as it does not include the instance name "example" of the resource. In HCL, when referencing attributes of a resource, you need to specify the resource type, instance name, and attribute name in the correct order.

data.kubernetes_namespace.name

Explanation

The format "data.kubernetes_namespace.name" is incorrect as it includes "data" which is typically used for data sources in Terraform, not for referencing attributes of resources. In this case, the resource is a "kubernetes_namespace" and should be referenced accordingly.

Your answer is correct

kubernetes_namespace.example.name

Explanation

The format "kubernetes_namespace.example.name" is correct. In HCL, when referencing attributes of a resource, you need to specify the resource type, instance name, and attribute name in the correct order separated by periods. This format accurately references the "name" attribute of the "kubernetes_namespace" resource with the instance name "example".

Overall explanation

Correct Answer: `kubernetes_namespace.example.name`

Explanation:

In Terraform, to reference an attribute of a resource, you use the resource type and name in the format `<resource_type>.<resource_name>.<attribute>`. In this case:

- The resource type is `kubernetes_namespace`.
- The resource name is `example`.
- The attribute is `name`.

So, to reference the `name` attribute, you would write `kubernetes_namespace.example.name`.

- `resource.kubernetes_namespace.example.name` is incorrect because `resource` is not part of the syntax.
- `kubernetes_namespace.test.name` is incorrect because `test` is not the resource name here; the name defined in the resource block is `example`.
- `data.kubernetes_namespace.name` is incorrect because this syntax would be used for data sources, not resources.

- "None of the above" is incorrect, as `kubernetes_namespace.example.name` is the correct answer.

Resources

References to Resource Attributes

Question 23 Incorrect

Which of the following is the safest way to inject sensitive values into a Terraform Cloud workspace?

Your answer is incorrect

Set the variable value on the command line with the -var flag

Explanation

Setting the variable value on the command line with the -var flag is not the safest way to inject sensitive values into a Terraform Cloud workspace. Passing sensitive information through command-line arguments can expose it to potential security risks, such as being stored in shell history or visible to other users on the system. Using the Terraform Cloud UI with the "Sensitive" checkbox is a more secure approach for handling sensitive data.

Write the value to a file and specify the file with the -var-file flag

Explanation

Writing the sensitive value to a file and specifying the file with the -var-file flag is not the safest way to inject sensitive values into a Terraform Cloud workspace. Storing sensitive information in a file can expose it to unauthorized access, and using the -var-file flag may not provide the necessary level of security for protecting sensitive data.

Correct answer

Set a value for the variable in the UI and check the "Sensitive" check box

Explanation

Setting a value for the variable in the Terraform Cloud UI and checking the "Sensitive" checkbox is the safest way to inject sensitive values into a Terraform Cloud workspace. By marking the variable as sensitive, Terraform encrypts the value and ensures that it is not displayed in logs or outputs, providing an extra layer of security for sensitive information.

Edit the state file directly just before running terraform apply

Explanation

Editing the state file directly just before running terraform apply is not a safe practice for injecting sensitive values into a Terraform Cloud workspace. Modifying the state file manually can lead to inconsistencies and potential data corruption, and it does not offer the same level of security and protection as using the built-in sensitive variable feature in the Terraform Cloud UI.

Overall explanation

Correct Answer: Set a value for the variable in the UI and check the "Sensitive" check box

Explanation:

In Terraform Cloud, the safest way to handle sensitive values is to define them directly within the workspace UI and mark them as **Sensitive**. This option prevents the sensitive data from being logged in the Terraform Cloud run output, stored in version control, or accidentally exposed to other users.

Explanation of Other Options:

- **Writing the value to a file and specifying it with the `-var-file` flag**: This approach is less secure because the file could be accidentally committed to version control or accessed by other users with file access permissions.
- **Editing the state file directly**: Manually editing the state file is highly discouraged as it increases the risk of corruption, inconsistency, and exposure of sensitive data.
- **Setting the variable on the command line with the `-var` flag**: Command-line variables are less secure since they may appear in command history or be visible in process lists, exposing sensitive data to unauthorized users.

Using the "Sensitive" checkbox in the Terraform Cloud UI is the most secure and recommended approach, as it limits exposure and handles sensitive data appropriately within Terraform Cloud.

Resources

Terraform workspace variables

Question 24 Incorrect

Which of the following can you do with terraform plan? **(Choose two.)**

Your selection is correct

View the execution plan and check if the changes match your expectations

Explanation

Viewing the execution plan and checking if the changes match your expectations is a key feature of terraform plan. This command shows you the proposed changes to your infrastructure based on the configuration files, allowing you to verify the expected outcome before applying the changes.

Your selection is incorrect

Execute a plan in a different workspace

Explanation

Executing a plan in a different workspace is not a feature of terraform plan. Workspaces in Terraform are used to manage different environments or configurations, but executing a plan in a specific workspace is done with the terraform workspace select command, not with terraform plan.

Schedule Terraform to run at a planned time in the future

Explanation

Scheduling Terraform to run at a planned time in the future is not a functionality provided by terraform plan. Terraform does not have built-in scheduling capabilities for running commands at specific times; it is typically triggered manually or through automation tools like CI/CD pipelines.

Correct selection

Save a generated execution plan to apply later

Explanation

Saving a generated execution plan with terraform plan allows you to review and apply the changes later. This is useful for reviewing the proposed changes before actually applying them to your infrastructure.

Overall explanation

Correct Answer: Save a generated execution plan to apply later, View the execution plan and check if the changes match your expectations

Explanation:

- The `terraform plan` command allows you to preview the changes that Terraform will apply to your infrastructure without actually making any modifications. It generates an execution plan, which can be saved for later use by redirecting the output to a file. This is useful to apply the plan at a later time using the `terraform apply` command.
- You can also use `terraform plan` to view the proposed changes, ensuring they align with your expectations. This is a critical step for reviewing and validating what Terraform will do before it makes any changes to your resources.
- **Execute a plan in a different workspace** is incorrect because `terraform plan` is executed in the current workspace by default. Changing the workspace is done separately using `terraform workspace select`.
- **Schedule Terraform to run at a planned time in the future** is incorrect because Terraform itself does not have a built-in scheduling function. Scheduling tasks would typically be managed outside of Terraform using other tools like cron jobs, Jenkins, or CI/CD pipelines.

Resources

[Create a Terraform plan]

Question 25Incorrect

Which of the following is **not** an advantage of using infrastructure as code operations?

Correct answer

Public cloud console configuration workflows

Explanation

Public cloud console configuration workflows are not considered an advantage of using infrastructure as code operations. Infrastructure as code promotes the use of declarative configuration files to define and manage infrastructure, rather than relying on manual configurations through a public cloud console.

Your answer is incorrect

Troubleshoot via a Linux diff command

Explanation

Troubleshooting via a Linux diff command is not directly related to the advantages of using infrastructure as code operations. While infrastructure as code can help in maintaining consistency and reproducibility, troubleshooting specific issues with a diff command is more related to comparing file contents in Linux environments.

API driven workflows

Explanation

API-driven workflows are a key advantage of using infrastructure as code operations. APIs allow for automation, integration, and orchestration of infrastructure provisioning and management tasks, enabling seamless interaction with cloud services and resources through code.

Self-service infrastructure deployment

Explanation

Self-service infrastructure deployment is indeed an advantage of using infrastructure as code operations. It allows users to provision and manage infrastructure resources independently without manual intervention, leading to faster and more efficient deployments.

Modify a count parameter to scale resources

Explanation

Modifying a count parameter to scale resources is an advantage of using infrastructure as code operations. By adjusting parameters in the code, users can easily scale resources up or down based on demand without manual intervention, ensuring scalability and flexibility in managing infrastructure.

Overall explanation

Correct Answer: Public cloud console configuration workflows

Explanation:

Infrastructure as Code (IaC) provides significant advantages in automation, consistency, and repeatability when managing infrastructure. Key benefits include:

- **Self-service infrastructure deployment:** Allows teams to deploy infrastructure independently without requiring manual approvals, thus increasing efficiency.
- **Troubleshoot via a Linux diff command:** Enables tracking of configuration changes, making it easier to identify differences between versions.
- **Modify a count parameter to scale resources:** Facilitates dynamic scaling by adjusting resource counts in code, which simplifies resource management.
- **API-driven workflows:** Allows seamless integration with other tools and systems, further enhancing automation and consistency.

Public cloud console configuration workflows do not align with the principles of IaC. These workflows involve manual steps through a cloud provider's web interface, which lacks the version control, automation, and repeatability that IaC provides. Instead, IaC emphasizes managing infrastructure through code, reducing the need for console-based configuration.

Question 26 Correct

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

Mandatory

Explanation

Changing the Terraform backend from the default "local" backend to a different one after the first terraform apply is not mandatory. While it is recommended to set up the backend before applying any changes, it is still possible to switch to a different backend after the initial apply without any issues.

Impossible

Explanation

It is not impossible to change the Terraform backend from the default "local" backend to a different one after the first terraform apply. Terraform provides flexibility in managing backend configurations, allowing users to make changes as needed throughout the infrastructure provisioning process.

Discouraged

Explanation

While it may be discouraged to frequently change the Terraform backend configuration, switching from the default "local" backend to a different one after the first terraform apply is not necessarily discouraged. It is important to consider the implications of changing the backend, such as potential state file conflicts, but it is still a valid option in certain scenarios.

Your answer is correct
Optional
Explanation
This choice is correct because changing the Terraform backend from the default "local" backend to a different one after the first terraform apply is optional. While it is best practice to define the backend configuration before applying any changes, Terraform allows you to switch backends at any point in your workflow.
Overall explanation
Correct Answer: Optional

Explanation:

Changing the Terraform backend from the default "local" backend to a different one (such as S3, Terraform Cloud, or others) after running your first `terraform apply` is optional. However, if you choose to do this, you must migrate the existing state file to the new backend. Terraform supports this process using `terraform init` to reconfigure the backend and migrate the state.

- **Mandatory** is incorrect because switching backends is not mandatory. It depends on your organizational or infrastructure requirements.
- **Impossible** is incorrect because it is possible to change the backend, but it requires proper migration steps.
- **Discouraged** is incorrect because changing the backend is supported, as long as the migration is handled properly.

Question 27Incorrect
How would you be able to reference an attribute from the vsphere_datacenter data source for use with the datacenter_id argument within the vsphere_folder resource in the following configuration?

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

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

data.vsphere_datacenter.dc
Explanation
The syntax data.vsphere_datacenter.dc is incorrect because it does not follow the correct format to reference an attribute from the vsphere_datacenter data source. It should directly access the attribute using data.dc.id.
Correct answer
data.vsphere_datacenter.dc.id
Explanation
The correct syntax to reference an attribute from the vsphere_datacenter data source is data.vsphere_datacenter.dc.id. This format correctly accesses the id attribute from the vsphere_datacenter data source for use with the datacenter_id argument within the vsphere_folder resource.
data.dc.id
Explanation
The correct syntax to reference an attribute from the vsphere_datacenter data source is data.dc.id. This syntax directly accesses the id attribute of the vsphere_datacenter data source without specifying the full path.
Your answer is incorrect
vsphere_datacenter.dc.id
Explanation
The syntax vsphere_datacenter.dc.id is incorrect because it does not include the data prefix to reference the vsphere_datacenter data source. The correct format should start with data.vsphere_datacenter.dc.id.
Overall explanation
The correct way to reference an attribute from the `vsphere_datacenter` data source for the `datacenter_id` argument within the `vsphere_folder` resource is:

data.vsphere_datacenter.dc.id

Explanation:

- Terraform uses the format `data.<data_source_name>.<name>.<attribute>` to reference attributes from a data source.
- Here, `data` indicates that you are referencing a data source. `vsphere_datacenter` is the name of the data source, `dc` is the name assigned to it in your configuration, and `id` is the attribute you want to reference.

Other options are incorrect because:

- `data.dc.id` is invalid as it omits the data source type (`vsphere_datacenter`).
- `data.vsphere_datacenter.dc` does not specify the attribute (`id`) you want to reference.
- `vsphere_datacenter.dc.id` omits the required `data` keyword, making it an invalid reference format.

Question 28 Correct

Which are forbidden actions when the Terraform state file is locked? **(Choose three.)**

terraform fmt

Explanation

Running `terraform fmt` is not a forbidden action when the Terraform state file is locked. This command is used to format Terraform configuration files for readability and consistency and does not directly modify the state file.

Your selection is correct

terraform destroy

Explanation

When the Terraform state file is locked, running `terraform destroy` is a forbidden action because it can modify or delete resources that are currently being managed by Terraform, potentially causing conflicts and inconsistencies in the state file.

Your selection is correct

terraform plan

Explanation

When the Terraform state file is locked, running `terraform plan` is a forbidden action because it generates an execution plan showing what actions Terraform will take to change the infrastructure. Applying this plan could cause conflicts with the locked state file.

terraform state list

Explanation

Running `terraform state list` is not a forbidden action when the Terraform state file is locked. This command is used to list resources in the Terraform state and does not modify the state file itself.

Your selection is correct

terraform apply

Explanation

When the Terraform state file is locked, running `terraform apply` is a forbidden action because it can make changes to the infrastructure based on the Terraform configuration, potentially causing conflicts with the locked state file.

terraform validate

Explanation

Running `terraform validate` is not a forbidden action when the Terraform state file is locked. This command is used to check the syntax and configuration of Terraform files for errors and does not directly modify the state file.

Overall explanation

Correct Answer: terraform destroy, terraform apply, terraform plan

Explanation:

When the Terraform state file is locked (which happens during operations like `terraform apply` or `terraform destroy`), it means that Terraform is actively working on modifying the infrastructure or managing the state. The lock prevents concurrent operations from happening and ensures that no conflicting changes are made to the infrastructure or the state file while Terraform is working.

- **terraform destroy**: This operation would modify the infrastructure and can only be run when no other Terraform operations are modifying the state.
- **terraform apply**: This operation is typically blocked while the state file is locked to prevent concurrent changes to the infrastructure.
- **terraform plan**: Since `terraform plan` can potentially alter the infrastructure state by suggesting changes, it is also locked out during the state file lock to avoid running conflicting plans simultaneously.

terraform fmt, **terraform state list**, and **terraform validate** are **not** operations that modify the infrastructure directly, and they do not require a state lock to run. These operations are primarily focused on formatting, listing resources, and validating configuration syntax, so they are allowed even when the state file is locked.

Resources

Terraform state Locking

Question 29Incorrect

Which parameters does terraform import require? **(Choose two.)**

Correct selection

Resource address

Path

Your selection is correct

Resource ID

Your selection is incorrect

Provider

Overall explanation

Correct Answer: Resource ID and Resource address

Explanation:

The `terraform import` command is used to bring an existing infrastructure resource into Terraform's state file so Terraform can manage it. To successfully perform this import, Terraform requires two key parameters:

- **Resource address**: This is the Terraform configuration address that identifies the specific resource in the configuration. For example, in `aws_instance.my_instance`, `aws_instance` is the resource type, and `my_instance` is the specific resource name in the Terraform configuration.
- **Resource ID**: This is the unique identifier that the cloud provider or API assigns to the resource. For example, for an AWS EC2 instance, the resource ID might look like `i-1234567890abcdef0`. Terraform uses this ID to map the resource in the cloud to the corresponding resource in the state file.

Incorrect Options:

- **Path**: Terraform import does not use a file path as an argument; it directly uses the resource address and ID.
- **Provider**: Terraform determines the provider based on the resource type in the resource address (like `aws_instance` for AWS). There is no need to specify the provider explicitly when using `terraform import`.

Resources

Command: import

Question 30Correct

When do you need to explicitly execute **terraform refresh**?

Before every terraform import

Explanation

While `terraform import` does require the resource to be imported into the Terraform state, it does not necessarily require an explicit `terraform refresh` before the import. The `terraform import` command can handle the state refresh as part of the import process.

Your answer is correct

None of the above

Explanation

The correct choice is this because `terraform refresh` is not explicitly needed before every `terraform plan`, `terraform apply`, or `terraform import`. The `terraform refresh` command is used to update the state file with the real-world infrastructure, but it is not a mandatory step before every operation.

Before every terraform plan

Explanation

Running `terraform refresh` before every `terraform plan` is not necessary because `terraform plan` already automatically refreshes the state of the infrastructure before generating the execution plan. This ensures that the plan is based on the latest state of the infrastructure.

Before every terraform apply

Explanation

Executing `terraform refresh` before every `terraform apply` is not required as `terraform apply` automatically refreshes the state of the infrastructure before making any changes. This ensures that the changes are applied based on the most up-to-date state of the infrastructure.

Overall explanation

Correct Answer: None of the above

Explanation:

Executing `terraform refresh` explicitly is generally not required before `terraform plan`, `terraform apply`, or `terraform import`. Terraform automatically checks and updates the state file during these operations to ensure it reflects the real-world infrastructure accurately.

However, `terraform refresh` can be useful in specific scenarios where you want to ensure the state file is fully updated with the latest real-world resource data without planning or applying any changes. This might be useful in troubleshooting situations or when verifying the current state of resources.

Resources

Command: refresh

Question 31Incorrect

What does this code do?

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

Your answer is incorrect

Requires any version of the AWS provider >= 3.0

Explanation

This choice is incorrect because the code snippet specifically includes the constraint "~> 3.0", which means it requires a version of the AWS provider that is greater than or equal to 3.0 but less than 4.0. It does not allow any version greater than 4.0.

Requires any version of the AWS provider > 3.0

Explanation

This choice is incorrect because the code snippet uses the "~>" operator, which constrains the AWS provider version to be greater than or equal to 3.0 but less than 4.0. It does not allow versions that are strictly greater than 3.0.

Correct answer

Requires any version of the AWS provider >= 3.0 and < 4.0

Explanation

This code snippet specifies that the Terraform configuration requires the AWS provider version to be greater than or equal to 3.0 but less than 4.0. The '~>' operator indicates a constraint that allows any version in the 3.x range, ensuring compatibility with minor updates while preventing major version upgrades.

Requires any version of the AWS provider after the 3.0 major release, like 4.1

Explanation

This choice is incorrect because the code snippet does not specify that it requires any version of the AWS provider after the 3.0 major release, like 4.1. The constraint "~> 3.0" indicates a range within the 3.x versions, not including versions from the 4.x series.

Overall explanation

Correct Answer: Requires any version of the AWS provider >= 3.0 and < 4.0

Explanation:

In this Terraform code, the `"~> 3.0"` version constraint is specified for the AWS provider. In Terraform, the `~>` operator is a "pessimistic" constraint that allows for updates within the specified major version (in this case, 3). Specifically:

- The constraint `~> 3.0` means that any version `>= 3.0` and `< 4.0` is acceptable.
- This allows patch updates within version 3 (like 3.1, 3.5, 3.10, etc.) but prevents major version updates to 4.x or above.

Incorrect Options:

- **`>= 3.0`**: This would allow any version 3.0 or higher, including versions in the 4.x range, which is not what `"~> 3.0"` specifies.
- **After the 3.0 major release, like 4.1**: This is incorrect because `"~> 3.0"` explicitly restricts to version 3.x only, preventing updates to 4.x.
- **`> 3.0`**: This would allow versions strictly greater than 3.0, which could include major version updates as well.

Resources
Version Constraints

Question 32 Correct
If a Terraform creation-time provisioner fails, what will occur by default?

The resource will not be affected, but the provisioner will need to be applied again
Explanation
If a Terraform creation-time provisioner fails, the resource will not be affected, but the provisioner will need to be applied again to ensure that the resource is properly provisioned. The failure of the provisioner does not automatically impact the resource itself.

Your answer is correct
The resource will be marked as "tainted"
Explanation
If a Terraform creation-time provisioner fails, the resource will be marked as "tainted." This means that Terraform will consider the resource to be potentially in an inconsistent state and will plan to recreate it during the next apply operation to ensure its proper provisioning.

Nothing, provisioners will not show errors in the command line
Explanation
If a Terraform creation-time provisioner fails, errors related to the provisioner will be displayed in the command line output. Terraform will not ignore errors from provisioners, and they will be shown to the user for troubleshooting and resolution.

The resource will be destroyed
Explanation
If a Terraform creation-time provisioner fails, the resource will not be automatically destroyed. The failure of the provisioner does not trigger the destruction of the resource, but it may need to be re-applied to ensure proper provisioning.

Overall explanation
Correct Answer: The resource will be marked as "tainted"

Explanation:

If a creation-time provisioner fails during a `terraform apply`, Terraform will mark the resource as "tainted." This means that while the resource was created, it is considered in a faulty or incomplete state because the provisioner did not execute successfully.

When a resource is marked as tainted, Terraform will attempt to destroy and recreate the resource during the next apply, ensuring that the provisioner has a chance to run again on a fresh instance.

Other options:

- **The resource will not be affected, but the provisioner will need to be applied again** is incorrect because the resource is indeed affected by being marked tainted.
- **The resource will be destroyed** is incorrect because Terraform does not automatically destroy the resource on a provisioner failure. It only marks it as tainted to handle it in the next `terraform apply`.

- **Nothing, provisioners will not show errors in the command line** is incorrect. Provisioner errors are reported in the command line, and Terraform takes action by marking the resource as tainted.

Resources

[Declaring Provisioners]

Question 33Correct

Which of the following is true about `terraform apply`? **(Choose two.)**

By default, it does not refresh your state file to reflect current infrastructure configuration

Explanation

By default, terraform apply does refresh your state file to reflect the current infrastructure configuration after applying changes. This helps to keep the state file up to date with the actual state of the infrastructure.

You cannot target specific resources for the operation

Explanation

You can target specific resources for the operation using the -target flag in terraform apply. This allows you to apply changes only to the specified resources, rather than applying changes to all resources defined in the configuration files.

Your selection is correct

Depending on provider specification, Terraform may need to destroy and recreate your infrastructure resources

Explanation

Depending on the provider specification and the changes made in the Terraform configuration, terraform apply may need to destroy and recreate your infrastructure resources. This is to ensure that the changes are applied correctly and reflect the desired state.

Your selection is correct

It only operates on infrastructure defined in the current working directory or workspace

Explanation

Terraform apply only operates on the infrastructure defined in the current working directory or workspace. It will apply changes to the resources specified in the Terraform configuration files within that directory.

You must pass the output of a terraform plan command to it

Explanation

It is not necessary to pass the output of a terraform plan command to terraform apply. While it is recommended to run terraform plan before applying changes to preview the modifications, it is not a mandatory step for applying changes.

Overall explanation

Correct Answers:

- **It only operates on infrastructure defined in the current working directory or workspace**
- **Depending on provider specification, Terraform may need to destroy and recreate your infrastructure resources**

Explanation:

1. **Operates on infrastructure in the current directory/workspace**: `terraform apply` runs based on the configuration files in the current working directory or workspace. Terraform's state is also managed at the workspace level, allowing separate environments to be managed independently.
2. **May need to destroy and recreate resources**: In some cases, if certain resource properties are modified in a way that requires replacement, Terraform will need to destroy and recreate those resources to apply the changes. This behavior depends on provider requirements for specific resources.

Other Options Explained:

- **Passing output of `terraform plan`**: This is not required; you can run `terraform apply` directly to both plan and apply changes in a single command. However, you can pass a saved plan file to `terraform apply` if desired, but it is not a requirement.
- **Refreshes state by default**: By default, `terraform apply` will refresh the state to ensure it reflects the latest status of resources, unless you specify `-refresh=false`.
- **Targeting specific resources**: You can target specific resources with the `-target` flag if you only want `terraform apply` to operate on certain parts of the infrastructure.

Resources

[Provisioning Infrastructure with Terraform]

Question 34 Correct

Terraform Cloud is available only as a paid offering from HashiCorp.

True

Explanation

Terraform Cloud is available in both free and paid versions from HashiCorp. The free version offers essential features for small teams and individual users, while the paid version provides additional functionality and support for larger organizations with more complex infrastructure needs.

Your answer is correct

False

Explanation

This choice is correct because Terraform Cloud is available in both free and paid versions from HashiCorp. The false statement accurately reflects the availability of Terraform Cloud as a paid offering only.

Overall explanation

Correct Answer: False

Explanation:

Terraform Cloud is available in both free and paid tiers. The free tier provides essential functionality such as remote state management, remote operations, and limited team collaboration features. Paid offerings, on the other hand, provide additional features like advanced team management, governance, security controls, and support for larger teams and enterprises. This makes it accessible for smaller teams and individual users while offering extended capabilities for organizations needing more advanced features.

Resources

Terraform cloud

Question 35 Incorrect

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

TF_LOG_FILE

Explanation

TF_LOG_FILE is not the correct environment variable to configure for making Terraform's logging more verbose. This variable is used to specify the file where Terraform logs should be written, but it does not control the verbosity of the logs.

TP_LOG_PATH

Explanation

TP_LOG_PATH is not a valid environment variable for configuring Terraform's logging verbosity. This variable does not exist in Terraform's documentation and is not recognized by the tool for controlling logging levels.

Correct answer

TF_LOG

Explanation

TF_LOG is the correct environment variable to configure for making Terraform's logging more verbose. By setting TF_LOG to a value of "TRACE", Terraform will output detailed logs that can help troubleshoot any issues and provide more insight into the deployment process.

Your answer is incorrect

TF_LOG_LEVEL

Explanation

TF_LOG_LEVEL is not the correct environment variable to configure for making Terraform's logging more verbose. This variable is used to set the log level for Terraform, but it does not directly control the verbosity of the logs.

Overall explanation

Correct Answer: TF_LOG

Explanation:

To enable debugging and increase verbosity in Terraform logs, you can set the `TF_LOG` environment variable. Setting `TF_LOG` to a specific level (such as `TRACE`, `DEBUG`, `INFO`, `WARN`, or `ERROR`) controls the verbosity of Terraform's output, with `TRACE` providing the most detailed information. This can help identify issues and track the execution process more closely.

- **TF_LOG_LEVEL** is not a valid environment variable in Terraform for debugging purposes.
- **TF_LOG_FILE** does not control verbosity. Instead, it is used to specify the path for logging output if you want to save logs to a file.
- **TP_LOG_PATH** is not a recognized Terraform environment variable.

Resources

Debugging Terraform

Question 36Correct

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

terraform get

Explanation

The `terraform get` command is used to download and install modules from the Terraform registry. It is not related to accessing the attributes and details of a specific resource managed by Terraform, so it is not the correct choice for this scenario.

Your answer is correct

terraform state show

Explanation

The `terraform state show` command is the correct choice for accessing all the attributes and details of a specific resource managed by Terraform. It displays detailed information about the resource, including its attributes, metadata, and dependencies, making it the appropriate command for this purpose.

terraform state list

Explanation

The `terraform state list` command is used to list all resources in the Terraform state. It does not provide detailed information about a specific resource's attributes and details, so it is not the correct choice for accessing all the details of a resource.

terraform state list

Explanation

The `terraform state list` command lists all resources in the Terraform state but does not provide detailed information about a specific resource's attributes and details. Therefore, it is not the appropriate command to use for accessing all the details of a resource managed by Terraform.

Overall explanation

The correct command to access all of the attributes and details of a resource managed by Terraform is:

terraform state show

Explanation:

- `terraform state show` provides detailed information about a specific resource managed by Terraform, including all attributes and their values. This is useful for inspecting the resource as it exists in the state file.

The other options are incorrect because:

- `terraform state list` lists all resources managed in the current state file, but does not show the detailed attributes of a specific resource.
- `terraform get` is used to download and update modules in the configuration, not to inspect resources.
- `terraform state list` appears twice, but as mentioned, it is for listing resources, not displaying their details.

Resources

Command: state show

Question 37Incorrect

What does Terraform use providers for? **(Choose three.)**

Enforce security and compliance policies

Correct selection

Simplify API interactions

Your selection is incorrect
Group a collection of Terraform configuration files that map to a single state file
Your selection is correct
Provision resources for on-premises infrastructure services
Your selection is correct
Provision resources for public cloud infrastructure services
Overall explanation
Correct Answers:

1. **Provision resources for on-premises infrastructure services**
2. **Simplify API interactions**
3. **Provision resources for public cloud infrastructure services**

Explanation:

Terraform providers are essential for connecting Terraform to external services. They enable Terraform to interact with APIs for various infrastructure platforms and manage resources. Here's how these answers apply:

- **Provision resources for on-premises infrastructure services**: Providers can manage resources not only in cloud environments but also in on-premises infrastructure, such as VMware or OpenStack.
- **Simplify API interactions**: Providers abstract the complexities of APIs, allowing Terraform to handle interactions with different services in a standardized way, without requiring you to deal with the API details.
- **Provision resources for public cloud infrastructure services**: Providers support a wide range of public cloud providers like AWS, Azure, and Google Cloud, allowing Terraform to create, update, and delete resources in those environments.

Incorrect Options:

- **Enforce security and compliance policies**: While providers may help manage resources within environments with security policies, enforcing policies is generally handled outside of Terraform, often through third-party tools.
- **Group a collection of Terraform configuration files that map to a single state file**: This is not the purpose of providers. Grouping configuration files and managing state is handled by Terraform itself, not by providers.

Question 38 Incorrect
Which of the following is **not** valid source path for specifying a module?

source = "github.com/hashicorp/example?ref=v1.0.0"
Explanation
The source path "github.com/hashicorp/example?ref=v1.0.0" is a valid source path for specifying a module in Terraform. It points to a specific repository on GitHub and includes the reference to version 1.0.0 using the "ref" parameter.
Your answer is incorrect
source = "hashicorp/consul/aws"
Explanation
The source path "hashicorp/consul/aws" is a valid source path for specifying a module in Terraform. It refers to a module located in the official HashiCorp organization on the Terraform Registry, specifically the Consul module for AWS. This is a common way to specify modules from the Terraform Registry.
source = "./module"
Explanation
The source path "./module" is a valid source path for specifying a module in Terraform. It refers to a module located in the same directory as the configuration file where it is being used. This is a common way to specify local modules.
Correct answer
source = "./modulelversion=v1.0.0"
Explanation
The source path "./modulelversion=v1.0.0" contains a typo with "modulel" instead of "module" and the version should be specified using "version" instead of "version=". This makes the source path invalid for specifying a module in Terraform.
Overall explanation
Correct Answer: source = "./modulelversion=v1.0.0"

Explanation:

The path `"./modulelversion=v1.0.0"` is not a valid way to specify a module source because it incorrectly combines a relative file path and a version specification in a single string. In Terraform, local paths (like `"./module"`) should point to directories containing module files without adding version tags or additional parameters directly in the path.

Other Options Explained:

- **"github.com/hashicorp/example?ref=v1.0.0"**: This is a valid source for specifying a module hosted on GitHub, with the `?ref=v1.0.0` syntax to specify a version (typically a tag or branch).
- **"./module"**: This is a valid relative path pointing to a module in a local directory.
- **"hashicorp/consul/aws"**: This is a valid reference for a module from the public Terraform Module Registry, using the format `namespace/module/provider`.

Resources

Module sources

Question 39Incorrect

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

terraform apply -upgrade

Explanation

The "terraform apply -upgrade" command is used to apply the changes to your infrastructure as described in your Terraform configuration files. It does not specifically handle updating plugins to the latest versions, so it is not the correct choice for ensuring your plugins are up to date.

Correct answer

terraform init -upgrade

Explanation

The correct command to use when you want to ensure your plugins are up to date with the latest versions is "terraform init -upgrade". This command will initialize the Terraform working directory and download any missing plugins or update existing plugins to the latest versions available.

terraform refresh -upgrade

Explanation

The "terraform refresh -upgrade" command is used to update the state file with the real-world infrastructure, but it does not handle updating plugins to the latest versions. It is not the appropriate command for ensuring your plugins are up to date with the latest versions.

Your answer is incorrect

terraform providers -upgrade

Explanation

The "terraform providers -upgrade" command is used to upgrade the providers used in your Terraform configuration files to the latest versions available. While this command is related to updating providers, it does not cover updating all plugins in general. The correct command for updating all plugins is "terraform init -upgrade".

Overall explanation

Correct Answer: terraform init -upgrade

Explanation:

The command `terraform init -upgrade` is used to ensure that all the provider plugins and modules in the Terraform configuration are up-to-date with the latest available versions. It checks for the latest versions and upgrades them if necessary. This is especially useful when you want to make sure your plugins are using the most recent updates and bug fixes.

- **terraform apply -upgrade** is incorrect because the `apply` command is used to apply changes to the infrastructure, not to upgrade plugins.
- **terraform refresh -upgrade** is incorrect because `terraform refresh` is used to update the state file with the latest data from the infrastructure, not for upgrading plugins.

- **terraform providers -upgrade** is not a valid command in Terraform; you upgrade providers through `terraform init -upgrade`.

Question 40Incorrect

Which of the following statements about local modules is incorrect?

Local modules are sourced from a directory on disk

Explanation

Local modules are indeed sourced from a directory on disk. This means that you can create and manage your own modules within your project directory, making it easier to organize and reuse your infrastructure code.

Local modules are not cached by terraform init command

Explanation

Local modules are actually cached by the terraform init command. When you use a local module, Terraform will copy the module directory into a hidden subdirectory within the .terraform directory in your working directory. This allows Terraform to track changes to the module and manage its dependencies.

Correct answer

Local modules support versions

Explanation

This statement is incorrect. Local modules do not support versions like modules sourced from a remote repository. When using local modules, you are responsible for managing the versioning and updates of the module code yourself.

Your answer is incorrect

None of the above (all statements above are correct)

Explanation

This statement is incorrect because not all statements above are correct. Choice C is incorrect as local modules do not support versions, unlike modules sourced from a remote repository.

All of the above (all statements above are incorrect)

Explanation

This statement is incorrect because not all statements above are incorrect. Choices A and B are valid statements about local modules, while choice C is the incorrect statement.

Overall explanation

Correct Answer: Local modules support versions

Explanation:

The incorrect statement here is that **local modules support versions**. Unlike remote modules that can specify versions, local modules (those stored on the local filesystem) do not support versioning in Terraform. Local modules are simply sourced from a directory path on disk, and Terraform will always use the module code exactly as it exists in that directory at the time of execution.

Other Options Explained:

- **Local modules are not cached by `terraform init`**: This is correct. The `terraform init` command does not cache local modules; it directly references the files in the specified directory.
- **Local modules are sourced from a directory on disk**: This is also correct. Local modules are accessed via a direct path on the local filesystem.
- **All of the above**: This is incorrect because only the versioning statement is incorrect, not all statements.
- **None of the above**: This is incorrect because one statement (about versioning) is incorrect, so "None of the above" would not be accurate here.

Resources

Module Blocks

Question 41Incorrect

How does Terraform determine dependencies between resources?

Correct answer

Terraform automatically builds a resource graph based on resources, provisioners, special meta-parameters, and the state file, if present.

Explanation

Terraform automatically builds a resource graph by analyzing the relationships between resources, provisioners, special meta-parameters, and the state file. This allows Terraform to determine the order in which resources should be created or updated based on their dependencies.

Terraform requires resource dependencies to be defined as modules and sourced in order

Explanation

While modules can help organize and reuse Terraform configurations, they are not required to define dependencies between resources. Terraform's dependency resolution mechanism can handle dependencies between resources within the same configuration file without the need for them to be defined as separate modules.

Terraform requires all dependencies between resources to be specified using the depends_on parameter

Explanation

While the depends_on parameter can be used to explicitly define dependencies between resources in Terraform, it is not the only way that Terraform determines dependencies. Terraform can also infer dependencies based on resource configurations and relationships without the need for explicit dependencies to be specified.

Your answer is incorrect

Terraform requires resources in a configuration to be listed in the order they will be created to determine dependencies

Explanation

Terraform does not require resources to be listed in a specific order in the configuration file to determine dependencies. Terraform's dependency resolution mechanism allows it to analyze the relationships between resources and determine the correct order for resource creation or updates, regardless of their order in the configuration file.

Overall explanation

Correct Answer: Terraform automatically builds a resource graph based on resources, provisioners, special meta-parameters, and the state file, if present.

Explanation:

Terraform automatically manages dependencies between resources by constructing a **resource dependency graph**. This graph is built using various elements in the configuration, including:

- **Implicit dependencies** created when resources reference other resources' attributes. For example, if a virtual machine resource uses a security group's ID, Terraform recognizes that the security group must be created before the VM.
- **Explicit dependencies** using the `depends_on` meta-parameter, which is optional but can be specified if you want to ensure a specific order between resources that may not have a direct reference in the configuration.
- **Provisioners** that may need resources to exist before they can run (like a `remote-exec` provisioner that requires a VM to be available).

Terraform does not require resources to be listed in a specific order or as modules for dependencies to work, nor does it require all dependencies to be explicitly defined with `depends_on`. The automatic resource graph handles most dependency detection, allowing users to focus on defining resources without manually managing the order of creation.

Incorrect Options:

- **Depends_on**: While it allows specifying explicit dependencies, it's not required for all dependencies, as Terraform automatically detects most dependencies based on references.
- **Order of listing resources**: Terraform does not depend on the order in which resources are written in the configuration, as it resolves dependencies through the graph.
- **Modules**: Modules are a way to group resources, but dependencies between resources do not need to be defined as modules.

Resources

[Create resource dependencies]

Question 42 Correct

What does `terraform refresh` modify?

Your answer is correct

Your state file

Explanation
Terraform refresh modifies your state file. When you run terraform refresh, it updates the state file with the current state of your infrastructure without making any changes to the actual resources.
Your Terraform configuration
Explanation
Terraform refresh does not modify your Terraform configuration. The configuration defines the desired state of your infrastructure, while refresh is used to update the state file to reflect the actual state of the resources.
Your Terraform plan
Explanation
Terraform refresh does not modify your Terraform plan. The plan is a preview of the changes Terraform will make to your infrastructure, while refresh is focused on updating the state file with the current state of the resources.
Your cloud infrastructure
Explanation
Terraform refresh does not modify your cloud infrastructure. It is a command used to reconcile the state Terraform knows about (via the state file) with the real-world infrastructure.
Overall explanation
Correct Answer: Your state file

Explanation:

The `terraform refresh` command is used to update the state file with the latest information from the actual infrastructure. It does not make any changes to the infrastructure itself, nor does it modify the configuration files or a Terraform plan. Instead, it ensures that the state file accurately reflects the current status of all managed resources by detecting any drift or changes that occurred outside of Terraform.

Incorrect Options:

- **Your cloud infrastructure**: `terraform refresh` only updates the state file and does not modify or recreate any infrastructure.
- **Your Terraform plan**: It does not create or modify a plan; it just syncs the state file with the actual resources.
- **Your Terraform configuration**: `terraform refresh` does not affect configuration files—it only updates the state.

Question 43 Correct
In Terraform HCL, an object type of **object({ name=string, age=number })** would match this value:

```
{

  name = "John"

  age = fifty two

 }
```
Explanation
Choice A is incorrect because the value for the 'age' attribute is not a valid number. In Terraform HCL, the age attribute should be assigned a numerical value, such as an integer or float, not a string like "fifty two". This does not match the object type specified in the question, which expects a number for the age attribute.
Your answer is correct
```
{

  name = "John"

  age = 52

 }
```
Explanation

Choice B is correct because the value provided for the 'age' attribute is a valid number (52), which matches the object type specified in the question. In Terraform HCL, when defining an object type with specific attribute types, the values assigned to those attributes must match the specified types.

Overall explanation

Correct Answer:

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

Explanation:

In Terraform HCL, an `object` type such as `object({ name = string, age = number })` requires that each specified attribute in the object matches the defined types. Here, `name` must be a string, and `age` must be a number.

In this case:

- The example `{ name = "John", age = 52 }` matches because `name` is a string and `age` is a number, as expected by the type definition.
- The example `{ name = "John", age = fifty two }` does not match because `"fifty two"` is a string, not a number, and therefore does not satisfy the type requirement for `age`.

This strict type enforcement is part of Terraform's type system to ensure values are of the correct format for the given configuration.

Question 44Incorrect

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

Nothing ג€" modules stored on GitHub always default to version 1.0.0

Explanation

Modules stored on GitHub do not automatically default to version 1.0.0. It is important to explicitly specify the version or tag when referencing a module stored in Git to ensure that the correct version is used.

Modules stored on GitHub do not support versioning

Explanation

Modules stored on GitHub do support versioning through tags and branches. It is possible to specify a specific version or tag when referencing a module stored in Git to ensure that the desired version is used in your Terraform configuration.

Correct answer

Append ?ref=v1. 0. 0 argument to the source path

Explanation

Appending ?ref=v1.0.0 argument to the source path when referencing a module stored in Git allows you to specify the exact tag or branch to use. This ensures that the module is pulled from the correct version in the Git repository.

Your answer is incorrect

Add version = "1.0.0" parameter to module block

Explanation

Adding version = "1.0.0" parameter to the module block is not the best way to specify a tag when referencing a module stored in Git. This parameter is used to specify the version of the module itself, not the version of the Git repository it is pulled from.

Overall explanation

Correct Answer: Append `?ref=v1.0.0` argument to the source path

Explanation:

When you are referencing a module stored in Git, you can specify a particular tag (like `v1.0.0`) by appending `?ref=v1.0.0` to the source URL. This allows Terraform to fetch that specific version of the module from the Git repository.

- **Adding version = "1.0.0"** within the module block is incorrect because this is used for module versions in Terraform Registry, not for Git-based modules.

- **Modules stored on GitHub always default to version 1.0.0** is not true because Git-based modules do not have a default version unless explicitly specified.
- **Modules stored on GitHub do not support versioning** is incorrect because Git repositories do support versioning through tags or commits, and Terraform can reference specific versions using the `ref` argument.

Resources

[Selecting a Revision]

Question 45 Correct

Which of the following is true about Terraform's implementation of infrastructure as code? (Choose two.)

You cannot reuse infrastructure configuration

Explanation

This choice is incorrect. One of the key advantages of using Terraform is the ability to reuse infrastructure configuration. Terraform allows you to define infrastructure as code, which can be versioned, shared, and reused across different environments and projects.

Your selection is correct

You can version your infrastructure configuration

Explanation

This choice is correct. Terraform allows you to version your infrastructure configuration using version control systems like Git. This enables you to track changes, collaborate with team members, and revert to previous configurations if needed, providing better control and visibility over your infrastructure changes.

It is only compatible with AWS infrastructure management

Explanation

This choice is incorrect. Terraform is not limited to managing AWS infrastructure only. It supports multiple cloud providers such as Azure, Google Cloud Platform, and more, making it a versatile tool for infrastructure management across different platforms.

It requires manual configuration of infrastructure resources

Explanation

This choice is incorrect. Terraform is designed to automate the configuration and provisioning of infrastructure resources. It eliminates the need for manual configuration by allowing you to define infrastructure as code using declarative configuration files.

Your selection is correct

It allows you to automate infrastructure provisioning

Explanation

This choice is correct. One of the key features of Terraform is its ability to automate infrastructure provisioning. By defining infrastructure as code, Terraform enables you to automate the creation, modification, and deletion of infrastructure resources, making it easier to manage and scale your infrastructure.

Overall explanation

Correct Answers: You can version your infrastructure configuration, It allows you to automate infrastructure provisioning

Explanation:

Terraform's implementation of infrastructure as code (IaC) provides several key benefits:

- **You can version your infrastructure configuration:** Terraform configurations are typically written in code files that can be version-controlled using systems like Git. This allows teams to track changes over time, roll back to previous configurations, and ensure that infrastructure changes are documented and auditable.
- **It allows you to automate infrastructure provisioning:** One of the main benefits of Terraform is that it enables automation of infrastructure provisioning. By defining infrastructure in code, you can execute Terraform commands to provision, update, or destroy infrastructure resources automatically, reducing the need for manual intervention.

Other Options Explained:

- **It is only compatible with AWS infrastructure management:** This is incorrect. Terraform is compatible with multiple cloud providers, including AWS, Azure, Google Cloud, and on-premises solutions, making it highly versatile for managing infrastructure across various platforms.
- **You cannot reuse infrastructure configuration:** This is incorrect. Terraform encourages reusable configurations through modules, which allow you to define, share, and reuse infrastructure components across different environments.

- **It requires manual configuration of infrastructure resources:** This is incorrect. Terraform eliminates the need for manual configuration by allowing resources to be defined and managed in code, enabling automation and reducing manual setup efforts.

Question 46 Correct

What command can you run to generate **DOT (Document Template)** formatted data to visualize Terraform dependencies?

terraform show

Explanation

The `terraform show` command is used to display the current state or a saved plan. It does not generate DOT formatted data to visualize Terraform dependencies.

terraform output

Explanation

The `terraform output` command is used to extract the output variables defined in the Terraform configuration. It does not generate DOT formatted data to visualize Terraform dependencies.

terraform refresh

Explanation

The `terraform refresh` command is used to update the state file by querying the infrastructure. It does not generate DOT formatted data to visualize Terraform dependencies.

Your answer is correct

terraform graph

Explanation

The `terraform graph` command is used to generate DOT formatted data that represents the Terraform configuration and its dependencies. This data can be visualized to understand the relationship between resources in the infrastructure.

Overall explanation

Correct Answer: terraform graph

Explanation:

The `terraform graph` command generates DOT-formatted data that represents the relationships between the resources in your Terraform configuration. DOT is a graph description language that can be used to create visualizations of your infrastructure's dependency graph. This is useful for visualizing how different resources depend on each other in a Terraform-managed infrastructure.

- **terraform refresh** updates the state file with real-time data about the resources but does not generate dependency graphs.
- **terraform show** outputs the current state or plan in a human-readable format, but it is not used for generating DOT graphs.
- **terraform output** displays the output values defined in the Terraform configuration, not for generating dependency graphs.

Thus, `terraform graph` is the correct command to generate dependency data in DOT format.

Resources

terraform graph command

Question 47 Correct

Terraform apply is failing with the following error. What next step should you take to determine the root cause of the problem?

Error loading state: AccessDenied: Access Denied status code: 403, request id: 288766CE5CCA24A0, host id: FOOBA

Review syslog for Terraform error messages

Explanation

Reviewing syslog for Terraform error messages may not provide specific details about the Access Denied error that is causing the Terraform apply to fail. Syslog typically captures system-level messages and may not contain the detailed information needed to troubleshoot the specific Terraform issue.

Your answer is correct

Set TF_LOG=DEBUG

Explanation

Setting TF_LOG=DEBUG will enable detailed logging for Terraform, allowing you to see more information about the actions and requests being made. This can help you identify the specific step or resource that is causing the Access Denied error and provide more insight into the root cause of the problem.

Review /var/log/terraform.log for error messages

Explanation

Reviewing /var/log/terraform.log for error messages can provide valuable information about the Terraform apply process and any errors that occurred during the execution. However, it may not specifically address the Access Denied error and may not provide enough detail to determine the root cause of the problem.

Run terraform login to reauthenticate with the provider

Explanation

Running terraform login to reauthenticate with the provider may not resolve the Access Denied error if the issue is related to permissions or policies set within the provider's environment. Reauthenticating may not address the underlying cause of the problem and could potentially result in the same error occurring again.

Overall explanation

Correct Answer: Set TF_LOG=DEBUG

Explanation:

The error message indicates that Terraform is failing to load the state due to access being denied (HTTP 403 status code), which is likely related to permissions issues or authentication problems with the provider.

Setting `TF_LOG=DEBUG` will enable detailed logging, allowing you to capture more specific information about the error, including how Terraform is interacting with the provider and why access is being denied. This will help in troubleshooting and pinpointing the exact root cause.

- **Review syslog for Terraform error messages** is incorrect because Terraform logs its actions to standard output and not syslog.
- **Run terraform login to reauthenticate with the provider** could be a potential next step, but setting `TF_LOG=DEBUG` will provide more detailed insight first before attempting a reauthentication.
- **Review /var/log/terraform.log for error messages** is incorrect because Terraform does not typically log to this file by default; errors are logged to the console or the debug log when `TF_LOG` is set.

Resources

Debugging in Terraform

Question 48 Correct

What advantage does an operations team that uses **infrastructure as cod**e have?

Your answer is correct

The ability to reuse best practice configurations and settings

Explanation

One of the key advantages of using infrastructure as code is the ability to reuse best practice configurations and settings. By defining infrastructure as code, teams can create reusable templates that embody best practices, standard configurations, and security settings. This promotes consistency, reduces errors, and speeds up the deployment process.

The ability to delete infrastructure

Explanation

While infrastructure as code allows for the creation and management of infrastructure through code, the ability to delete infrastructure is not necessarily a unique advantage of using this approach. Deleting infrastructure can be done manually or through code, but it is not a specific advantage of infrastructure as code itself.

The ability to update existing infrastructure

Explanation

Updating existing infrastructure is a common practice in infrastructure management, whether done through manual configuration changes or through infrastructure as code. While infrastructure as code does provide a structured and version-controlled way to make updates, it is not the sole advantage of using this approach.

The ability to autoscale a group of servers

Explanation
Autoscaling a group of servers is a capability that can be achieved through infrastructure as code by defining scaling policies and configurations in the code. While infrastructure as code enables automation and scalability, the ability to autoscale servers is not the only advantage of using this approach.
Overall explanation
Correct Answer: The ability to reuse best practice configurations and settings

Explanation:

Infrastructure as Code (IaC) provides significant advantages, particularly by allowing teams to define and reuse standardized configurations. This approach helps ensure consistency across environments, as well as adherence to best practices. By using IaC, operations teams can capture ideal configurations in code, then apply them consistently across multiple deployments, minimizing manual errors and increasing efficiency.

- **Deleting infrastructure** and **updating infrastructure** can be done without IaC, though IaC makes these actions more reliable and consistent.
- **Autoscaling** is typically handled by cloud provider services or orchestration tools rather than by IaC itself, although IaC can configure autoscaling settings.

Question 49Incorrect
Which command lets you experiment with Terraform's built-in functions?
terraform env
Explanation
The 'terraform env' command is used to manage Terraform environments, such as creating, deleting, or switching between different environments. It is not specifically designed for experimenting with Terraform's built-in functions.
Your answer is incorrect
terraform test
Explanation
The 'terraform test' command is not a standard Terraform command and is not used for experimenting with Terraform's built-in functions. It is important to use the correct command that is specifically designed for the task at hand.
Correct answer
terraform console
Explanation
The 'terraform console' command allows you to interactively experiment with Terraform's built-in functions. It provides a REPL (Read-Eval-Print Loop) environment where you can test and evaluate different functions before incorporating them into your Terraform configurations.
terraform validate
Explanation
The 'terraform validate' command is used to check the syntax and configuration of Terraform files to ensure they are valid and can be successfully applied. While it is an essential command for validating configurations, it is not intended for experimenting with Terraform's built-in functions.
Overall explanation
Correct Answer: `terraform console`

Explanation:

The `terraform console` command allows you to experiment with and evaluate Terraform's built-in functions directly from the command line. It opens an interactive console where you can test expressions, inspect variable values, and work with various Terraform functions in real-time. This is particularly useful for checking how functions behave with specific inputs before using them in your configuration files.

Other options:

- `terraform env` is used to manage different Terraform workspaces (environments), not for experimenting with functions.
- `terraform test` is not a standard Terraform command for experimenting with functions; Terraform does not include this command in its CLI.
- `terraform validate` checks the syntax and configuration of your Terraform files to ensure they are syntactically correct, but it doesn't allow for function experimentation.

Resources

[Terraform CLI Command: console]()

Question 50Correct

Open source Terraform can only import publicly-accessible and open-source modules.

True

Explanation

This statement is incorrect. Open source Terraform can import both publicly-accessible and private modules, not just open-source modules. Private modules can be stored in version control systems like GitHub or GitLab and imported into Terraform configurations.

Your answer is correct

False

Explanation

This statement is correct. Open source Terraform is not limited to importing only publicly-accessible and open-source modules. It can also import private modules stored in version control systems, allowing for greater flexibility and customization in Terraform configurations.

Overall explanation

Correct Answer: False

Explanation:

Open source Terraform can import both publicly accessible modules and private modules. Although the public Terraform Module Registry is available for open-source and publicly accessible modules, you can also configure Terraform to pull modules from private repositories or internal sources (like a private GitHub repository or an internally hosted Source Control Manager). This flexibility allows teams to securely reuse infrastructure code from private, restricted-access repositories.

Resources

[Terraform language modules]()

Question 51Correct

You want to tag multiple resources with a string that is a combination of a generated `random_id` and a variable.

How should you use the same value in all these resources without repeating the `random_id` and variable in each resource?

Your answer is correct

Local values

Explanation

Local values in Terraform allow you to define values that can be reused across multiple resources within the same module. By using local values, you can generate the random_id and variable combination once and reference it in all the resources that require the same value without repeating the code in each resource.

Modules

Explanation

Modules in Terraform are used to encapsulate a set of resources into a reusable and shareable component. While modules can help in organizing and reusing code, they are not specifically designed for generating and reusing a single value like a combination of random_id and a variable across multiple resources.

Data source

Explanation

Data sources in Terraform are used to fetch information from an external source, such as an API or a database, to use in your Terraform configuration. While data sources can provide dynamic information, they are not designed for generating and reusing values like a combination of random_id and a variable across multiple resources.

Outputs

Explanation

Outputs in Terraform are used to extract information from the Terraform state and make it available for other configurations or for external use. While outputs can be used to pass values between different parts of your Terraform configuration, they are not the ideal choice for generating and reusing a single value like a combination of random_id and a variable across multiple resources.

Overall explanation

The correct answer is: Local values

Explanation:

- **Local values** allow you to define a reusable value that can be referenced across multiple resources within a configuration. By defining the combination of the `random_id` and the variable as a local value, you avoid duplicating the same logic or expression in multiple resource definitions.
- This approach simplifies your code and makes it easier to maintain, as changes to the value only need to be made in one place.

Example:

```
variable "my_variable" {
 default = "example"
}

resource "random_id" "id" {
 byte_length = 8
}

locals {
 combined_tag = "${random_id.id.hex}-${var.my_variable}"
}

resource "aws_instance" "example1" {
 tags = {
   Name = local.combined_tag
 }
}

resource "aws_instance" "example2" {
 tags = {
   Name = local.combined_tag
 }
}
```

Why the others are incorrect:

- **Data source**: Data sources are used to fetch data from external systems or providers, not for storing or reusing computed values within a configuration.
- **Modules**: Modules encapsulate and reuse blocks of Terraform configurations, but they are overkill for this situation, as you're simply reusing a computed value.
- **Output**: Outputs are used to display values or pass data between modules, not for sharing values within the same configuration.

Resources

[Local Values]

Question 52Correct

Terraform installs its providers during which phase?

Refresh

Explanation

The Refresh phase is responsible for updating the state file with the current state of the infrastructure. It does not involve installing providers, but rather checks the current state against the desired state defined in the configuration files.

All of the above

Explanation

While all of the above phases are important in the Terraform workflow, the installation of providers specifically occurs during the Init phase. The Plan phase creates an execution plan, and the Refresh phase updates the state file, but neither of these phases involves provider installation.

Plan

Explanation

During the Plan phase, Terraform creates an execution plan that outlines what it will do when you call apply. This phase is focused on determining what actions need to be taken to achieve the desired state of the infrastructure, rather than installing providers.

Your answer is correct

Init

Explanation

The Init phase is where Terraform installs the necessary providers and modules specified in the configuration files. This phase ensures that all required dependencies are available before any actual infrastructure changes are made.

Overall explanation

Correct Answer: Init

Explanation:

Terraform installs its providers during the `terraform init` phase. When you run `terraform init`, Terraform initializes the working directory, downloads the necessary provider plugins, and prepares the environment for the execution of Terraform commands. This is the phase where Terraform ensures that it has all the required providers for the infrastructure you're defining.

- **Plan**: The `terraform plan` command is used to generate an execution plan but does not install providers. Providers must already be installed during initialization.
- **Refresh**: The `terraform refresh` command updates the state file to reflect the current state of the infrastructure but does not install or update providers.
- **All of the above**: This is incorrect because the only phase where providers are installed is during `terraform init`.

Resources

Terraform Init

Question 53 Correct

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

Versioning

Explanation

Versioning is another important benefit of infrastructure as code. By storing your infrastructure configurations in version control, you can track changes, revert to previous versions, and collaborate more effectively with team members.

Reusability of code

Explanation

Reusability of code is a key benefit of adopting infrastructure as code. By defining your infrastructure in code, you can easily reuse and replicate configurations across different environments or projects.

Your answer is correct

Interpolation

Explanation

Interpolation is actually a feature of Terraform that allows you to insert values into strings or configuration files. It is not a benefit of adopting infrastructure as code, but rather a specific functionality within Terraform itself.

Automation

Explanation

Automation is a fundamental benefit of adopting infrastructure as code. By defining your infrastructure in code, you can automate the provisioning, configuration, and management of your resources, leading to increased efficiency and consistency in your infrastructure deployments.

Overall explanation

The correct answer is: Interpolation

Explanation:

- **Interpolation** is a specific feature in Terraform that allows you to dynamically insert values into configurations. While useful, it is not a general benefit of adopting infrastructure as code (IaC). It is a functionality provided by some IaC tools, but not a core reason for using IaC.

Benefits of adopting IaC:

1. **Reusability of code**: IaC enables the use of modular and reusable code, reducing duplication and improving maintainability.
2. **Versioning**: IaC allows you to store configurations in version control systems, enabling tracking of changes and rollbacks.
3. **Automation**: IaC automates the provisioning, management, and decommissioning of infrastructure, reducing manual effort and human error.

Why interpolation is not a benefit:

Interpolation is a technical capability, not a high-level benefit or reason to adopt IaC. While useful in simplifying configurations, it does not broadly apply to all IaC tools or explain the value of IaC as a methodology.

Question 54Incorrect

Which of the following statements about **Terraform modules** is not true?

Correct answer

Modules must be publicly accessible

Explanation

This statement is not true. Modules 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 accessibility.

Module is a container for one or more resources

Explanation

This statement is true. A module in Terraform is a container for one or more resources that are used together. Modules help in organizing and reusing Terraform configurations.

Modules can be called multiple times

Explanation

This statement is true. Modules can be called multiple times within the same configuration to create multiple instances of the resources defined within the module.

Your answer is incorrect

Modules can call other modules

Explanation

This statement is true. Modules can call other modules, allowing for modular and reusable configurations in Terraform. This enables the building of complex infrastructure setups by composing smaller, reusable modules.

Overall explanation

Correct Answer: Modules must be publicly accessible

Explanation:

- **Modules must be publicly accessible**: This statement is not true. Modules in Terraform can be both public and private. You can store modules locally or in private repositories (e.g., GitHub, GitLab, etc.). They do not have to be publicly accessible. The module can be referenced locally or from a private registry.
- **Modules can be called multiple times**: This is true. Terraform modules can be called multiple times within the same configuration, which allows for reusability and consistency. Each instance can be parameterized differently.
- **Module is a container for one or more resources**: This is correct. A module is a logical container in Terraform that can encapsulate resources, input variables, output values, and other modules, providing a way to organize infrastructure as code.
- **Modules can call other modules**: This is also true. Terraform modules can be composed of other modules, allowing for hierarchical and reusable infrastructure designs. This feature helps to break down complex configurations into simpler, manageable units.

Resources

Terraform language modules

Question 55Correct

Which are examples of infrastructure as code? (Choose two.)

Your selection is correct

Versioned configuration files

Explanation

Versioned configuration files are examples of infrastructure as code. Versioned configuration files, such as Terraform configuration files, allow for defining and managing infrastructure resources in a declarative manner. These files can be version-controlled, enabling collaboration, tracking changes, and ensuring consistency in infrastructure deployments.

Change management database records

Explanation

Change management database records are not examples of infrastructure as code. Change management database records typically track changes made to infrastructure manually or through traditional methods, rather than defining and managing infrastructure through code.

Your selection is correct

Docker files

Explanation

Docker files are examples of infrastructure as code. Docker files are used to define the configuration of Docker containers, specifying the base image, dependencies, and commands to run within the container. By defining container configurations in code, Docker files enable reproducible and automated container deployments, aligning with the principles of infrastructure as code.

Cloned virtual machine images

Explanation

Cloned virtual machine images are not examples of infrastructure as code. Infrastructure as code refers to managing and provisioning infrastructure using code and automation tools, such as Terraform. Cloned virtual machine images are pre-configured snapshots of virtual machines and do not involve defining infrastructure through code.

Overall explanation

Correct Answer: Versioned configuration files, Docker files

Explanation:

- **Versioned configuration files** are an example of infrastructure as code because they define the configuration of infrastructure resources in a machine-readable format, and they can be version-controlled. These configuration files specify how resources should be provisioned, modified, and managed, making them a core part of the infrastructure as code (IaC) methodology.
- **Docker files** are another example of infrastructure as code. Dockerfiles are scripts that define the environment and application setup for a Docker container. By defining the infrastructure and application stack in code, Dockerfiles allow for repeatable and automated container deployment, which aligns with IaC principles.
- **Cloned virtual machine images** are not considered infrastructure as code by themselves. While they might be used to deploy infrastructure, they don't typically define the infrastructure in a declarative or version-controlled way like IaC does.
- **Change management database records** are also not examples of infrastructure as code. They may track infrastructure changes but don't directly define or automate the infrastructure setup itself.

Question 56Correct

Terraform variable names are saved in the state file.

Your answer is correct

False

Explanation

This choice is correct because Terraform variable names are not saved in the state file. Variables are used to customize configurations and provide flexibility without hardcoding values. The state file, on the other hand, stores the current state of the infrastructure managed by Terraform, including resource attributes and dependencies.

True

Explanation

Terraform variable names are not saved in the state file. The state file contains information about the resources that Terraform manages, their current state, and metadata. Variables are used to parameterize configurations and are typically defined in separate files or passed as input when running Terraform commands.

Overall explanation

Correct Answer: False

Explanation:

Terraform variable names themselves are **not** saved in the state file. The state file only contains the evaluated values of the variables as they apply to the actual resources in the infrastructure. This file maintains a mapping of resources and their current values in the cloud environment to ensure that the infrastructure matches the configuration defined in the Terraform code.

The state file does not need to know or store the variable names; it only uses the resulting values after variable interpolation and processing.

Question 57 Incorrect

You're writing a Terraform configuration that needs to read input from a local file called **id_rsa.pub**.

Which built-in Terraform function can you use to import the file's contents as a string?

Your answer is incorrect

fileset("id_rsa.pub")

Explanation

The fileset() function is used to retrieve a set of files matching a specific pattern within a directory. It is not designed to read the contents of a single file and return them as a string, so it is not the correct choice for importing the contents of id_rsa.pub.

filebase64("id_rsa.pub")

Explanation

The filebase64() function is used to encode a file's contents as a base64 string. While it can be useful for certain scenarios, it is not the appropriate function for simply reading the contents of a file as a string, so it is not the correct choice in this case.

Correct answer

file("id_rsa.pub")

Explanation

The file() function is the correct choice for importing the contents of a file as a string in Terraform. It reads the contents of the specified file and returns them as a string, making it the appropriate function to use in this scenario to read the contents of id_rsa.pub.

templatefile("id_rsa.pub")

Explanation

The templatefile() function is used to render a template file with variables provided as input. It is not intended for reading the raw contents of a file and returning them as a string, so it is not the correct choice for importing the contents of id_rsa.pub.

Overall explanation

Correct Answer: file("id_rsa.pub")

Explanation:

The `file()` function in Terraform reads the contents of a local file and returns it as a string. In this case, using `file("id_rsa.pub")` will load the contents of the `id_rsa.pub` file, allowing you to use it in your configuration as needed, such as for SSH keys.

- **fileset("id_rsa.pub")** is incorrect because `fileset()` is used for selecting multiple files in a directory based on a pattern, not for reading file contents.
- **filebase64("id_rsa.pub")** reads the file but encodes it as a base64 string, which is unnecessary unless you specifically need base64 encoding.
- **templatefile("id_rsa.pub")** is for reading template files with variables, which is more complex than simply importing a plain text file.

Resources