Terraform Associate Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Questions

1. When does terraform apply reflect changes in the cloud environment?

A. Immediately upon execution

B. After a manual approval

C. After the next scheduled run

D. However long it takes the resource provider to fulfill the request

2. Which backend does the Terraform CLI use by default?

A. Remote

B. Cloud

C. Local

D. Shared

3. If a module uses local values, what feature can be used to expose those values?

A. Terraform variable

B. Terraform output

C. Terraform data source

D. Terraform input

4. Which command is primarily used to create a new resource in Terraform?

A. terraform plan

B. terraform apply

C. terraform init

D. terraform destroy

5. Which Terraform collection type is best for storing key/value pairs?

A. List

B. Map

C. Set

D. Array

6. A Terraform provider is not responsible for:

- **A. Provisioning resources in a single cloud**
- **B. Managing configurations**
- **C. Provisioning infrastructure in multiple clouds**
- **D. Interacting with the cloud provider's API**

7. What is the purpose of the .terraform.lock.hcl file in Terraform?

- **A. To secure the Terraform configurations**
- **B. To track provider dependencies**
- **C. To log changes made**
- **D. To store user credentials securely**

8. What is true about Terraform Cloud tiers regarding team management?

- **A. Only the higher tiers support team management**
- **B. All Terraform Cloud tiers support team management and governance**
- **C. None of the tiers support team management**
- **D. Team management is a feature of local Terraform installations only**

9. Which of the following statements describes the behavior of Terraform when managing infrastructure?

- **A. It locks the state file after every change**
- **B. It allows manual changes to infrastructure without tracking**
- **C. It automatically updates the state file after any change made outside of Terraform**
- **D. It does not track changes made outside of Terraform until the next plan or apply**

10. How does Terraform know the order to create resources?

- **A. Through user-defined order in configuration**
- **B. Automatically based on dependencies**
- **C. Using defined tags for resource grouping**
- **D. Users must manually specify order**

Answers

1. D
2. C
3. B
4. B
5. B
6. C
7. B
8. B
9. D
10. B

Explanations

1. When does terraform apply reflect changes in the cloud environment?

A. Immediately upon execution

B. After a manual approval

C. After the next scheduled run

D. However long it takes the resource provider to fulfill the request

The correct response to when Terraform apply reflects changes in the cloud environment is that it occurs however long it takes the resource provider to fulfill the request. This means that after executing the `terraform apply` command, Terraform doesn't instantly update resources; rather, it sends requests to the respective cloud provider's API to create, update, or delete resources as specified in the Terraform configuration files. The time it takes for changes to be reflected in the cloud environment is dependent on the provider's system and how quickly it can process the request. For example, some resources may be provisioned almost instantly, while others might take several minutes or longer depending on various factors such as the complexity of the resource or the current workload of the cloud provider. In contrast, immediate execution implies that changes would show up in the cloud environment as soon as the apply command runs, which is not accurate because it does not account for the time needed for the provider to process these requests. Manual approvals do not normally occur in standard deployment processes for infrastructure as code in Terraform, as the apply command is generally automated. Lastly, thinking that changes would only reflect after a scheduled run does not align with how Terraform's apply function operates, as it performs actions on demand rather than on a schedule.

2. Which backend does the Terraform CLI use by default?

A. Remote

B. Cloud

C. Local

D. Shared

The Terraform CLI uses the local backend by default, which means that it stores the state file on the local filesystem. This state file is crucial because it keeps track of the resources that Terraform manages, providing a mapping between the configuration and the actual deployment. When you initialize a Terraform project without specifying a backend, it automatically creates or looks for a `terraform.tfstate` file in the current working directory. Using the local backend allows users to work with Terraform in a straightforward manner, especially for small projects or when experimenting. However, as projects scale or collaboration becomes necessary, transitioning to a more robust backend, such as remote or cloud backends, becomes common practice. These backends offer additional features like state locking and remote storage, which are not available with the local backend. Other options like remote, cloud, and shared do not represent the default behavior when first using Terraform, and they require explicit configuration to be utilized. This local backend is a fundamental aspect of how Terraform operates, making it essential to understand for effective usage.

3. If a module uses local values, what feature can be used to expose those values?

A. Terraform variable

B. Terraform output

C. Terraform data source

D. Terraform input

When a module uses local values, Terraform outputs are the feature that can be used to expose those values. Outputs in Terraform allow you to define return values from a module that can be accessed from the parent module or elsewhere in your configuration. This is particularly useful when you want other parts of your infrastructure to use computed values generated within a module, making the outputs a vital communication interface between modules. By defining outputs, you provide a way to retrieve data that is not directly accessible outside of the module itself. This can include local values that have been calculated for use within the module, which you now want to share at a higher level. For example, an output might expose the result of a calculation or a reference to a resource created within the module, enabling other parts of your Terraform configuration to utilize this data effectively. In contrast, variables serve as inputs to modules rather than a means to expose internal values, data sources are used to fetch and reference data from outside your Terraform configuration, and inputs are parameters that allow you to customize module behavior at runtime. Outputs serve a distinct purpose in encapsulating and sharing internal data from a module, making them the appropriate choice for exposing local values.

4. Which command is primarily used to create a new resource in Terraform?

A. terraform plan

B. terraform apply

C. terraform init

D. terraform destroy

The command primarily used to create a new resource in Terraform is `terraform apply`. This command is responsible for applying the planned changes that Terraform generates based on the configuration files you have defined. When you run `terraform apply`, Terraform will create, modify, or delete resources in your cloud provider's environment to match the desired state specified in your configuration files. Before actually applying any changes, `terraform apply` may prompt the user to confirm the execution of the planned actions. This provides an opportunity to review changes before they are implemented. If the resources do not already exist, this command will create them, effectively setting up the infrastructure that your Terraform configuration describes. The other commands serve different purposes. For instance, `terraform plan` generates an execution plan that shows what actions Terraform will take but does not actually create any resources. `terraform init` is used to initialize a Terraform working directory and prepare it for other commands, while `terraform destroy` is intended to remove existing resources rather than create them. Thus, `terraform apply` is the correct answer for creating new resources in Terraform.

5. Which Terraform collection type is best for storing key/value pairs?

A. List

B. Map

C. Set

D. Array

The best collection type for storing key/value pairs in Terraform is a Map. A Map is an unordered collection composed of unique keys that are associated with values. This structure allows for easy retrieval and management of data based on the specified keys, making it suitable for various configuration scenarios. Maps are particularly advantageous when you want to associate specific configurations or settings with identifiable keys, as they provide greater readability and organization. For example, in scenarios where you are defining resource attributes or passing configuration variables, using a Map can help you define those relationships clearly. On the other hand, Lists, Sets, and Arrays do not support key/value data structures. Lists are ordered collections that allow duplicate values but do not associate values with unique keys. Sets, while also unordered and unique, only store values without a corresponding key, which limits their functionality for storing pairs. Arrays are not a distinct collection type in Terraform terminology and typically refer to ordered lists in programming, again without key associations. Thus, a Map is the optimal choice for managing key/value pairs in Terraform.

6. A Terraform provider is not responsible for:

A. Provisioning resources in a single cloud

B. Managing configurations

C. Provisioning infrastructure in multiple clouds

D. Interacting with the cloud provider's API

A Terraform provider is specifically designed to facilitate interactions with a cloud provider's API and to manage resources. Its primary role is to abstract the complexities of the underlying infrastructure and allow Terraform configurations to interact with various resources provided by cloud services. The option indicating that a provider is not responsible for provisioning infrastructure in multiple clouds reflects a nuanced understanding of providers. While some providers are built for specific cloud platforms, Terraform itself can integrate multiple providers, enabling users to define infrastructure across multiple cloud environments with a single configuration file. However, a single provider typically focuses on resources within a single cloud or ecosystem. Providers do indeed manage resources within a specific cloud environment and are responsible for provision operations needed to ensure those resources are created, updated, or destroyed as needed. They interact with the APIs of cloud providers to perform these tasks. Therefore, it is accurate to state that each provider does not inherently handle the orchestration of resources across multiple clouds; that responsibility typically lies with the overall Terraform execution and configuration rather than individual providers.

7. What is the purpose of the .terraform.lock.hcl file in Terraform?

A. To secure the Terraform configurations

B. To track provider dependencies

C. To log changes made

D. To store user credentials securely

The .terraform.lock.hcl file is essential for maintaining the integrity and consistency of provider dependencies in Terraform projects. When you run Terraform commands that involve providers, the tool resolves the required provider versions and their dependencies. These resolved versions are documented in the .terraform.lock.hcl file. This file ensures that anyone who executes the same Terraform configuration in the future will use the exact same versions of the providers, leading to predictable and reproducible infrastructure deployments. By locking the provider versions, it prevents unintentional upgrades that could introduce breaking changes or unexpected behavior in your infrastructure. This is particularly valuable in collaborative environments or in production systems where stability is crucial. The other options do not accurately represent the file's purpose. While securing configurations, logging changes, and storing user credentials are important aspects of managing Terraform projects, they do not pertain to the specific role of the .terraform.lock.hcl file in tracking provider dependencies.

8. What is true about Terraform Cloud tiers regarding team management?

A. Only the higher tiers support team management

B. All Terraform Cloud tiers support team management and governance

C. None of the tiers support team management

D. Team management is a feature of local Terraform installations only

All Terraform Cloud tiers indeed support team management and governance features. This capability allows users to create teams within their organization, assign roles and permissions to those teams, and manage their access to resources and workflows in a more structured way. This is essential for organizations that require collaborative efforts within their DevOps teams and want to ensure that different teams can operate within their own workflows while adhering to organizational policies. This functionality is beneficial for streamlining access control and maintaining best practices across teams. By supporting team management in all tiers, Terraform Cloud provides flexibility and scalability for teams of different sizes, accommodating various organizational needs. The other options do not accurately reflect the capabilities of Terraform Cloud. While higher tiers may offer advanced features, the correct position is that all tiers provide fundamental team management capabilities. This inclusivity ensures that even users at the basic level can benefit from effective collaboration tools that facilitate governance and secure resource management.

9. Which of the following statements describes the behavior of Terraform when managing infrastructure?

A. It locks the state file after every change

B. It allows manual changes to infrastructure without tracking

C. It automatically updates the state file after any change made outside of Terraform

D. It does not track changes made outside of Terraform until the next plan or apply

The behavior of Terraform in managing infrastructure is accurately captured by the assertion that it does not track changes made outside of Terraform until the next plan or apply is executed. When Terraform manages infrastructure, it maintains a state file that reflects the current configuration of the managed resources. If changes are made directly to the infrastructure outside of Terraform (such as through the cloud provider's console), those alterations are not immediately recognized by Terraform. During the next execution of a `terraform plan` or `terraform apply`, Terraform will then compare the actual infrastructure against the state file to identify discrepancies. This allows Terraform to reflect any changes that have occurred outside of its purview during the next operations, enabling users to see what is different and manage those changes accordingly. The other options do not align with Terraform's behavior: locking the state file after every change is not how Terraform works, as it only locks the state during critical operations; allowing manual changes without tracking is counter to the principles of infrastructure as code; and automatic state file updates after external changes would undermine Terraform's purpose of tracking and applying infrastructure as defined in its configuration files.

10. How does Terraform know the order to create resources?

A. Through user-defined order in configuration

B. Automatically based on dependencies

C. Using defined tags for resource grouping

D. Users must manually specify order

Terraform determines the order to create resources primarily through automatic dependency management. When Terraform analyzes the configuration files, it assesses the relationships between different resources. For instance, if one resource needs to reference the output of another resource (like an IP address or a database ID), Terraform understands these dependencies and constructs a dependency graph. This ensures that resources are created in the correct order, with each resource being provisioned only after its dependencies have been satisfied. This automatic handling of dependencies alleviates the requirement for users to manually specify the order in which resources should be created, as Terraform intelligently interprets the necessary relationships. As a result, Terraform can optimize the resource creation process while ensuring that everything is provisioned successfully without user intervention in the order of operations.