Question 1Skipped

True or False? Before installing and using Terraform, you must download and install Go as a prerequisite.

True

Explanation

False. Installing Go (the programming language) is not a prerequisite for installing and using Terraform. Terraform is distributed as a standalone binary executable that can be downloaded and installed directly from the Terraform website or via package managers like Homebrew (on macOS) or Chocolatey (on Windows).

Correct answer

False

Explanation

Correct. Installing Go (the programming language) is **not a prerequisite** for installing and using Terraform. Terraform is distributed as a standalone binary executable that can be downloaded and installed directly from the Terraform website or via package managers like Homebrew (on macOS) or Chocolatey (on Windows).

Overall explanation

While Go is used by HashiCorp developers to build Terraform from its source code, end-users do not need to install Go themselves. They can simply download the pre-built Terraform binary suitable for their operating system and architecture, making the installation process straightforward and independen of Go.

https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

Domain

Objective 1 - Understand Infrastructure as Code concepts

Question 2Skipped

True or False? You can migrate the Terraform backend but only if there are no resources currently being managed.

True

Correct answer

False

Overall explanation

If you are already using Terraform to manage infrastructure, you probably want to transfer to another backend, such as Terraform Cloud, so you can continue managing it. By migrating your Terraform state, you can hand off infrastructure without de-provisioning anything.

https://learn.hashicorp.com/tutorials/terraform/cloud-migrate

Domain

Objective 7 - Implement and Maintain State

Question 3Skipped

What are some problems with how infrastructure was traditionally managed before Infrastructure as Code? (select three)

Correct selection

Traditional deployment methods are not able to meet the demands of the modern business where resources tend to live days to weeks, rather than months to years

Correct selection

Requests for infrastructure or hardware often required a ticket, increasing the time required to deploy applications

Pointing and clicking in a management console is a scalable approach and reduces human error as businesses are moving to a multi-cloud deployment model

Correct selection

Traditionally managed infrastructure can't keep up with cyclic or elastic applications

Overall explanation

Traditionally, infrastructure was managed using manual processes and user interfaces, which could lead to several problems, including:

1. **Configuration drift**: With manual configuration, it can be difficult to ensure that all infrastructure components are consistently configured. Over time, differences in configuration can accumulate, leading to configuration drift, where systems in the same environment are no longer identical.
2. **Lack of standardization**: Manual configuration can also result in inconsistencies across environments, which can make it difficult to manage and troubleshoot infrastructure. For example, different environments may have different versions of software or different security settings, making it hard to replicate issues or ensure consistent behavior.
3. **Slow provisioning**: Provisioning infrastructure manually can be time-consuming, especially for complex configurations or when setting up multiple resources. This can lead to delays in development and deployment, as teams may need to wait for infrastructure to be set up before they can begin work.
4. **Human error**: Manual provisioning and configuration is prone to human error, which can lead to security vulnerabilities, performance issues, or downtime. For example, a misconfigured firewall rule could leave systems open to attack, or a typo in a configuration file could cause a system to crash.
5. **Difficulty in documentation**: With manual configuration, it can be challenging to keep documentation up to date and accurate. This can make it hard for teams to understand

how infrastructure is configured, what changes have been made, and how to troubleshoot issues.

Overall, these problems can make it difficult to manage infrastructure at scale and can lead to increased costs, reduced agility, and increased risk of errors and downtime. Infrastructure as Code helps to address many of these issues by providing a standardized, repeatable, and automated way to manage infrastructure resources.

https://developer.hashicorp.com/terraform/intro#infrastructure-as-code

Domain

Objective 1 - Understand Infrastructure as Code concepts

Question 4Skipped

What is the best and easiest way for Terraform to read and write secrets from HashiCorp Vault?

CLI access from the same machine running Terraform

integration with a tool like Jenkins

API access using the AppRole auth method

Correct answer

Vault provider

Overall explanation

The Vault provider allows Terraform to read from, write to, and configure Hashicorp Vault.

https://registry.terraform.io/providers/hashicorp/vault/latest/docs

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 5Skipped

In Terraform Cloud, a workspace can be mapped to how many VCS repos?

5

Correct answer

1

3

2

Overall explanation

A workspace can only be configured to a single VCS repo, however, multiple workspaces can use the same repo, if needed. A good explanation of how to configure your code repositories can be found here.

https://developer.hashicorp.com/terraform/cloud-docs/workspaces/creating

Domain

Objective 9 - Understand Terraform Cloud Capabilities

Question 6Skipped

When using modules to deploy infrastructure, how would you export a value from one module to import into another module?

For example, a module dynamically deploys an application instance or virtual machine, and you need the IP address in another module to configure a related DNS record in order to reach the newly deployed application.

preconfigure the IP address as a parameter in the DNS module

configure the pertinent provider's configuration with a list of possible IP addresses to use

export the value using terraform export and input the value using terraform input

Correct answer

configure an output value in the application module in order to use that value for the DNS module

Overall explanation

Output values are like the return values of a Terraform module and have several uses such as a child module using those outputs to expose a subset of its resource attributes to a parent module.

https://developer.hashicorp.com/terraform/language/expressions#references-to-named-values

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 7Skipped

Terry is using a module to deploy some EC2 instances on AWS for a new project. He is viewing the code that is calling the module for deployment, which is shown below. *Where is the value of the security group originating?*

```
1. module "ec2_instances" {
2.   source  = "terraform-aws-modules/ec2-instance/aws"
3.   version = "4.3.0"
4.
5.   name          = "my-ec2-cluster"
6.   instance_count = 2
7.
```

```
8.  ami                = "ami-0c5204531f799e0c6"
9.  instance_type      = "t2.micro"
10. vpc_security_group_ids = [module.vpc.default_security_group_id]
11. subnet_id          = module.vpc.public_subnets[0]
12.
13. tags = {
14.   Terraform   = "true"
15.   Environment = "dev"
16. }
```

Correct answer

the output of another module

an environment variable being using during a terraform apply

from a variable likely declared in a .tfvars file being passed to another module

the Terraform public registry

Overall explanation

- The required vpc_security_group_ids and subnet_id arguments reference resources created by the vpc module. The Terraform Registry page contains the full list of arguments for the ec2-instance module.

- A great tutorial to look at this workflow can be found on the HashiCorp Learn site - https://learn.hashicorp.com/tutorials/terraform/module-use

Domain

Objective 5 - Interact with Terraform Modules

Question 8Skipped

Using multi-cloud and provider-agnostic tools provides which of the following benefits? (select two)

slower provisioning speed allows the operations team to catch mistakes before they are applied

increased risk due to all infrastructure relying on a single tool for management

Correct selection

operations teams only need to learn and manage a single tool to manage infrastructure, regardless of where the infrastructure is deployed

Correct selection

can be used across major cloud providers and VM hypervisors

Overall explanation

Using a tool like Terraform can be advantageous for organizations deploying workloads across multiple public and private cloud environments. Operations teams only need to learn a single tool, a single language, and can use the same tooling to enable a DevOps-like experience and workflows.

https://developer.hashicorp.com/terraform/intro/use-cases#multi-cloud-deployment

Domain

Objective 2 - Understand Terraform's purpose (vs. other IaC)

Question 9Skipped

Terraform language has built-in syntax for creating lists. Which of the following is the correct syntax to create a list in Terraform?

<"string1", "string2", "string3">

("string1", "string2", "string3")

{"string1", "string2", "string3"}

Correct answer

["string1", "string2", "string3"]

Overall explanation

Terraform language has built-in syntax for creating lists using the [and] delimiters.

https://developer.hashicorp.com/terraform/language/expressions/types#list

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 10Skipped

Which of the following best describes the default local backend?

The local backend is the directory where resources deployed by Terraform have direct access to in order to update their current state

The local backend is where Terraform stores logs to be processed by a log collector

Correct answer

The local backend stores state on the local filesystem, locks the state using system APIs, and performs operations locally

The local backend is how Terraform connects to public cloud services, such as AWS, Azure, or GCP

Overall explanation

Information on the default local backend can be found at this link.

Example:

```
terraform {
  backend "local" {
    path = "relative/path/to/terraform.tfstate"
  }
}
```

Domain

Objective 7 - Implement and Maintain State

Question 11Skipped

After running into issues with Terraform, you need to enable verbose logging to assist with troubleshooting the error. Which of the following values provides the **MOST** verbose logging?

DEBUG

INFO

WARN

ERROR

Correct answer

TRACE

Overall explanation

In Terraform, you can enable verbose logging by using the **-debug** command line option or setting the **TF_LOG** environment variable to **DEBUG**. This will provide additional log messages to help with troubleshooting errors.

However, if you need even more detailed logging, you can set the **TF_LOG** environment variable to **TRACE**. This will provide the most verbose logging, including every step taken by Terraform during plan, apply, and destroy operations, as well as additional debugging information.

Here's an example of how to set the **TF_LOG** environment variable to **TRACE** on a Unix-based system:

```
$ export TF_LOG=TRACE
```

Note that enabling verbose logging can result in a large amount of output, so it should only be used when necessary for troubleshooting purposes. Once you have resolved the issue, you can turn off verbose logging by removing the **TF_LOG** environment variable or set it to a lower level, such as **DEBUG** or **INFO**.

https://developer.hashicorp.com/terraform/internals/debugging

Domain

Objective 4 - Use Terraform Outside of Core Workflow

Question 12Skipped

Which of the following actions are performed during a terraform init? (select three)

Correct selection

initializes the backend configuration

provisions the declared resources in your configuration

Correct selection

downloads the required modules referenced in the configuration

Correct selection

downloads the providers/plugins required to execute the configuration

Overall explanation

The terraform init command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

https://developer.hashicorp.com/terraform/cli/commands/init

https://learn.hashicorp.com/tutorials/terraform/aws-build#create-infrastructure

Domain

Objective 6 - Use the Core Terraform Workflow

Question 13Skipped

Kristen is using modules to provision an Azure environment for a new application. She is using the following code to specify the version of her virtual machine module. Which of the following Terraform features supports the versioning of a module? (select two)

```
1. module "compute" {
2.   source  = "azure/compute/azurerm"
3.   version = "5.1.0"
4.
5.   resource_group_name = "production_web"
6.   vnet_subnet_id     = azurerm_subnet.aks-default.id
7. }
```

Correct selection

Terraform registry

Correct selection

private registry

modules stored in GitLab

local file paths

Overall explanation

Version constraints are supported only for modules installed from a module registry, such as the public Terraform Registry or Terraform Cloud's private registry. Other module sources can provide their own versioning mechanisms within the source string itself, or might not support versions at all. In particular, modules sourced from local file paths do not support version; since they're loaded from the same source repository, they always share the same version as their caller.

https://developer.hashicorp.com/terraform/language/modules/syntax#version

Domain

Objective 5 - Interact with Terraform Modules

Question 14Skipped

Your organization has moved to AWS and has manually deployed infrastructure using the console. Recently, a decision has been made to standardize on Terraform for all deployments moving forward.

What can you do to ensure that the existing resources are managed by Terraform moving forward without causing interruption to existing resources?

resources that are manually deployed in the AWS console cannot be imported by Terraform

submit a ticket to AWS and ask them to export the state of all existing resources and use terraform import to import them into the state file

Correct answer

using terraform import, import the existing infrastructure to bring the resources under Terraform management

delete the existing resources and recreate them using new a Terraform configuration so Terraform can manage them moving forward

Overall explanation

To ensure that existing resources in AWS are managed by Terraform moving forward without causing interruption to existing resources, there are a few steps that you can follow:

1. **Create a new Terraform configuration that represents the existing resources in AWS**. This can be done manually by examining the resources in the AWS console and

recreating them in Terraform code, or automatically by using a tool like Terraforming or CloudMapper.

2. Import the existing resources into Terraform using the **terraform import** command. This command allows you to associate the existing resources in AWS with the new Terraform configuration. You will need to specify the resource type, name, and ID for each resource you want to import.
3. Use Terraform to manage all future changes to the infrastructure. With the existing resources now managed by Terraform, you can make changes to them through Terraform code and use the normal Terraform workflow of plan, apply, and destroy to manage the infrastructure going forward.

It's important to note that importing existing resources into Terraform can be a complex and error-prone process, especially for large and complex infrastructures. It's recommended to test the import process thoroughly in a development or staging environment before attempting to import production resources. Additionally, be sure to carefully review the Terraform code before running **terraform apply** to avoid accidentally modifying or deleting existing resources.

https://developer.hashicorp.com/terraform/cli/commands/import

Domain

Objective 4 - Use Terraform Outside of Core Workflow

Question 15Skipped

Which of the following best describes a Terraform provider?

a container for multiple resources that are used together

Correct answer

a plugin that Terraform uses to translate the API interactions with the service or provider

describes an infrastructure object, such as a virtual network, compute instance, or other components

serves as a parameter for a Terraform module that allows a module to be customized

Overall explanation

In Terraform, a **provider** is a plugin that enables Terraform to interact with a specific cloud or service provider, such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). Providers are responsible for understanding the APIs and resources the target infrastructure platform provides and for translating Terraform configuration code into API calls that can create, read, update, and delete resources.

Each provider typically consists of resource types, data sources, and other settings that define the provider's capabilities within Terraform. These resources and data sources correspond to the resources that can be managed within the target infrastructure, such as virtual machines, storage accounts, or network interfaces.

To use a provider in Terraform, you must first configure it in your Terraform code by specifying the provider's name and any required configuration settings, such as access keys, secret keys,

or region. Once the provider is configured, you can then use its resources and data sources in your Terraform code to define the infrastructure you want to manage.

Terraform has a large and growing ecosystem of third-party providers that support a wide range of infrastructure platforms and services, as well as an official set of core providers maintained by HashiCorp, the company behind Terraform. The availability and quality of providers is a crucial factor in the usefulness of Terraform as a tool for managing infrastructure as code.

https://developer.hashicorp.com/terraform/language/providers

Domain

Objective 3 - Understand Terraform Basics

Question 16Skipped

What Terraform command can be used to inspect the current state file?

Example:

```
# aws_instance.example:
resource "aws_instance" "example" {
    ami                          = "ami-2757f631"
    arn                          = "arn:aws:ec2:us-east-1:130490850807:instance/i-0
    associate_public_ip_address  = true
    availability_zone            = "us-east-1c"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-0bbf06244e44211d1"
    instance_state               = "running"
    instance_type                = "t2.micro"
```

terraform read

terraform inspect

Correct answer

terraform show

terraform state

Overall explanation

The terraform show command is used to provide human-readable output from a state or plan file. This can be used to inspect a plan to ensure that the planned operations are expected, or to inspect the current state as Terraform sees it.

Machine-readable output can be generated by adding the -json command-line flag.

Note: When using the -json command-line flag, any sensitive values in Terraform state will be displayed in plain text.

https://developer.hashicorp.com/terraform/cli/commands/show

Domain

Objective 7 - Implement and Maintain State

Question 17Skipped

Why is it a good idea to declare the required version of a provider in a Terraform configuration file?

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.57.0"
    }
  }
}
```

to match the version number of your application being deployed via Terraform

to remove older versions of the provider

to ensure that the provider version matches the version of Terraform you are using

Correct answer

providers are released on a separate schedule from Terraform itself; therefore, a newer version could introduce breaking changes

Overall explanation

Declaring the required version of a provider in a Terraform configuration file is a good idea for several reasons:

1. **Reproducibility**: By specifying the exact version of a provider, you can ensure that anyone who runs your Terraform configuration will use the same version of the provider as you. This makes your infrastructure configuration more reproducible and helps avoid issues that can arise when different versions of a provider are used.
2. **Predictability**: When you specify a specific provider version, you can be confident that your infrastructure configuration will behave predictably, regardless of changes to the provider in future versions. ***This can help you avoid surprises and reduce the risk of unintended consequences.***

3. **Compatibility**: Different versions of a provider may have different APIs, resources, or behaviors, which can cause issues if you switch to a new version without realizing the differences. By specifying the required version of a provider in your Terraform configuration, you can ensure that your configuration remains compatible with the specific version of the provider you have tested and validated.
4. **Version locking**: When you specify the required version of a provider in your Terraform configuration, you effectively lock the version of the provider to that version unless you explicitly change it. This can help prevent issues that may arise when using a different, potentially incompatible version of the provider.

In summary, specifying the required version of a provider in your Terraform configuration file helps ensure that your infrastructure configuration is more predictable, reproducible, compatible, and reduces the risk of unintended consequences or issues caused by version differences.

https://developer.hashicorp.com/terraform/language/providers/requirements#requiring-providers

Domain

Objective 3 - Understand Terraform Basics

Question 18Skipped

You have been given requirements to create a security group for a new application. Since your organization standardizes on Terraform, you want to add this new security group with the fewest number of lines of code. What feature could you use to iterate over a list of required tcp ports to add to the new security group?

terraform import

splat expression

dynamic backend

Correct answer

dynamic block

Overall explanation

A dynamic block acts much like a for expression, but produces nested blocks instead of a complex typed value. It iterates over a given complex value and generates a nested block for each element of that complex value.

You can find more information on dynamic blocks using this link.

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 19Skipped

In order to make a Terraform configuration file dynamic and/or reusable, static values should be converted to use what?

Correct answer

input variables

module

regular expressions

output value

Overall explanation

Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurations.

https://learn.hashicorp.com/tutorials/terraform/aws-variables

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 20Skipped

True or False? Workspaces provide similar functionality in the open-source and Terraform Cloud versions of Terraform.

Correct answer

False

True

Overall explanation

Workspaces, managed with the terraform workspace command, isn't the same thing as Terraform Cloud's workspaces. Terraform Cloud workspaces act more like completely separate working directories.

CLI workspaces (OSS) are just alternate state files.

Question 21Skipped

Which of the following is considered a Terraform plugin?

Correct answer

Terraform provider

Terraform logic

Terraform language

Terraform tooling

Overall explanation

In Terraform, a plugin is a binary executable that implements a specific provider. **A provider is a plugin that allows Terraform to manage a specific cloud provider or service.**

When Terraform runs, it loads the plugins required to manage the resources specified in the configuration files. Each provider has its own plugin, and Terraform loads the plugins for the providers specified in the configuration.

The plugin is responsible for interacting with the cloud provider's API, translating Terraform configurations into API calls, and managing the state of the resources that Terraform manages.

Plugins are stored in the Terraform plugin cache, a directory on the local machine that contains the binary executables for each plugin. When Terraform runs, it looks for plugins in the cache and automatically downloads any missing plugins from the Terraform Registry or a specified source.

Terraform plugins are written in Go and follow a specific plugin protocol, which defines the interactions between Terraform and the plugin. The plugin protocol allows Terraform to communicate with the plugin and provides a standard way for plugins to manage resources across different providers.

https://developer.hashicorp.com/terraform/plugin

Domain

Objective 3 - Understand Terraform Basics

Question 22Skipped

You found a module on the Terraform Registry that will provision the resources you need. What information can you find on the Terraform Registry to help you quickly use this module? (select three)

A Download button to Quickly Get the Module Code

Explanation

A Download button to Quickly Get the Module Code is not typically found on the Terraform Registry. While you can access the module code from the registry, the focus is more on providing information about the module, such as input variables, outputs, and dependencies, rather than offering a direct download button.

Correct selection

Dependencies to use the Module

Explanation

Dependencies to use the Module is valuable information that can be found on the Terraform Registry. Dependencies specify any other modules or resources that the module relies on to function properly. Knowing the dependencies helps you ensure that all necessary components are in place before using the module.

Correct selection

A list of Outputs

Explanation

A list of Outputs is another important piece of information available on the Terraform Registry for a module. Outputs define the values that are exposed by the module after it has been applied. Understanding the outputs helps you utilize the resources provisioned by the module in your Terraform configuration.

Correct selection

Required Input Variables

Explanation

Required Input Variables are crucial information that you can find on the Terraform Registry for a module. Knowing the required input variables helps you understand what values need to be provided to the module for it to function correctly. This information is essential for quickly using the module without any errors.

Overall explanation

The Terraform Registry makes it simple to find and use modules.

Every page on the registry has a search field for finding modules. Enter any type of module you're looking for (examples: "vault," "vpc," "database"), and the resulting modules will be listed. The search query will look at the module name, provider, and description to match your search terms. On the results page, filters can be used to further refine search results.

https://developer.hashicorp.com/terraform/registry/modules/use

Domain

Objective 5 - Interact with Terraform Modules

Question 23Skipped

You want to start managing resources that were not originally provisioned through infrastructure as code. Before you can import the resource's current state, what must you do before running the terraform import command?

Correct answer

update the Terraform configuration file to include the new resources that match the resources you want to import

shut down or stop using the resources being imported so no changes are inadvertently missed

modify the Terraform state file to add the new resources so Terraform will have a record of the resources to be managed

run terraform apply -refresh-only to ensure that the state file has the latest information for existing resources.

Overall explanation

NOTE: HashiCorp has released functionality that automatically generates the Terraform configuration for imported resources. However, please keep in mind that the exam

questions are not updated immediately, and therefore, you may find that the exam still expects you to know you should create the configuration yourself.

The current implementation of Terraform import can only import resources into the state. It does not generate a configuration. Because of this, and before running terraform import, it is necessary to manually write a resource configuration block for the resource to which the imported object will be mapped.

First, add the resources to the configuration file:

```
1. resource "aws_instance" "example" {
2.   # ...instance configuration...
3. }
```

Then run the following command:

$ terraform import aws_instance.example i-abcd1234

https://developer.hashicorp.com/terraform/cli/commands/import

Domain

Objective 4 - Use Terraform Outside of Core Workflow

Question 24Skipped

In regards to Terraform state file, select all the statements below which are correct: (select four)

Correct selection

Terraform Cloud always encrypts state at rest

the state file is always encrypted at rest

using the mask feature, you can instruct Terraform to mask sensitive data in the state file

Correct selection

the Terraform state can contain sensitive data, therefore the state file should be protected from unauthorized access

Correct selection

storing state remotely can provide better security

Correct selection

when using local state, the state file is stored in plain-text

Overall explanation

Terraform state can contain sensitive data, depending on the resources in use and your definition of "sensitive." The state contains resource IDs and all resource attributes. For resources such as databases, this may contain initial passwords.

When using local state, state is stored in plain-text JSON files.

If you manage any sensitive data with Terraform (like database passwords, user passwords, or private keys), treat the state itself as sensitive data.

Storing Terraform state remotely can provide better security. As of Terraform 0.9, Terraform does not persist state to the local disk when remote state is in use, and some backends can be configured to encrypt the state data at rest.

https://developer.hashicorp.com/terraform/language/state/sensitive-data

Domain

Objective 7 - Implement and Maintain State

Question 25Skipped

What are the core Terraform workflow steps to use infrastructure as code?

1. 1) Plan
2. 2) Apply
3. 3) Pray

Correct answer

1. 1) Write
2. 2) Plan
3. 3) Apply

1. 1) Plan
2. 2) Apply
3. 3) Destroy

1. 1) Code
2. 2) Validate
3. 3) Apply

Overall explanation

The core Terraform workflow has three steps:

- **Write** - Author infrastructure as code.

- **Plan** - Preview changes before applying.

- **Apply** - Provision reproducible infrastructure.

This guide walks through how each of these three steps plays out in the context of working as an individual practitioner, how they evolve when a team is collaborating on infrastructure, and how Terraform Cloud enables this workflow to run smoothly for entire organizations.

https://developer.hashicorp.com/terraform/intro/core-workflow

Domain

Objective 6 - Use the Core Terraform Workflow

Question 26Skipped

What does the command terraform fmt do?

deletes the existing configuration file

Correct answer

rewrite Terraform configuration files to a canonical format and style

updates the font of the configuration file to the official font supported by HashiCorp

formats the state file in order to ensure the latest state of resources can be obtained

Overall explanation

The **terraform fmt** command is a formatting tool in Terraform that helps to automatically format Terraform configuration files to follow a consistent style and make them more readable.

Running **terraform fmt** will parse the configuration files in the current directory and recursively in subdirectories and rewrite them using a standard formatting style, including indentation, spacing, and line breaks. It will modify the original files in place, so it's vital to ensure that the files are backed up or committed to a version control system before running this command.

By running **terraform fmt**, it helps to ensure that the Terraform configuration files are consistent across the project and easy to read, especially when working with large and complex infrastructure codebases. Consistent code style makes it easier for multiple people to collaborate on a project and makes it easier to understand the configuration files when returning to the project after an extended period.

It's a best practice to run **terraform fmt** before committing any changes to the configuration files, to ensure that all changes have the same formatting style and are easy to read.

https://developer.hashicorp.com/terraform/cli/commands/fmt

Domain

Objective 4 - Use Terraform Outside of Core Workflow

Question 27Skipped

After executing a terraform plan, you notice that a resource has a tilde (~) next to it. What does this mean?

the resource will be created

Terraform can't determine how to proceed due to a problem with the state file

the resource will be destroyed and recreated

Correct answer

the resource will be updated in place

Overall explanation

The prefix -/+ means that Terraform will destroy and recreate the resource, rather than updating it in-place. Some attributes and resources can be updated in-place and are shown with the ~ prefix.

```
  ~ root_block_device {
      ~ delete_on_termination = true -> (known after apply)
      ~ iops                  = 100 -> (known after apply)
      ~ volume_id             = "vol-0079e485d9e28a8e5" -> (known after apply)
      ~ volume_size           = 8 -> (known after apply)
      ~ volume_type           = "gp2" -> (known after apply)
    }
}
```

https://developer.hashicorp.com/terraform/cli/commands/plan

https://learn.hashicorp.com/tutorials/terraform/infrastructure-as-code

Domain

Objective 6 - Use the Core Terraform Workflow

Question 28Skipped

In the terraform block, which configuration would be used to identify the specific version of a provider required?

required_versions

Correct answer

required_providers

required-provider

required-version

Overall explanation

To identify a specific version of a provider in Terraform, you can use the **required_providers** configuration block. This block allows you to specify the provider's name and the version range you want to use by using Terraform's version constraints syntax.

Here's an example of how to use the **required_providers** block to specify a specific version of the AWS provider:

```
terraform {
 required_providers {
  aws = {
   source = "hashicorp/aws"
   version = "3.57.0"
  }
 }
}
```

In this example, we're specifying that we require version 3.57.0 of the AWS provider, which is hosted at the **hashicorp/aws** source. Note that the version constraint syntax allows you to specify a range of versions using operators such as **>=** and **<=**.

When you run **terraform init** with this configuration, Terraform will download and install the specified version of the AWS provider, and will use it for all subsequent Terraform commands for that module. If the specified version is not available, Terraform will return an error and fail to initialize the configuration.

https://developer.hashicorp.com/terraform/language/providers/requirements#requiring-providers

Domain

Objective 3 - Understand Terraform Basics

Question 29Skipped

What are the benefits of using Infrastructure as Code? (select five)

Correct selection

Infrastructure as Code gives the user the ability to recreate an application's infrastructure for disaster recovery scenarios

Correct selection

Infrastructure as Code allows a user to turn a manual task into a simple, automated deployment

Correct selection

Infrastructure as Code is easily repeatable, allowing the user to reuse code to deploy similar, yet different resources

Infrastructure as Code easily replaces development languages such as Go and .Net for application development

Correct selection

Infrastructure as Code provides configuration consistency and standardization among deployments

Correct selection

Infrastructure as Code is relatively simple to learn and write, regardless of a user's prior experience with developing code

Overall explanation

Infrastructure as Code (IaC) refers to the practice of managing and provisioning infrastructure resources through code, rather than manual processes or user interfaces. Some of the benefits of using IaC include:

1. Consistency and repeatability: IaC allows for the creation of infrastructure in a consistent and repeatable way. This ensures that the same infrastructure can be deployed across multiple environments (e.g. development, testing, production) with minimal differences, reducing the risk of issues due to configuration drift or environment-specific issues.
2. Speed and agility: IaC allows for rapid provisioning and scaling of infrastructure resources, reducing the time it takes to set up and modify infrastructure. This enables teams to quickly respond to changing business needs or shifting workloads, without the delays associated with manual provisioning processes.
3. Version control: IaC code can be stored in version control systems, allowing teams to track changes over time and revert to previous versions if necessary. This provides a history of infrastructure changes and ensures that teams are always working with the most up-to-date version of the infrastructure code.
4. Collaboration and documentation: IaC code can be shared and collaborated on, allowing teams to work together to design and maintain infrastructure resources. It also provides a single source of truth for documentation, making it easier to understand how the infrastructure is configured and how it has changed over time.
5. Cost savings: IaC can help reduce infrastructure costs by allowing teams to more effectively manage resources, optimize usage, and avoid over-provisioning. It can also reduce the need for manual intervention, which can save time and reduce the risk of errors.

Overall, using IaC can help organizations achieve greater consistency, speed, agility, collaboration, and cost savings in their infrastructure management practices.

https://developer.hashicorp.com/terraform/intro#infrastructure-as-code

Domain

Objective 1 - Understand Infrastructure as Code concepts

Question 30Skipped

Select two answers to complete the following sentence:

Before a new provider can be used, it must be _____ and _____. (select two)

Correct selection

initialized

Correct selection

declared or used in a configuration file

approved by HashiCorp

uploaded to source control

Overall explanation

Each time a new provider is added to configuration -- either explicitly via a provider block or by adding a resource from that provider -- Terraform must initialize the provider before it can be used. Initialization downloads and installs the provider's plugin so that it can later be executed.

https://developer.hashicorp.com/terraform/language/providers/requirements#provider-installation

Domain

Objective 3 - Understand Terraform Basics

Question 31Skipped

When using variables in Terraform Cloud, what level of scope can the variable be applied to? (select three)

All workspaces across multiple Terraform Cloud organizations

Correct selection

Multiple workspaces using a variable set

Correct selection

A specific Terraform run in a single workspace

Correct selection

All current and future workspaces in a project using a variable set

Overall explanation

Terraform Cloud allows you to store important values in one place, which you can use across multiple projects. You can easily update the values, and the changes will apply to all projects that use them. Additionally, you can modify the values for specific projects without affecting others that use the same values. TFC allows you to use variables within a workspace, or use variable sets that can be used across multiple (or all) TFC workspaces.

Run-specific variables can be used by setting Terraform variable values using the -var and -var-file arguments in a single workspace

You can create a variable set by adding variables to the variable set and then applying a variable set scope so it can be used by ***multiple TFC workspaces***

You can also apply the variable set globally which will ***apply the variable set to all existing and future workspaces***

Wrong Answer:

Variable sets are constrained to a single organization. You can't create variable sets that can be used across multiple TFC organizations.

https://developer.hashicorp.com/terraform/cloud-docs/workspaces/variables

Domain

Objective 9 - Understand Terraform Cloud Capabilities

Question 32Skipped

What happens when a terraform plan is executed?

reconciles the state Terraform knows about with the real-world infrastructure

Correct answer

creates an execution plan and determines what changes are required to achieve the desired state in the configuration files.

the backend is initialized and the working directory is prepped

applies the changes required in the target infrastructure in order to reach the desired configuration

Overall explanation

The terraform plan command is used to create an execution plan. Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files.

After a plan has been run, it can be executed by running a terraform apply

https://developer.hashicorp.com/terraform/cli/commands/plan

Domain

Objective 6 - Use the Core Terraform Workflow

Question 33Skipped

Which of the following describes the process of leveraging a local value within a Terraform module and exporting it for use by another module?

Correct answer

Exporting the local value as an output from the first module, then importing it into the second module's configuration.

Explanation

This choice correctly describes the process of exporting a local value from one Terraform module by defining it as an output and then importing it into another module's configuration. This allows for the sharing of values between modules.

Using Terraform's built-in cross-module referencing feature to automatically share local values between modules.

Explanation

This choice is incorrect as Terraform does not have a built-in cross-module referencing feature to automatically share local values between modules. Exporting and importing values through outputs and configurations is the standard way to achieve this in Terraform.

Importing the local value directly into the second module's configuration.

Explanation

This choice is incorrect as importing a local value directly into another module's configuration is not a standard practice in Terraform. Exporting the value as an output and then importing it is the recommended approach for sharing values between modules.

Defining the local value in the first module, then passing it as an argument to the second module.

Explanation

This choice is incorrect as passing a local value as an argument to another module does not involve exporting and importing the value. It is a different method of sharing values between modules in Terraform.

Overall explanation

To use a local value in one Terraform module and make it available to another, you can define the local value within the first module and then export it as an output. The exported value can then be imported into the configuration of the second module. This approach establishes a clear pathway for sharing data between modules, enabling efficient and modular infrastructure management within your Terraform configurations.

https://developer.hashicorp.com/terraform/language/values/locals

Domain

Objective 3 - Understand Terraform Basics

Question 34Skipped

What is the downside to using Terraform to interact with sensitive data, such as reading secrets from Vault?

Terraform requires a unique auth method to work with Vault

Correct answer

secrets are persisted to the state file

Terraform and Vault must be running on the same physical host

Terraform and Vault must be running on the same version

Overall explanation

Interacting with Vault from Terraform causes any secrets that you read and write to be persisted in both Terraform's state file *and* in any generated plan files. For any Terraform module that reads or writes Vault secrets, these files should be treated as sensitive and protected accordingly.

https://registry.terraform.io/providers/hashicorp/vault/latest/docs

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 35Skipped

True or False? Multiple providers can be declared within a single Terraform configuration file.

Correct answer

True

False

Overall explanation

True. Multiple providers can be declared within a single Terraform configuration file. In fact, it is common to declare multiple providers within a single configuration file, particularly when managing resources across multiple cloud providers.

When declaring multiple providers within a single configuration file, each provider should have a unique configuration block that specifies its name, source, and any required settings or credentials. Here's an example of what a configuration block for two different providers might look like:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
    }
    google = {
      source = "hashicorp/google"
    }
  }
```

```
}

provider "aws" {
  region = "us-west-2"
  access_key = "ACCESS_KEY"
  secret_key = "SECRET_KEY"
}

provider "google" {
  project = "my-project"
  credentials = file("path/to/credentials.json")
}
```

In this example, we have declared two providers (**aws** and **google**) within a single configuration file. The **terraform** block declares the required providers, while the **provider** blocks specify the provider-specific settings and credentials.

When executing Terraform commands, you can use the **-target** option to specify which provider you want to apply changes to. For example, you could apply changes to the AWS provider by running **terraform apply -target=aws**.

https://developer.hashicorp.com/terraform/language/providers/configuration

Domain

Objective 3 - Understand Terraform Basics

Question 36Skipped

By default, where does Terraform OSS/CLI store its state file?

Amazon S3 bucket

Correct answer

current working directory

shared directory

remotely using Terraform Cloud

Overall explanation

By default, the state file is stored in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.

https://developer.hashicorp.com/terraform/language/settings/backends/configuration

Domain

Objective 7 - Implement and Maintain State

Question 37Skipped

True or False? State is a requirement for Terraform to function.

False

Correct answer

True

Overall explanation

True.

State is a fundamental concept in Terraform that keeps track of the resources Terraform manages, their configuration, and their current state. Terraform uses this information to determine the differences between the desired state and the current state and to generate a plan for creating, updating, or deleting resources to match the desired state.

The state file is a critical component of Terraform and is required for its proper functioning. It is typically stored remotely in a shared location, such as a storage service or version control system, to allow multiple members of a team to collaborate on infrastructure changes.

https://developer.hashicorp.com/terraform/language/state/purpose

Domain

Objective 2 - Understand Terraform's purpose (vs. other IaC)

Question 38Skipped

When multiple engineers start deploying infrastructure using the same state file, what is a feature of remote state storage that is critical to ensure the state does not become corrupt?

Correct answer

state locking

object storage

encryption

workspaces

Overall explanation

If supported by your backend, Terraform will lock your state for all operations that could write state. This prevents others from acquiring the lock and potentially corrupting your state.

State locking happens automatically on all operations that could write state. You won't see any message that it is happening. If state locking fails, Terraform will not continue. You can disable state locking for most commands with the -lock flag but it is not recommended.

https://developer.hashicorp.com/terraform/language/state/locking

Domain

Objective 7 - Implement and Maintain State

Question 39Skipped

Published modules via the Terraform Registry provide which of the following benefits? (select four)

Correct selection

automatically generated documentation

support from any code repo

Correct selection

allow browsing version histories

Correct selection

show examples and READMEs

Correct selection

support versioning

Overall explanation

Public modules are managed via ***Git and GitHub***. Publishing a module takes only a few minutes. Once a module is published, you can release a new version of a module by simply pushing a properly formed Git tag. The module must be on GitHub and must be a public repo. This is only a requirement for the public registry. If you're using a private registry, you may ignore this requirement.

The key here is that HashiCorp uses **GitHub** for published modules.

Domain

Objective 5 - Interact with Terraform Modules

Question 40Skipped

Which configuration block type is used to declare settings and behaviors specific to Terraform?

data block

Correct answer

terraform block

resource block

provider block

Overall explanation

In Terraform, the **terraform** block is used to configure Terraform settings and to specify a required version constraint for the Terraform CLI.

The **terraform** block is optional and is typically placed at the top of a Terraform configuration file. Here is an example of a **terraform** block:

```
terraform {
  required_version = ">= 0.12.0, < 0.13.0"
  backend "s3" {
    bucket = "my-terraform-state"
    key    = "terraform.tfstate"
    region = "us-west-2"
  }
}
```

In this example, the **terraform** block specifies that the Terraform configuration requires a version of at least 0.12.0 but less than 0.13.0. The block also contains a **backend** block, which configures the backend where the Terraform state is stored. In this case, the backend is an S3 bucket in the **us-west-2** region.

The **terraform** block can also be used to configure other settings such as the maximum number of concurrent operations (**max_parallelism**), the number of retries for failed operations (**retryable_errors**), and the default input values for variables (**default**).

By including a **terraform** block in the Terraform configuration, you can ensure that the correct version of Terraform is used and that the configuration is validated against the correct syntax and semantics for that version. This helps to ensure that the configuration will run correctly and consistently across different environments.

https://developer.hashicorp.com/terraform/language/settings

Domain

Objective 3 - Understand Terraform Basics

Question 41Skipped

What Terraform feature is shown in the example below?

```
resource "aws_security_group" "example" {
  name = "sg-app-web-01"

  dynamic "ingress" {
```

```
5.      for_each = var.service_ports
6.      content {
7.       from_port = ingress.value
8.       to_port   = ingress.value
9.       protocol  = "tcp"
10.    }
11.  }
12. }
```

conditional expression

Correct answer

dynamic block

local values

data source

Overall explanation

You can dynamically construct repeatable nested blocks like ingress using a special dynamic block type, which is supported inside resource, data, provider, and provisioner blocks.

https://developer.hashicorp.com/terraform/language/expressions#dynamic-blocks

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 42Skipped

Rick is writing a new Terraform configuration file and wishes to use modules in order to easily consume Terraform code that has already been written. Which of the modules shown below will be created first?

```
1.  terraform {
2.    required_providers {
3.      aws = {
4.       source = "hashicorp/aws"
5.     }
6.    }
7.  }
8.
```

```
provider "aws" {
 region = "us-west-2"
}

module "vpc" {
 source  = "terraform-aws-modules/vpc/aws"
 version = "2.21.0"

 name = var.vpc_name
 cidr = var.vpc_cidr

 azs           = var.vpc_azs
 private_subnets = var.vpc_private_subnets
 public_subnets  = var.vpc_public_subnets

 enable_nat_gateway = var.vpc_enable_nat_gateway

 tags = var.vpc_tags
}

module "ec2_instances" {
 source  = "terraform-aws-modules/ec2-instance/aws"
 version = "2.12.0"

 name         = "my-ec2-cluster"
 instance_count = 2

 ami                 = "ami-0c5204531f799e0c6"
 instance_type        = "t2.micro"
 vpc_security_group_ids = [module.vpc.default_security_group_id]
 subnet_id           = module.vpc.public_subnets[0]
```

40.

41. tags = {

42. Terraform = "true"

43. Environment = "dev"

44. }

45. }

Correct answer

1. module "vpc"

1. module "ec2_instances"

Overall explanation

The VPC module will be executed first since the ec2_instances module has dependencies on the VPC module. Both vpc_security_group_ids and subnet_id require outputs from the VPC module.

https://learn.hashicorp.com/tutorials/terraform/module-use

Domain

Objective 5 - Interact with Terraform Modules

Question 43Skipped

What are some of the features of Terraform state? (select three)

Correct selection

increased performance

inspection of cloud resources

Correct selection

mapping configuration to real-world resources

Correct selection

determining the correct order to destroy resources

Overall explanation

See this page on the purpose of Terraform state and the benefits it provides.

Domain

Objective 7 - Implement and Maintain State

Question 44Skipped

When configuring a remote backend in Terraform, it might be a good idea to purposely omit some of the required arguments to ensure secrets and other relevant data are not inadvertently shared with others. What alternatives are available to provide the remaining values to Terraform to initialize and communicate with the remote backend? (select three)

directly querying HashiCorp Vault for the secrets

Correct selection

interactively on the command line

Correct selection

command-line key/value pairs

Correct selection

use the -backend-config=PATH flag to specify a separate config file

Overall explanation

You do not need to specify every required argument in the backend configuration. Omitting certain arguments may be desirable to avoid storing secrets, such as access keys, within the main configuration. When some or all of the arguments are omitted, we call this a *partial configuration*.

With a partial configuration, the remaining configuration arguments must be provided as part of the initialization process. There are several ways to supply the remaining arguments:

Interactively: Terraform will interactively ask you for the required values unless interactive input is disabled. Terraform will not prompt for optional values.

File: A configuration file may be specified via the init command line. To specify a file, use the -backend-config=PATH option when running terraform init. If the file contains secrets it may be kept in a secure data store, such as Vault, in which case it must be downloaded to the local disk before running Terraform.

Command-line key/value pairs: Key/value pairs can be specified via the init command line. Note that many shells retain command-line flags in a history file, so this isn't recommended for secrets. To specify a single key/value pair, use the -backend-config="KEY=VALUE" option when running terraform init.

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 45Skipped

Select the answer below that completes the following statement:

Terraform Cloud can be managed from the CLI but requires _____?

a TOTP token

a username and password

Correct answer

an API token

authentication using MFA

Overall explanation

API and CLI access are managed with API tokens, which can be generated in the Terraform Cloud UI. Each user can generate any number of personal API tokens, which allow access with their own identity and permissions. Organizations and teams can also generate tokens for automating tasks that aren't tied to an individual user.

https://developer.hashicorp.com/terraform/cloud-docs/users-teams-organizations/api-tokens

Domain

Objective 9 - Understand Terraform Cloud Capabilities

Question 46Skipped

By default, a child module will have access to all variables set in the calling (parent) module.

Correct answer

False

Explanation

This choice is correct because, by default, a child module in Terraform does not inherit all variables set in the calling (parent) module. The parent module must explicitly pass variables to the child module by defining input variables in the child module's configuration. This approach helps maintain clear boundaries between modules and prevents unintended variable leakage.

True

Explanation

By default, a child module does not have access to all variables set in the calling (parent) module. The child module must explicitly declare input variables to receive values from the parent module. This ensures that only the necessary variables are passed between modules, promoting modularity and encapsulation.

Overall explanation

A Terraform module (usually the root module of a configuration) can *call* other modules to include their resources into the configuration. A module that has been called by another module is often referred to as a *child module*.

When you declare variables in the root module of your configuration, you can set their values using CLI options and environment variables. When you declare them in child modules, the calling module should pass values in the module block.

Example of a module block that has multiple variables passed to it:

```
1. module "server" {
2.   source       = "./modules/server"
```

3. ami = data.aws_ami.ubuntu.id
4. size = "t2.micro"
5. subnet_id = aws_subnet.public_subnets["public_subnet_3"].id
6. security_groups = [aws_security_group.vpc-ping.id, aws_security_group.ingress-ssh.id, aws_security_group.vpc-web.id]
7. }

https://developer.hashicorp.com/terraform/language/modules/develop

Domain

Objective 5 - Interact with Terraform Modules

Question 47Skipped

Given the Terraform configuration below, which order will the resources be created?

1. resource "aws_instance" "web_server" {
2. ami = "i-abdce12345"
3. instance_type = "t2.micro"
4. }
5.
6. resource "aws_eip" "web_server_ip" {
7. vpc = true
8. instance = aws_instance.web_server.id
9. }

no resources will be created

aws_eip will be created first

aws_instance will be created second

Correct answer

aws_instance will be created first

aws_eip will be created second

resources will be created in parallel

Overall explanation

The aws_instance will be created first, and then aws_eip will be created second due to the aws_eip's resource dependency of the aws_instance id

More information on resource dependencies can be found at this link.

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 48Skipped

Stephen is writing brand new code and needs to ensure it is syntactically valid and internally consistent. Stephen doesn't want to wait for Terraform to access any remote services while making sure his code is valid. *What command can he use to accomplish this?*

terraform show

Correct answer

terraform validate

terraform fmt

terraform apply -refresh-only

Overall explanation

The terraform validate command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

Validate runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state. It is thus primarily useful for general verification of reusable modules, including correctness of attribute names and value types.

https://developer.hashicorp.com/terraform/cli/commands/validate

Domain

Objective 6 - Use the Core Terraform Workflow

Question 49Skipped

In the example below, where is the value of the DNS record's IP address originating from?

```
1. resource "aws_route53_record" "www" {
2.   zone_id = aws_route53_zone.primary.zone_id
3.   name    = "www.helloworld.com"
4.   type    = "A"
5.   ttl     = "300"
6.   records = [module.web_server.instance_ip_addr]
7. }
```

value of the web_server parameter from the variables.tf file

by querying the AWS EC2 API to retrieve the IP address

Correct answer

the output of a module named web_server

the regular expression named module.web_server

Overall explanation

In a parent module, outputs of child modules are available in expressions as module.<MODULE NAME>.<OUTPUT NAME>. For example, if a child module named web_server declared an output named instance_ip_addr, you could access that value as module.web_server.instance_ip_addr.

https://developer.hashicorp.com/terraform/language/expressions#references-to-named-values

Domain

Objective 8 - Read, Generate, and Modify Configuration

Question 50Skipped

Frank has a file named main.tf which is shown below. Which of the following statements are true about this code? (select two)

```
1. module "servers" {
2.   source = "./app-cluster"
3.
4.   servers = 5
5. }
```

Correct selection

main.tf is the calling module

main.tf is the child module

Correct selection

app-cluster is the child module

app-cluster is the calling module

Overall explanation

To *call* a module means to include the contents of that module into the configuration with specific values for its input variables. Modules are called from within other modules using module blocks. A module that includes a module block like this is the *calling module* of the child module.

The label immediately after the module keyword is a local name, which the calling module can use to refer to this instance of the module.

https://developer.hashicorp.com/terraform/language/modules#calling-a-child-module

Domain

Objective 5 - Interact with Terraform Modules

Question 51Skipped

True or False? Similar to Terraform OSS, you must use the CLI to switch between workspaces when using Terraform Cloud workspaces.

True

Correct answer

False

Overall explanation

False.

When using Terraform Cloud workspaces, you do not need to use the Terraform CLI to switch between workspaces. Terraform Cloud provides a web-based interface where you can manage your workspaces and their associated infrastructure.

In Terraform Cloud, each workspace represents a separate environment (e.g., development, staging, production), and you can view and manage them individually through the Terraform Cloud web UI. You can select a workspace from the workspace switcher in the web interface to make changes to the infrastructure associated with that workspace directly.

The Terraform CLI is primarily used for running Terraform commands locally on your development machine. When working with Terraform Cloud, you typically interact with your workspaces through the web UI or by using Terraform Cloud's API.

The Terraform CLI does have a command (`terraform workspace`) to manage workspaces when using the local backend, but it's not necessary when using Terraform Cloud as your backend, as the web interface handles workspace management for you.

https://developer.hashicorp.com/terraform/cloud-docs/workspaces

Domain

Objective 9 - Understand Terraform Cloud Capabilities

Question 52Skipped

What happens when a terraform apply command is executed?

reconciles the state Terraform knows about with the real-world infrastructure

Correct answer

applies the changes required in the target infrastructure in order to reach the desired configuration

the backend is initialized and the working directory is prepped

creates the execution plan for the deployment of resources

Overall explanation

The terraform apply command is used to apply the changes required to reach the desired state of the configuration, or the pre-determined set of actions generated by a terraform plan execution plan.

https://developer.hashicorp.com/terraform/cli/commands/apply

Domain

Objective 6 - Use the Core Terraform Workflow

Question 53Skipped

Which Terraform command will force a resource to be destroyed and recreated even if there are no configuration changes that would require it?

terraform apply -refresh-only

terraform destroy

Correct answer

terraform apply -replace=<address>

terraform fmt

Overall explanation

The **terraform apply -replace=<address>** command manually marks a Terraform-managed resource to be replaced, forcing it to be destroyed and recreated during the **apply**. Even if there are no configuration changes that would require a change or deletion of this resource, this command will instruct Terraform to replace it. This can come in handy if a resource has become degraded or damaged outside of Terraform.

IMPORTANT - PLEASE READ

This command replaces **terraform taint**, and it's possible you may still see **terraform taint** on the exam. Be prepared to know both of these commands.

https://developer.hashicorp.com/terraform/cli/commands/taint

Domain

Objective 4 - Use Terraform Outside of Core Workflow

Question 54Skipped

From the answers below, select the advantages of using Infrastructure as Code. (select four)

Correct selection

Easily change and update existing infrastructure

Provide a codified workflow to develop customer-facing applications

Correct selection

Safely test modifications using a "dry run" before applying any actual changes

Correct selection

Easily integrate with application workflows (GitHub Actions, Azure DevOps, CI/CD tools)

Correct selection

Provide reusable modules for easy sharing and collaboration

Overall explanation

Infrastructure as Code is **not** used to develop applications, but it can be used to help deploy or provision those applications to a public cloud provider or on-premises infrastructure.

All of the others are benefits to using Infrastructure as Code over the traditional way of managing infrastructure, regardless if it's public cloud or on-premises.

https://developer.hashicorp.com/terraform/intro

Domain

Objective 1 - Understand Infrastructure as Code concepts

Question 55Skipped

Which of the following allows Terraform users to apply policy as code to enforce standardized configurations for resources being deployed via infrastructure as code?

private

functions

Correct answer

sentinel

workspaces

Overall explanation

Sentinel is an embedded policy-as-code framework integrated with the HashiCorp Enterprise products. It enables fine-grained, logic-based policy decisions, and can be extended to use information from external sources.

https://www.hashicorp.com/sentinel

Domain

Objective 9 - Understand Terraform Cloud Capabilities

Question 56Skipped

What advantages does Terraform offer over using a provider's native tooling for deploying resources in multi-cloud environments? (select three)

Correct selection

Terraform can manage cross-cloud dependencies

Terraform is not cloud-agnostic and can only be used to deploy resources across a single public cloud at a time

Correct selection

Terraform simplifies management and orchestration, helping operators build large-scale, multi-cloud infrastructure

Correct selection

Terraform can help businesses deploy applications on multiple clouds and on-premises infrastructure

Overall explanation

Terraform offers several advantages over using a provider's native tooling for deploying resources in multi-cloud environments, including:

1. **Multi-cloud support:** Terraform provides a consistent interface for managing infrastructure resources across multiple cloud providers, including AWS, Azure, Google Cloud, and more. This allows organizations to use a single tool for managing their entire multi-cloud environment rather than needing to learn and use multiple provider-specific tools.
2. **Standardized configuration:** Terraform uses a declarative configuration language to define infrastructure resources, which can be used to define resources across multiple providers in a standardized way. This provides consistency and reduces the need for provider-specific knowledge.
3. **Idempotent execution:** Terraform only makes changes to infrastructure resources if the desired state differs from the current state, which means that it can be safely run multiple times without causing unintended changes. This reduces the risk of configuration drift and ensures that infrastructure remains consistent over time.
4. **Plan preview:** Terraform can generate a plan that shows the changes it will make to infrastructure resources before it applies them. This provides visibility into changes and helps to reduce the risk of unintended consequences.
5. **Collaboration and version control:** Terraform configurations can be stored in version control systems, allowing multiple team members to collaborate on infrastructure changes. This provides a centralized location for documentation, change history, and issue tracking, making it easier to manage infrastructure changes over time.

Overall, using Terraform for deploying resources in multi-cloud environments can provide a consistent, standardized, and efficient approach to managing infrastructure across multiple providers.

https://developer.hashicorp.com/terraform/intro/use-cases#multi-cloud-deployment

Domain

Objective 2 - Understand Terraform's purpose (vs. other IaC)

Question 57Skipped

Which of the following Terraform files should be ignored by Git when committing code to a repo? (select two)

Correct selection

terraform.tfstate

variables.tf

Correct selection

terraform.tfvars

outputs.tf

Overall explanation

When using Terraform with Git, it is generally recommended to ignore certain files in order to avoid committing sensitive or unnecessary information to your repository. The specific files that should be ignored may vary depending on your project and configuration, but as a general rule, you should ignore the following files:

1. **.terraform** directory: This directory contains local Terraform state files, which should not be committed to the repository.
2. **terraform.tfstate** and **terraform.tfstate.backup**: These files contain the current state of your infrastructure, and should not be committed to the repository.
3. **.tfvars** files: These files may contain sensitive information, such as passwords or API keys, and should be kept out of version control. Instead, you can use environment variables or other secure methods to pass this information to Terraform.
4. ***.tfplan** files: These files contain the plan generated by Terraform when applying changes to your infrastructure, and may include sensitive information such as resource IDs. They should not be committed to the repository.

To ignore these files in Git, you can add them to your **.gitignore** file.

https://github.com/github/gitignore/blob/main/Terraform.gitignore

https://www.hashicorp.com/resources/a-practitioner-s-guide-to-using-hashicorp-terraform-cloud-with-github#:~:text=Gitignore%20Considerations

Domain

Objective 3 - Understand Terraform Basics