HashiCorp TA-002-P Practice Test Pdf Exam Material [Q117-Q135

HashiCorp TA-002-P Practice Test Pdf Exam Material [Q117-Q135]
SUCCESS

HashiCorp TA-002-P Practice Test Pdf Exam Material
TA-002-P Answers TA-002-P Free Demo Are Based On The Real Exam

HashiCorp TA-002-P Exam Syllabus Topics:

TopicDetailsTopic 1- Given a scenario: choose when to use terraform state to view Terraform state- Describe plugin based architectureTopic 2- Describe the benefits of Sentinel, registry, and workspaces- Describe variable scope within modules-child modulesTopic 3- Given a scenario: choose when to use terraform taint to taint Terraform resources- Handle backend authentication methodsTopic 4- Create and differentiate resource and data configuration- Understand the use of collection and structural typesTopic 5- Execute changes to infrastructure with Terraform (terraform apply)- Handle Terraform and provider installation and versioningTopic 6- Explain multi-cloud and provider-agnostic benefits- Understand infrastructure as code (IaC) conceptsTopic 7- Destroy Terraform managed infrastructure (terraform destroy)- Given a scenario: choose when to use terraform fmt to format codeTopic 8- Describe secure secret injection best practice- Use resource addressing and resource parameters to connect resources togetherTopic 9- Given a scenario: choose when to use terraform import to import existing infrastructure into your Terraform state- Understand Terraform basicsTopic 10- Explain when to use and not use provisioners and when to use local-exec or remote-exec- Describe advantages of IaC patternsTopic 11- Describe backend block in configuration and best practices for partial configurations- Demonstrate using multiple providers

NO.117 Which of the following clouds does not have a provider maintained HashiCorp?
* IBM Cloud
* DigitalOcean
* OpenStack
* AWS

IBM Cloud does not have a provider maintained by HashiCorp, although IBM Cloud does maintain their own Terraform provider.

https://www.terraform.io/docs/providers/index.html

NO.118 Which of the following best describes a Terraform provider?
* A plugin that Terraform uses to translate the API interactions with the service or provider.
* Serves as a parameter for a Terraform module that allows a module to be customized.
* Describes an infrastructure object, such as a virtual network, compute instance, or other components.
* A container for multiple resources that are used together.

A provider is responsible for understanding API interactions and exposing resources. Providers generally are an IaaS (e.g. Alibaba Cloud, AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Cloud, DNSimple, Cloudflare).

https://www.terraform.io/docs/providers/index.html

NO.119 Which task does terraform init not perform?
* Sources all providers present in the configuration and ensures they are downloaded and available locally
* Connects to the backend
* Sources any modules and copies the configuration locally
* Validates all required variables are present

Reference: https://www.terraform.io/docs/cli/commands/init.html

NO.120 Which backend does the Terraform CLI use by default?
* Terraform Cloud
* Consul
* Remote
* Local

NO.121 Setting the TF_LOG environment variable to DEBUG causes debug messages to be logged into syslog.
* True
* False

NO.122 When does terraform apply reflect changes in the cloud environment?
* Immediately
* However long it takes the resource provider to fulfill the request
* After updating the state file
* Based on the value provided to the -refresh command line argument
* None of the above

NO.123 What are some of the features of Terraform state? (select three)
* inspection of cloud resources
* determining the correct order to destroy resources
* mapping configuration to real-world resources
* increased performance

NO.124 How is terraform import run?
* As a part of terraform init
* As a part of terraform plan
* As a part of terraform refresh
* By an explicit call
* All of the above

NO.125 Which of the following terraform subcommands could be used to remove the lock on the state for the current configuration?
* Unlock
* force-unlock
* Removing the lock on a state file is not possible
* state-unlock

Explanation

https://www.terraform.io/docs/commands/force-unlock.html

NO.126 Mary has created a database instance in AWS and for ease of use is outputting the value of the database password with the following code:

1. output &#8220;db_password&#8221;

2. {

3. value = local.db_password

4. }

Mary wants to hide the output value in the CLI after terraform apply? What is the best way?
* Use secure parameter
* Use sensitive parameter
* Use cryptographic hash
* Encrypt the value using encrypt() function

NO.127 You have created a terraform script that uses a lot of new constructs that have been introduced in terraform v0.12. However, many developers who are cloning the script from your git repo, are using v0.11, and getting errors. What can be done from your end to solve this problem?
* Force developer to use v0.12 by using terraform setting &#8216;required_version&#8217; and set it to >=0.12.
* Refactor the code to support both v0.11, and v0.12. It might be a difficult process, but there is no other way.
* Add a condition in front of each such specific construct, to check whether the running terraform version id v0.11 or v0.12, and ,work accordingly.
* Add comments in your code to tell developers to use v0.12 . If they use v0.11 , that should be their problem , which they need to figure out.
https://www.terraform.io/docs/configuration/terraform.html

NO.128 You want terraform plan and terraform apply to be executed in Terraform Cloud&#8217;s run environment but the output is to be streamed locally. Which one of the below you will choose?
* Local Backends.
* Terraform Backends.

* This can be done using any of the local or remote backends.
* Remote Backends.

When using full remote operations, operations like terraform plan or terraform apply can be executed in Terraform Cloud&#8217;s run environment, with log output streaming to the local terminal. Remote plans and applies use variable values from the associated Terraform Cloud workspace.

Terraform Cloud can also be used with local operations, in which case only state is stored in the Terraform Cloud backend.

https://www.terraform.io/docs/backends/types/remote.html

NO.129 State is a requirement for Terraform to function
* True
* False

Explanation

State is a necessary requirement for Terraform to function. It is often asked if it is possible for Terraform to work without state, or for Terraform to not use state and just inspect cloud resources on every run.

Purpose of Terraform State

State is a necessary requirement for Terraform to function. It is often asked if it is possible for Terraform to work without state, or for Terraform to not use state and just inspect cloud resources on every run. This page will help explain why Terraform state is required.

As you&#8217;ll see from the reasons below, state is required. And in the scenarios where Terraform may be able to get away without state, doing so would require shifting massive amounts of complexity from one place (state) to another place (the replacement concept).

1. Mapping to the Real World

Terraform requires some sort of database to map Terraform config to the real world. When you have a resource resource &#8220;aws_instance&#8221; &#8220;foo&#8221; in your configuration, Terraform uses this map to know that instance i-abcd1234 is represented by that resource.

For some providers like AWS, Terraform could theoretically use something like AWS tags. Early prototypes of Terraform actually had no state files and used this method. However, we quickly ran into problems. The first major issue was a simple one: not all resources support tags, and not all cloud providers support tags.

Therefore, for mapping configuration to resources in the real world, Terraform uses its own state structure.

2. Metadata

Alongside the mappings between resources and remote objects, Terraform must also track metadata such as resource dependencies.

Terraform typically uses the configuration to determine dependency order. However, when you delete a resource from a Terraform configuration, Terraform must know how to delete that resource. Terraform can see that a mapping exists for a resource not in your configuration and plan to destroy. However, since the configuration no longer exists, the order cannot be determined from the configuration alone.

To ensure correct operation, Terraform retains a copy of the most recent set of dependencies within the state.

Now Terraform can still determine the correct order for destruction from the state when you delete one or more items from the configuration.

One way to avoid this would be for Terraform to know a required ordering between resource types. For example, Terraform could know that servers must be deleted before the subnets they are a part of. The complexity for this approach quickly explodes, however: in addition to Terraform having to understand the ordering semantics of every resource for every cloud, Terraform must also understand the ordering across providers.

Terraform also stores other metadata for similar reasons, such as a pointer to the provider configuration that was most recently used with the resource in situations where multiple aliased providers are present.

3. Performance

In addition to basic mapping, Terraform stores a cache of the attribute values for all resources in the state. This is the most optional feature of Terraform state and is done only as a performance improvement.

When running a terraform plan, Terraform must know the current state of resources in order to effectively determine the changes that it needs to make to reach your desired configuration.

For small infrastructures, Terraform can query your providers and sync the latest attributes from all your resources. This is the default behavior of Terraform: for every plan and apply, Terraform will sync all resources in your state.

For larger infrastructures, querying every resource is too slow. Many cloud providers do not provide APIs to query multiple resources at once, and the round trip time for each resource is hundreds of milliseconds. On top of this, cloud providers almost always have API rate limiting so Terraform can only request a certain number of resources in a period of time. Larger users of Terraform make heavy use of the -refresh=false flag as well as the -target flag in order to work around this. In these scenarios, the cached state is treated as the record of truth.

4. Syncing

In the default configuration, Terraform stores the state in a file in the current working directory where Terraform was run. This is okay for getting started, but when using Terraform in a team it is important for everyone to be working with the same state so that operations will be applied to the same remote objects.

Remote state is the recommended solution to this problem. With a fully-featured state backend, Terraform can use remote locking as a measure to avoid two or more different users accidentally running Terraform at the same time, and thus ensure that each Terraform run begins with the most recent updated state.

NO.130 I78

correct?
* When using local state, the state file is stored in plain-text.
* The state file is always encrypted at rest.
* Storing state remotely can provide better security.
* Using the mask feature, you can instruct Terraform to mask sensitive data in the state file.
* The Terraform state can contain sensitive data, therefore the state file should be protected from unauthorized access.
* Terraform Cloud always encrypts state at rest.

Terraform state can contain sensitive data, depending on the resources in use and your definition of &#8220;sensitive.&#8221; 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.

When using remote state, state is only ever held in memory when used by Terraform. It may be encrypted at rest, but this depends on the specific remote state backend.

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.

Recommendations

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

Storing 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.

For example:

* Terraform Cloud always encrypts state at rest and protects it with TLS in transit. Terraform Cloud also knows the identity of the user requesting state and maintains a history of state changes. This can be used to control access and track activity. Terraform Enterprise also supports detailed audit logging.

* The S3 backend supports encryption at rest when the encrypt option is enabled. IAM policies and logging can be used to identify any invalid access. Requests for the state go over a TLS connection.

NO.131 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 all existing is managed by Terraform moving forward without interruption to existing services?
* 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.
* Delete the existing resources and recreate them using new a Terraform configuration so Terraform can manage them moving forward.
* Resources that are manually deployed in the AWS console cannot be imported by Terraform.
* Using terraform import, import the existing infrastructure into your Terraform state.
Explanation

Terraform is able to import existing infrastructure. This allows us take resources we&#8217;ve created by some other means (i.e. via console) and bring it under Terraform management.

This is a great way to slowly transition infrastructure to Terraform.

The terraform import command is used to import existing infrastructure.

To import a resource, first write a resource block for it in our configuration, establishing the name by which it will be known to Terraform.

Example:

resource &#8220;aws_instance&#8221; &#8220;import_example&#8221; {

&#8230;instance configuration&#8230;

}

Now terraform import can be run to attach an existing instance to this resource configuration.

$ terraform import aws_instance.import_example i-03efafa258104165f

aws_instance.import_example: Importing from ID &#8220;i-03efafa258104165f&#8221;&#8230;

aws_instance.import_example: Import complete!

Imported aws_instance (ID: i-03efafa258104165f)

aws_instance.import_example: Refreshing state&#8230; (ID: i-03efafa258104165f) Import successful!

The resources that were imported are shown above. These resources are now in your Terraform state and will henceforth be managed by Terraform.

This command locates the AWS instance with ID i-03efafa258104165f (which has been created outside Terraform) and attaches its existing settings, as described by the EC2 API, to the name aws_instance.import_example in the Terraform state.

NO.132 Refer to the below code where developer is outputting the value of the database password but has used sensitive parameter to hide the output value in the CLI.

output &#8220;db_password&#8221; { value = aws_db_instance.db.password description = &#8220;The password for logging in to the database.&#8221; sensitive = true} Since sensitive is set to true, the value associated with db password will not be present in state file as plain-text?
* False
* True
Sensitive output values are still recorded in the state, and so will be visible to anyone who is able to access the state data.

NO.133 What is the name of the default file where Terraform stores the state?

Type your answer in the field provided. The text field is not case-sensitive and all variations of the correct answer are accepted.
Terraformtfstate

NO.134 A user runs terraform init on their RHEL based server and per the output, two provider plugins are downloaded: $ terraform init Initializing the backend&#8230;

Initializing provider plugins&#8230;

&#8211; Checking for available provider plugins&#8230;

&#8211; Downloading plugin for provider &#8220;aws&#8221; (hashicorp/aws) 2.44.0&#8230;

&#8211; Downloading plugin for provider &#8220;random&#8221; (hashicorp/random) 2.2.1&#8230;

:

Terraform has been successfully initialized! Where are these plugins downloaded to?
* The .terraform.plugins directory in the directory terraform init was executed in.
* The .terraform/plugins directory in the directory terraform init was executed in.
* /etc/terraform/plugins
* The .terraform.d directory in the directory terraform init was executed in.

NO.135 Open source Terraform can only import publicly-accessible and open-source modules.
* True
* False

Understanding functional and technical aspects of HashiCorp Certified: Terraform Associate TA-002-P Professional Exam Object Management

The following will be discussed in **HASHICORP TA-002 exam dumps**:

- Advantages of Sentinel, registry, and workspaces Differentiate OSS and TFE workspaces- Express secure secret injection best practice- Design and distinguish resource and data configuration- Exhibit use of variables and outputs- Use Terraform built-in functions to write configuration **TA-002-P [Jun-2022 Newly Released Exam Questions For You To Pass:** https://www.vceprep.com/TA-002-P-latest-vce-prep.html]