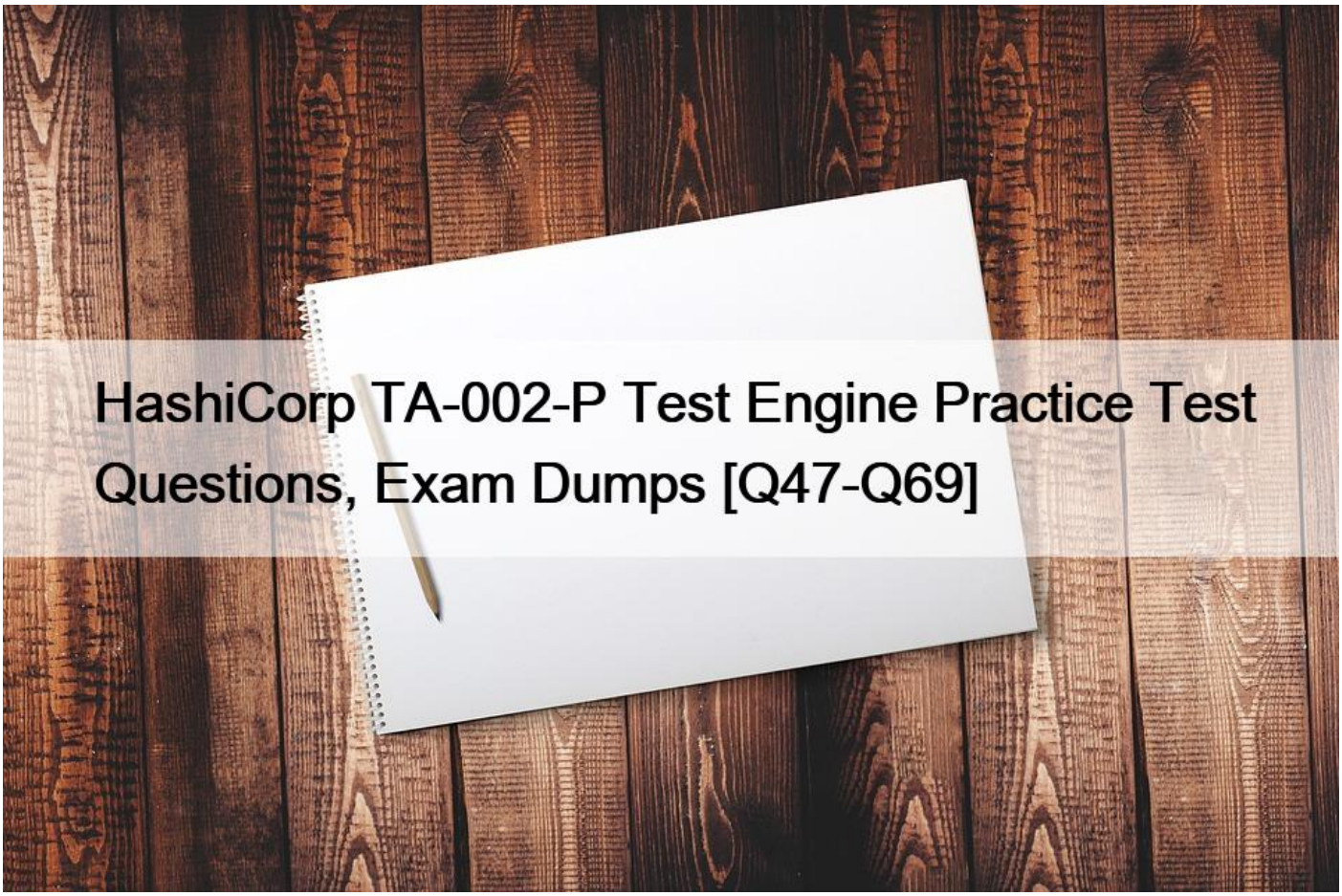
This page was exported from - Free valid test braindumps
Export date: Wed Apr 9 6:16:40 2025 / +0000 GMT

HashiCorp TA-002-P Test Engine Practice Test Questions, Exam Dumps [Q47-Q69



HashiCorp TA-002-P Test Engine Practice Test Questions, Exam Dumps
100% Free TA-002-P Daily Practice Exam With 94 Questions

HashiCorp TA-002-P (HashiCorp Certified: Terraform Associate) Certification Exam is a globally recognized certification that validates an individual's expertise in using Terraform, an open-source infrastructure as code (IaC) tool. HashiCorp Certified: Terraform Associate certification is designed for professionals who want to demonstrate their skills in managing and automating infrastructure using Terraform. It is an excellent way for IT professionals to enhance their career prospects and demonstrate their ability to work with cloud infrastructure.

NO.47 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

NO.48 Terraform and Terraform providers must use the same major version number in a single configuration.
* True
* False

NO.49 Which two steps are required to provision new infrastructure in the Terraform workflow? (Choose two.)
* Destroy
* Apply
* Import
* Init
* Validate

NO.50 Ric wants to enable detail logging and he wants highest verbosity of logs. Which of the following environment variable settings is correct option for him to select.
* Set TF_LOG = DEBUG
* Set VAR_TF = TRACE
* Set TF_LOG = TRACE
* Set VAR_TF_LOG = TRACE

Explanation

https://www.terraform.io/docs/internals/debugging.html

NO.51 1. resource “aws_s3_bucket” “example” {

2. bucket = “my-test-s3-terraform-bucket”

3. …} resource “aws_iam_role” “test_role” {

4. name = “test_role”

5. …}

Due to the way that the application code is written, the s3 bucket must be created before the test role is

created, otherwise there will be a problem. How can you ensure that?
* Add explicit dependency using depends_on . This will ensure the correct order of resource creation.
* This will already be taken care of by terraform native implicit dependency. Nothing else needs to be

done from your end.
* This is not possible to control in terraform . Terraform will take care of it in a native way , and create a

dependency graph that is best suited for the parallel resource creation.
* Create 2 separate terraform config scripts , and run them one by one , 1 for s3 bucket , and another for

IAM role , run the S3 bucket script first.

Explanation

Implicit dependency works only if there is some reference of one resource to another. Explicit dependency is

the option here.

NO.52 Which of the following command can be used to view the specified version constraints for all providers used

in the current configuration.
* terraform providers
* terraform state show
* terraform provider
* terraform plan

Explanation

Use the terraform providers command to view the specified version constraints for all providers used in the

current configuration.

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

NO.53 Your configuration file has been locked accidentally. What of the following command would you use to unlock?
* terraform filename-unlock
* delete the file and create a new state file
* terraform force-unlock
* state.tf-unlock

NO.54 What is the result of the following terraform function call?
* True
* False

https://www.terraform.io/docs/configuration/functions/index.html

NO.55 You have been working in a Cloud provider account that is shared with other team members. You previously used Terraform to create a load balancer that is listening on port 80. After some application changes, you updated the Terraform code to change the port to 443.

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

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

What will happen when you terraform apply upon returning to your desk?
* Terraform will not make any changes to the Load Balancer and will update the state file to reflect any changes made.
* Terraform will change the port back to 80 in your code
* Terraform will change the load balancer port to 80, and) then change it back to 443
* Terraform will fail with in error because the state file is no longer accurate

NO.56 In regards to deploying resources in multi-cloud environments, what are some of the benefits of using

Terraform rather than a provider’s native tooling? (select three)
* Terraform can help businesses deploy applications on multiple clouds and on-premises infrastructure.
* Terraform is not cloud-agnostic and can be used to deploy resources across a single public cloud.
* Terraform simplifies management and orchestration, helping operators build large-scale, multi-cloud

infrastructure.

* Terraform can manage cross-cloud dependencies.

Explanation

Terraform is cloud-agnostic and allows a single configuration to be used to manage multiple providers, and to

even handle cross-cloud dependencies. This simplifies management and orchestration, helping operators build

large-scale multi-cloud infrastructures.

https://www.terraform.io/intro/use-cases.html

NO.57 What allows you to conveniently switch between multiple instances of a single configuration within its single backend?

* Local backends
* Providers
* Remote backends
* Workspaces

Explanation

Named workspaces allow conveniently switching between multiple instances of a single configuration within its single backend. … A common use for multiple workspaces is to create a parallel, distinct copy of a set of infrastructure in order to test a set of changes before modifying the main production infrastructure.

Workspaces, allowing multiple states to be associated with a single configuration. The configuration still has only one backend, but multiple distinct instances of that configuration to be deployed without configuring a new backend or changing authentication credentials.

https://www.terraform.io/docs/state/workspaces.html

NO.58 Select all Operating Systems that Terraform is available for. (select five)

* Linux
* macOS
* Unix
* Solaris
* Windows
* FreeBSD

Explanation

Terraform is available for macOS, FreeBSD, OpenBSD, Linux, Solaris, Windows https://www.terraform.io/downloads.html

NO.59 You are reviewing Terraform configurations for a big project in your company. You noticed that there are several identical sets of resources that appear in multiple configurations. What feature of Terraform would you recommend to use to reduce the amount of cloned configuration between the different configurations?

* Packages
* Backends
* Provisioners
* Modules

Modules are reusable configuration packages that Terraform can share through a variety of sources including Terraform Registries, GitHub, and Amazon S3 buckets.

A module is a container for multiple resources that are used together. Modules can be used to create lightweight abstractions, so that

you can describe your infrastructure in terms of its architecture, rather than directly in terms of physical objects.

Modules are reusable configuration packages that Terraform can share through a variety of sources including Terraform Registries, GitHub, and Amazon S3 buckets.

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

NO.60 In order to reduce the time it takes to provision resources, Terraform uses parallelism. By default, how many

resources will Terraform provision concurrently?
* 5
* 50
* 10
* 20

NO.61 ABC Enterprise has recently tied up with multiple small organizations for exchanging database information.

Due to this, the firewall rules are increasing and are more than 100 rules. This is leading firewall configuration

file that is difficult to manage. What is the way this type of configuration can be managed easily?
* Terraform Backends
* Terraform Functions
* Dynamic Blocks
* Terraform Expression

NO.62 What is not processed when running a terraform refresh?
* State file
* Configuration file
* Credentials
* Cloud provider

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

NO.63 During a terraform apply, a resource is successfully created but eventually fails during provisioning. What happens to the resource?
* The resource will be planned for destruction and recreation upon the next terraform apply
* Terraform will retry to provision again.
* The failure of provisioner will be ignored and it will not cause a failure to terraform apply
* The resource will be automatically destroyed.

If a creation-time provisioner fails, the resource is marked as tainted. A tainted resource will be planned for destruction and recreation upon the next terraform apply. Terraform does this because a failed provisioner can leave a resource in a semi-configured state. Because Terraform cannot reason about what the provisioner does, the only way to ensure proper creation of a resource is to recreate it. This is tainting.

You can change this behavior by setting the on_failure attribute, which is covered in detail below.

https://www.terraform.io/docs/provisioners/index.html#creation-time-provisioners

https://www.terraform.io/docs/provisioners/index.html#destroy-time-provisioners

https://www.terraform.io/docs/provisioners/index.html#failure-behavior

NO.64 Which of the following is a meta-argument defined in the configuration files of Terraform?
* tfvar
* depends_on
* instance aws
* varl

NO.65 State locking does not happen automatically and must be specified at run
* False
* True
State locking happens automatically on all operations that could write state.

https://www.terraform.io/docs/state/locking.html

NO.66 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’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 “aws_instance” “foo” 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.67 Which option can not be used to keep secrets out of Terraform configuration files?
* A Terraform provider
* Environment variables
* A -var flag
* secure string

NO.68 Module version is required to reference a module on the Terraform Module Registry.

* True
* False

NO.69 Workspaces in Terraform provides similar functionality in the open-source, Terraform Cloud, and Enterprise

versions of Terraform.
* True
* False
Explanation

https://www.terraform.io/docs/cloud/migrate/workspaces.html

Workspaces, managed with the terraform workspace command, aren’t the same thing as Terraform Cloud’s

workspaces. Terraform Cloud workspaces act more like completely separate working directories; CLI

workspaces are just alternate state files.

HashiCorp is a technology company that provides solutions for infrastructure automation. One of their most popular tools is Terraform, an open-source infrastructure as code software. Terraform allows users to define and manage infrastructure in a declarative manner, providing a simple way to automate the provisioning and deployment of resources across multiple cloud providers and on-premises data centers.

Use Valid New TA-002-P Test Notes & TA-002-P Valid Exam Guide:
https://www.validbraindumps.com/TA-002-P-exam-prep.html]