



TA-002-P^{Q&As}

HashiCorp Certified: Terraform Associate

Pass HashiCorp TA-002-P Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

<https://www.passapply.com/ta-002-p.html>

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by HashiCorp
Official Exam Center

- ⚙️ **Instant Download** After Purchase
- ⚙️ **100% Money Back** Guarantee
- ⚙️ **365 Days** Free Update
- ⚙️ **800,000+** Satisfied Customers





QUESTION 1

You have declared a variable name my_var in terraform configuration without a value associated with it.

```
variable my_var {}
```

After running terraform plan it will show an error as variable is not defined.

A. True

B. False

Correct Answer: B

Input variables are usually defined by stating a name, type and a default value. However, the type and default values are not strictly necessary. Terraform can deduct the type of the variable from the default or input value. Variables can be

predetermined in a file or included in the command-line options. As such, the simplest variable is just a name while the type and value are selected based on the input.

```
variable "variable_name" {}
```

```
terraform apply-var variable_name="value"
```

The input variables, like the one above, use a couple of different types: strings, lists, maps, and boolean. Here are some examples of how each type are defined and used.

String

Strings mark a single value per structure and are commonly used to simplify and make complicated values more user-friendly. Below is an example of a string variable definition.

```
variable "template" {
```

```
type = string
```

```
default = "01000000-0000-4000-8000-000030080200"
```

```
}
```

A string variable can then be used in resource plans. Surrounded by double quotes, string variables are a simple substitution such as the example underneath.

```
storage = var.template
```

List

Another type of Terraform variables lists. They work much like a numbered catalogue of values. Each value can be called by their corresponding index in the list. Here is an example of a list variable definition.

```
variable "users" {
```

```
type = list
```



```
default = ["root", "user1", "user2"]  
  
}
```

Lists can be used in the resource plans similarly to strings, but you'll also need to denote the index of the value you are looking for.

```
username = var.users[0]
```

Map

Maps are a collection of string keys and string values. These can be useful for selecting values based on predefined parameters such as the server configuration by the monthly price.

```
variable "plans" {  
  
  type = map  
  
  default = {  
  
    "5USD" = "1xCPU-1GB"  
  
    "10USD" = "1xCPU-2GB"  
  
    "20USD" = "2xCPU-4GB"  
  
  }  
  
}
```

You can access the right value by using the matching key. For example, the variable below would set the plan to "1xCPU-1GB".

```
plan = var.plans["5USD"]
```

The values matching to their keys can also be used to look up information in other maps. For example, underneath is a shortlist of plans and their corresponding storage sizes.

```
variable "storage_sizes" {  
  
  type = map  
  
  default = {  
  
    "1xCPU-1GB" = "25"  
  
    "1xCPU-2GB" = "50"  
  
    "2xCPU-4GB" = "80"  
  
  }  
  
}
```

These can then be used to find the right storage size based on the monthly price as defined in the previous example.

```
size = lookup(var.storage_sizes, var.plans["5USD"]) Boolean
```



The last of the available variable type is boolean. They give the option to employ simple true or false values. For example, you might wish to have a variable that decides when to generate the root user password on a new deployment.

```
variable "set_password" {  
  
  default = false  
  
}
```

The above example boolean can be used similarly to a string variable by simply marking down the correct variable.

```
create_password = var.set_password
```

By default, the value is set to false in this example. However, you can overwrite the variable at deployment by assigning a different value in a command-line variable.

```
terraform apply-var set_password="true"
```

QUESTION 2

terraform refresh command will not modify infrastructure, but does modify the state file.

- A. True
- B. False

Correct Answer: A

The terraform refresh command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file. This does not modify infrastructure, but does modify the state file. <https://www.terraform.io/docs/commands/refresh.html>

QUESTION 3

Which statements best describes what the local variable assignment is doing in the following code snippet:

- A. Create a distinct list of route table name objects
- B. Create a map of route table names to subnet names
- C. Create a map of route table names from a list of subnet names
- D. Create a list of route table names eliminating duplicates

Correct Answer: D

QUESTION 4

How would you reference the Volume IDs associated with the ebs_block_device blocks in this configuration?



```
resource "aws_instance" "example" {
  ami = "ami-abc123"
  instance_type = "t2.micro"

  ebs_block_device {
    device_name = "sda2"
    volume_size = 16
  }

  ebs_block_device {
    device_name = "sda3"
    volume_size = 20
  }
}
```

- A. `aws_instance.example.ebs_block_device[*].volume_id`
- B. `aws_instance.example.ebs_block_device.volume_id`
- C. `aws_instance.example.ebs_block_device[sda2,sda3].volume_id`
- D. `aws_instance.example.ebs_block_device.*.volume_id`

Correct Answer: A

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html

QUESTION 5

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?

- A. Force developer to use v0.12 by using terraform setting `required_version` and set it to `>=0.12`.
- B. Refactor the code to support both v0.11, and v0.12. It might be a difficult process, but there is no other way.
- C. Add a condition in front of each such specific construct, to check whether the running terraform version is v0.11 or v0.12, and work accordingly.
- D. 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.



Correct Answer: A

<https://www.terraform.io/docs/configuration/terraform.html>

[Latest TA-002-P Dumps](#)

[TA-002-P Study Guide](#)

[TA-002-P Exam Questions](#)