HashiCorp Terraform Associate Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Table of Contents

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- **Practice answering questions under realistic conditions,**
- **Improve accuracy and speed,**
- **Review explanations to strengthen weak areas, and**
- **Approach the exam with greater confidence.**

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here’s how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don’t just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There’s no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

Questions

1. **Which command checks that all code referencing multiple modules is properly formatted without making changes?**

   A. terraform fmt -write=false

   B. terraform fmt -list -recursive

   C. terraform fmt -check -recursive

   D. terraform fmt -check

2. **You can access state stored with the local backend by using the terraform_remote_state data source.**

   A. False

   B. Not possible

   C. Only with remote backend

   D. True

3. **How can Terraform variables be provided to a configuration?**

   A. They are only defined in the variables file

   B. They must be hard-coded in the configuration

   C. They cannot be overridden at apply time

   D. They can be provided via environment variables, -var command-line flag, or .tfvars files

4. **Which command will show all resources that would be deleted if you proceed with destruction?**

   A. Run terraform plan -destroy.

   B. Run terraform destroy and it will first output all the resources that will be deleted before prompting for approval.

   C. This command does not exist.

   D. Run terraform apply with a destroy plan.

5. **Which Terraform command lets you view all attributes and details of a managed resource?**

   A. terraform state show

   B. terraform state list

   C. terraform state pull

   D. terraform show

6. What is the correct syntax to define an output in a Terraform module to expose the public IP of an aws_instance named 'web'?

A. output "ip" { value = aws_instance.web.public_ip }

B. output "ip" { value = module.web.public_ip }

C. export ip = aws_instance.web.public_ip

D. output "ip" { value = var.ip }

7. Which argument is required when declaring a Terraform output?

A. Sensitive

B. Description

C. Default

D. Value

8. You can develop a custom provider to manage its resources using Terraform.

A. True

B. False

C. It is not possible

D. Only with a paid version of Terraform

9. What does terraform destroy do?

A. It destroys all resources managed by the current state

B. It deletes the configuration files from disk

C. It removes the state file

D. It only plans to destroy resources but does not apply

10. Backends in Terraform are responsible for storing and retrieving state data.

A. They validate configurations

B. They manage external plugins

C. They store and retrieve state data

D. They run providers

Answers

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

Explanations

1. Which command checks that all code referencing multiple modules is properly formatted without making changes?

A. terraform fmt -write=false

B. terraform fmt -list -recursive

C. terraform fmt -check -recursive

D. terraform fmt -check

Focusing on verifying formatting without changing files across a multi-module setup. The -check flag makes Terraform fmt run as a dry run: it determines whether any file would be reformatted and exits with a non-zero status if formatting is needed, without writing any changes. The -recursive flag ensures this check travels into all subdirectories, including each module, so every Terraform file in the project is covered. Together, terraform fmt -check -recursive checks that all code across all modules is properly formatted and leaves everything untouched if it's already correct. If everything is formatted, it exits cleanly with code 0.

2. You can access state stored with the local backend by using the terraform_remote_state data source.

A. False

B. Not possible

C. Only with remote backend

D. True

The main idea is that the terraform_remote_state data source is designed to read outputs from another Terraform state file, wherever that state is stored. That means you're not limited to remote backends—you can read from a local state as well if you configure it to point at the local state file. So, if one configuration stores its state locally on disk, you can still access its outputs from another configuration by using terraform_remote_state and directing it to that local state file (for example, by using a local backend configuration that points to the path of the local terraform.tfstate). This lets you reuse values produced in one configuration, such as IDs or other outputs, in another configuration without duplicating logic. In short, you can access state stored with the local backend through the terraform_remote_state data source, which is why the statement is true.

3. How can Terraform variables be provided to a configuration?

A. They are only defined in the variables file

B. They must be hard-coded in the configuration

C. They cannot be overridden at apply time

D. They can be provided via environment variables, -var command-line flag, or .tfvars files

Terraform variables can be provided from multiple sources and overridden at plan or apply time. You can supply values through environment variables (using the TF_VAR_<NAME> prefix), via the -var CLI flag to set a specific value, or with .tfvars files that Terraform reads to fill in variable values. This mix gives you flexibility to keep configuration separate from data and to adapt configurations for different environments without editing the code. Therefore, the method described—environment variables, the -var flag, or .tfvars files—covers the standard ways to provide variable values. The other statements are not accurate because variables aren't limited to a single file, don't have to be hard-coded, and can indeed be overridden at apply time through these mechanisms.

4. Which command will show all resources that would be deleted if you proceed with destruction?

A. Run terraform plan -destroy.

B. Run terraform destroy and it will first output all the resources that will be deleted before prompting for approval.

C. This command does not exist.

D. Run terraform apply with a destroy plan.

Previewing destructive changes before applying them is crucial to avoid accidental deletions. Running the plan command with the -destroy flag asks Terraform to generate a what-if execution plan for destroying resources, but without making any changes. It will list every resource that would be deleted, so you can review and confirm exactly what would be removed if you proceed. This explicit preview is why it's the best choice: it provides a clear, non-destructive view of the destruction scope. While the destroy action will eventually remove resources and typically shows what will be destroyed before you confirm, plan -destroy is specifically designed to display that impact without performing any deletions, making it the most reliable way to see all resources that would be deleted.

5. Which Terraform command lets you view all attributes and details of a managed resource?

A. terraform state show

B. terraform state list

C. terraform state pull

D. terraform show

To see all attributes of a specific managed resource, use the state show command. It reads the current state and prints every attribute for the given resource address, including computed values and the resource ID. For example, terraform state show aws_instance.web_server would display the full details Terraform knows about that instance. This is why it's the best choice: it targets a single resource and dumps its complete state. The other state-related commands serve different purposes: listing shows all resource addresses in state, fetching the state from a backend, or displaying the entire state or a plan, rather than focusing on the full details of one resource.

6. What is the correct syntax to define an output in a Terraform module to expose the public IP of an aws_instance named 'web'?

A. output "ip" { value = aws_instance.web.public_ip }

B. output "ip" { value = module.web.public_ip }

C. export ip = aws_instance.web.public_ip

D. output "ip" { value = var.ip }

Outputs in a Terraform module are how you expose values to the module's caller. To reveal the public IP of an aws_instance named web, you define an output and set its value to the resource's attribute. The correct approach uses the resource reference format: aws_instance.web.public_ip, placed inside an output block. So the valid form is output "ip" { value = aws_instance.web.public_ip }. This makes the instance's public IP available to whatever module consumes this one. Other patterns aren't appropriate here. export isn't Terraform syntax for outputs, and var.ip would just pass in a value from a variable rather than expose the resource's attribute. Referencing module.web.public_ip would point to a child module's output, not to the aws_instance resource within this module.

7. Which argument is required when declaring a Terraform output?

A. Sensitive

B. Description

C. Default

D. Value

The thing being tested is what you must include to actually expose something from a module. When you declare an output, Terraform needs to know what data to return, and that comes from the value expression. That value is required because it defines the content of the output. Without it, there's nothing to output and Terraform will error. The other attributes—description and sensitive—are optional, used for documentation or hiding the value in CLI output, and there are additional optional attributes like type, but they aren't required. In practice, you'd write something like: output "example" { value = aws_instance.web.public_ip } and you could add a description or mark it sensitive if needed.

8. You can develop a custom provider to manage its resources using Terraform.

A. True

B. False

C. It is not possible

D. Only with a paid version of Terraform

Terraform is built to be extended with providers, which are plugins that implement how to create, read, update, and delete resources for a given API. This design allows you to write your own custom provider to manage resources exposed by any service, including your own, and then manage those resources with Terraform alongside other infrastructure. You don't need any paid edition to do this—the open-source version of Terraform plus the Terraform Plugin SDK is all that's required. You'd implement the provider in Go, define the resources and data sources, build a binary, and configure Terraform to use it (locally or by publishing it). So the statement is true: you can develop a custom provider to manage its resources using Terraform.

9. What does terraform destroy do?

A. It destroys all resources managed by the current state

B. It deletes the configuration files from disk

C. It removes the state file

D. It only plans to destroy resources but does not apply

Terraform destroy removes the infrastructure that Terraform is currently managing in its state. When you run it, Terraform builds a plan to delete every resource tracked in the state and then applies that plan, effectively deleting those resources from the cloud or on-prem systems. After the run, the state is updated to reflect that those resources no longer exist, so Terraform's view of reality matches what's left. It does not delete your configuration files on disk, and it doesn't remove the state file itself unless you take separate action to do so. If you only want to remove specific resources, you can target them, but by default it destroys everything managed by the current state.

10. Backends in Terraform are responsible for storing and retrieving state data.

A. They validate configurations

B. They manage external plugins

C. They store and retrieve state data

D. They run providers

Backends determine where Terraform stores and retrieves state data. The state is a snapshot that tracks all managed resources, their IDs, and metadata Terraform needs to plan and apply changes correctly. A backend can store this state locally or remotely, and remote backends often provide locking to prevent concurrent writes, which is essential for teamwork. This role is distinct from validating configurations, which happens during planning and applying, and from how Terraform loads and runs provider plugins, which are responsible for creating, updating, and destroying resources. So the statement that backends store and retrieve state data best captures their purpose.

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://hashicorpterraform.examzify.com

We wish you the very best on your exam journey. You've got this!