Terraform Associate Practice Exam (Sample)

Study Guide

BY EXAMZIFY

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. As a member of the operations team, which provision is best to run a script on a virtual machine created by Terraform?
   - A. local-exec
   - B. remote-exec
   - C. user-data
   - D. provisioner

2. Which command is used to initialize a working directory containing Terraform configuration files?
   - A. terraform start
   - B. terraform init
   - C. terraform bootstrap
   - D. terraform configure

3. Which file typically contains the configuration for Terraform providers?
   - A. main.tf
   - B. provider.tf
   - C. variables.tf
   - D. outputs.tf

4. Which errors does 'terraform validate' report related to configuration consistency?
   - A. Duplicating variable names
   - B. Declaring a resource identifier more than once
   - C. Encoding issues in the configuration files
   - D. Inconsistent indentation of the HCL

5. When working with Terraform Enterprise and Cloud Workspaces, how are they conceptually viewed?
   - A. As multiple instances of the same workspace
   - B. As completely separate working directories
   - C. As extensions of the same workspace
   - D. As deprecated versions of the original Terraform

6. Which provisioner runs a process on the created resource?

- **A. local-exec**
- **B. remote-exec**
- **C. null-exec**
- **D. command-exec**

7. How do you specify a tag of v1.0.0 when referencing a Git module?

- **A. Append IT to the end of the URL**
- **B. Use the argument ref=v1.0.0**
- **C. Add v1.0.0 after the repository name**
- **D. Include version in the configuration file**

8. Where in your Terraform configuration would you specify a state backend?

- **A. In the providers block**
- **B. In the variables block**
- **C. In the terraform block**
- **D. In the resources block**

9. What is the workflow for deploying new infrastructure using Terraform?

- **A. Write code and run commands randomly**
- **B. Run init, apply, then plan**
- **C. Write a configuration, then init, plan, and apply**
- **D. Only run the apply command**

10. Which function is not valid in Terraform's string functions?

- **A. Join**
- **B. Replace**
- **C. Slice**
- **D. Split**

Answers

1. **B**
2. **B**
3. **B**
4. **B**
5. **B**
6. **B**
7. **B**
8. **C**
9. **C**
10. **C**

Explanations

1. As a member of the operations team, which provision is best to run a script on a virtual machine created by Terraform?

A. local-exec

B. remote-exec

C. user-data

D. provisioner

The option to run a script on a virtual machine created by Terraform that is most suitable is remote-exec. This provisioner is specifically designed to execute commands or scripts on a remote machine after it has been created and is accessible via SSH or WinRM, depending on the environment.   When you use remote-exec, Terraform establishes a connection to the VM and can run scripts directly in that context, which is essential for configuring the machine as needed after its initial creation. This capability allows for greater flexibility, such as installing software, configuring services, or performing any number of tasks that must be executed in the context of the newly created VM. Other options like local-exec are intended for running commands on the machine that is executing Terraform itself, rather than on the remote VM. User-data typically refers to a mechanism used in cloud-init or similar services to pass configuration scripts to instances at launch time, but it does not provide the same execution control as remote-exec. A provisioner is more of a category that encompasses both local-exec and remote-exec, but it does not precisely indicate which type is best suited for running scripts on a VM created by Terraform. Thus, remote-exec stands out as the most appropriate choice in this context.

2. Which command is used to initialize a working directory containing Terraform configuration files?

A. terraform start

B. terraform init

C. terraform bootstrap

D. terraform configure

The command used to initialize a working directory containing Terraform configuration files is "terraform init." This command prepares the directory for other Terraform commands by setting up the necessary file structures, downloading provider plugins, and verifying that the required plugins are available. When you run "terraform init," Terraform performs several critical tasks: 1. **Provider Initialization**: It detects which providers are specified in the configuration files and downloads the required provider binaries from the Terraform Registry. This ensures that you have the correct version of each provider necessary for your project. 2. **Backend Initialization**: If you are using a remote backend for storing your Terraform state, "terraform init" will configure it as specified in your configuration files. This enables the management of your state file remotely. 3. **Module Installation**: If your configuration references any modules, "terraform init" will also retrieve those modules, ensuring that your configuration will function as intended. 4. **File Structure Setup**: It sets up the necessary folder structures and files to ensure the working directory is ready for further Terraform commands. The other choices do not serve this purpose. "terraform start," "terraform bootstrap," and "terraform configure" are not valid Terraform commands, as they do not perform the initialization functions that "terraform init

3. Which file typically contains the configuration for Terraform providers?

A. main.tf

B. provider.tf

C. variables.tf

D. outputs.tf

The configuration for Terraform providers is generally contained within a file named "provider.tf." This file is specifically designated for defining the providers used in a Terraform configuration, along with their required settings and versions. By isolating provider definitions in this way, it enhances clarity and organization, allowing users to easily identify and manage the integration of various cloud services or APIs. In "provider.tf," you typically specify details such as which provider to use (e.g., AWS, Azure, Google Cloud) and any necessary credentials or configurations required to connect to that provider's services. This structure is useful because it allows for a focused view of provider-related configurations, making it easier to adjust or update these settings without needing to sift through the broader configuration files. Other files serve different purposes within a Terraform configuration. "main.tf" often contains the primary resources and core logic of the infrastructure being defined, whereas "variables.tf" is used to define input variables that can be referenced throughout the configuration. "outputs.tf" is dedicated to specifying output values that can be used after the Terraform execution is complete. Each file has its role, but when it comes to defining providers, "provider.tf" is the standard choice.

4. Which errors does 'terraform validate' report related to configuration consistency?

A. Duplicating variable names

B. Declaring a resource identifier more than once

C. Encoding issues in the configuration files

D. Inconsistent indentation of the HCL

The command 'terraform validate' primarily checks for syntax and configuration errors within the Terraform configuration files. Among the given choices, declaring a resource identifier more than once is a clear violation of Terraform's resource declaration rules. When Terraform encounters duplicate resource identifiers, it cannot determine which resource to reference, leading to ambiguity and potential functional issues in the infrastructure definition. This form of error directly affects the integrity and reliability of the configuration, as Terraform expects resource names to be unique within the same module. Thus, 'terraform validate' will report this situation as an error, allowing the user to correct it before proceeding to apply changes. In contrast, duplicating variable names and inconsistent indentation may not necessarily be flagged as critical errors by 'terraform validate,' as it primarily focuses on semantic correctness rather than stylistic consistency. Encoding issues in configuration files could potentially lead to parsing errors but would likely arise during the execution phase rather than during validation.

5. When working with Terraform Enterprise and Cloud Workspaces, how are they conceptually viewed?

A. As multiple instances of the same workspace

B. As completely separate working directories

C. As extensions of the same workspace

D. As deprecated versions of the original Terraform

When working with Terraform Enterprise and Cloud Workspaces, they are conceptually viewed as completely separate working directories. Each workspace serves as an isolated environment allowing users to manage infrastructure and states independently. This means that each workspace can have its own set of variables, configurations, and state files that are distinct from those of other workspaces. This separation is crucial for managing different environments like development, testing, and production, enabling teams to work simultaneously without interference between projects. The concept of separate working directories aligns with Terraform's design philosophy of maintaining infrastructure as code while allowing flexibility in environment management. Each workspace can be thought of as a dedicated context for deploying and managing resources, thus providing a clean and organized way to interact with multiple environments. This independence of workspaces ensures safe and predictable infrastructure management practices.

6. Which provisioner runs a process on the created resource?

A. local-exec

B. remote-exec

C. null-exec

D. command-exec

The provisioner that runs a process on the created resource is the remote-exec provisioner. This type of provisioner is designed to execute scripts or commands directly on the remote resource that has just been created. For example, if you are provisioning a virtual machine in the cloud, the remote-exec provisioner allows you to run commands within the context of that machine after it has been provisioned. Using the remote-exec provisioner is beneficial when you need to configure the resource or install software as part of the deployment process. It connects to the resource via SSH (for Linux-based systems) or WinRM (for Windows), executing the specified commands in the proper environment of the created resource. In contrast, the local-exec provisioner runs commands on the machine where Terraform is executed, not on the target resource itself. Null-exec is a placeholder provisioner that does nothing, and "command-exec" is not a recognized Terraform provisioner type. Thus, remote-exec is the correct choice for running processes on the resource that has been created.

7. How do you specify a tag of v1.0.0 when referencing a Git module?

A. Append IT to the end of the URL

B. Use the argument ref=v1.0.0

C. Add v1.0.0 after the repository name

D. Include version in the configuration file

To specify a tag of v1.0.0 when referencing a Git module in Terraform, using the argument ref=v1.0.0 is the correct approach. The `ref` argument allows you to define a specific version of the module you wish to use, whether it be a branch, tag, or commit hash. By setting `ref` to v1.0.0, Terraform will pull the module code associated with that specific tag from the Git repository. This method ensures that your infrastructure configuration is consistent and reproducible by locking it to a known, stable version of the module, rather than relying on the latest commit from the repository, which may lead to unexpected changes and behaviors in your deployments. Regarding the other options, appending IT to the URL, adding v1.0.0 after the repository name, or including version in the configuration file may not follow the standard syntax and semantics that Terraform expects when working with Git module sources. Thus, they would not effectively achieve the goal of accurately pinning to that specific version.

8. Where in your Terraform configuration would you specify a state backend?

A. In the providers block

B. In the variables block

C. In the terraform block

D. In the resources block

The state backend in a Terraform configuration is specified in the terraform block. This block is dedicated to defining settings related to the Terraform configuration, including settings for the backend which is responsible for storing the state file. Defining the backend in the terraform block enables you to configure how and where Terraform will save the state of your infrastructure. This could be a local file, or it could be a remote backend like Amazon S3, HashiCorp Consul, or Terraform Cloud, allowing for collaboration and state management across different team members. By centralizing this configuration in the terraform block, it provides clarity and organization in the Terraform files, distinguishing these settings from resource definitions or variable declarations. Other areas such as the providers block, variables block, and resources block serve different purposes. The providers block configures the providers needed for the infrastructure resources, the variables block defines input variables for parametrization, and the resources block is focused solely on declaring infrastructure resources. Thus, none of these blocks would be appropriate for specifying a state backend.

9. What is the workflow for deploying new infrastructure using Terraform?

A. Write code and run commands randomly

B. Run init, apply, then plan

C. Write a configuration, then init, plan, and apply

D. Only run the apply command

The workflow for deploying new infrastructure using Terraform follows a systematic approach to ensure that the configuration is correctly managed and applied. The correct sequence involves writing a configuration file that defines the desired state of the infrastructure. After creating the configuration, the first command to run is 'init', which initializes the working directory containing the Terraform configuration files. This step downloads any necessary provider plugins and prepares the environment for working with Terraform. Following initialization, the 'plan' command is executed. This command allows users to preview the changes that will occur based on the current configuration compared to the existing state. It's an essential step to verify that the changes are as expected before any infrastructure alterations occur. Finally, the 'apply' command is run to actually implement the changes outlined in the plan, creating the infrastructure as specified. This three-step process — writing a configuration, then initializing, planning, and applying — ensures that users have clear visibility into the changes being made and reduces the risk of unintended consequences during the deployment of new infrastructure.

10. Which function is not valid in Terraform's string functions?

A. Join

B. Replace

C. Slice

D. Split

The function that is not valid in Terraform's string functions is Slice. In Terraform, string functions fulfill specific roles related to manipulating strings, but Slice is not among them.   The Join function is used to concatenate elements of a list into a single string, using a specified delimiter. Replace allows for specific substrings within a string to be replaced with another substring. The Split function divides a string into a list of substrings based on a specified delimiter. Each of these functions is part of Terraform's core capabilities for string manipulation. On the other hand, Slice refers more to a collection of elements, such as lists or maps, and is used with data types that support indexing rather than direct string manipulation. This distinction highlights why Slice does not fit into the category of string functions within Terraform. Understanding the specific roles and applications of these functions is crucial for effective Terraform usage.

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://terraformassociate.examzify.com

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