TERRAFORM ASSOCIATE PRACTICE EXAM (SAMPLE)

STUDY GUIDE



SAMPLE STUDY GUIDE WITH LIMITED QUESTIONS

VISIT US ONLINE FOR THE FULL VERSION STUDY GUIDE

https://terraformassociate.examzify.com

Copyright © 2026 by Examzify - A Kaluba Technologies Inc. product.

ALL RIGHTS RESERVED.

No part of this book may be reproduced or transferred in any form or by any means, graphic, electronic, or mechanical, including photocopying, recording, web distribution, taping, or by any information storage retrieval system, without the written permission of the author.

Notice: Examzify makes every reasonable effort to obtain accurate, complete, and timely information about this product from reliable sources.

SAMPLE

Table of Contents

Copyright 1

Table of Contents 2

Introduction 3

How to Use This Guide 4

Questions 5

Answers 8

Explanations 10

Next Steps 16

SAMPLE

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.

SAMPLE

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!

SAMPLE

Questions

SAMPLE

1. Why should secrets not be hard coded into Terraform code?

A. It complicates infrastructure deployment

B. Terraform code is stored in version control

C. Hard coding is generally faster for deployment

D. Terraform is designed to handle sensitive data

2. Which of the following commands will check the overall compatibility of code with existing infrastructure?

A. terraform apply

B. terraform validate

C. terraform plan

D. terraform init

3. What is the use of the terraform output command?

A. To display the state of resources

B. To show the created resource configurations

C. To retrieve outputs defined in the configuration

D. To summarize changes made

4. How does Terraform know the order to create resources?

A. Through user-defined order in configuration

B. Automatically based on dependencies

C. Using defined tags for resource grouping

D. Users must manually specify order

5. Is it necessary to initialize a Terraform backend before it can be configured?

A. True

B. False

C. Only in production environments

D. Only for remote backends

SAMPLE

6. What does 'terraform apply' primarily do?

A. Creates infrastructure based on configurations

B. Changes the state file directly

C. Validates the configuration only

D. Destroys existing infrastructure

7. How does Terraform determine dependencies between resources?

A. Through user-defined settings in the configuration

B. By checking the state file only

C. By automatically building a resource graph

D. By executing resources in the order they are defined

8. Which statement is true about Terraform and Terraform providers concerning version numbers?

A. They can have different major version numbers in a configuration

B. They must use the same major version number in a single configuration

C. They can operate independently of each other

D. They do not have any versioning requirements

9. Can open source Terraform import publicly-accessible and open-source modules?

A. True

B. False

C. Only if configured correctly

D. Only for specific providers

10. What does the command 'terraform fmt' do when executed without flags in a directory with configuration files?

A. It checks the formatting without making changes

B. It rewrites Terraform files in a canonical format

C. It formats comments in the files

D. It only validates the files without changing them

SAMPLE

Answers

SAMPLE

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

SAMPLE

Explanations

SAMPLE

1. Why should secrets not be hard coded into Terraform code?

A. It complicates infrastructure deployment

B. Terraform code is stored in version control

C. Hard coding is generally faster for deployment

D. Terraform is designed to handle sensitive data

Hard coding secrets into Terraform code poses significant risks, particularly regarding security and maintainability. One of the major reasons for avoiding hard-coded secrets is that Terraform code is typically stored in version control systems like Git. When secrets are hard coded, they become part of the source code that may be exposed to anyone with access to the repository. This exposure can lead to unauthorized access and potential data breaches, as sensitive information such as API keys, passwords, and other confidential data will inadvertently be shared with anyone who has access to the repository history. Additionally, using version control means that any changes to the code, including alterations of hard-coded secrets, are tracked over time. This poses a security challenge because the history will contain these sensitive details unless they are properly managed and removed. Best practices recommend using parameters or environment variables to handle sensitive information, which keeps the secrets out of the codebase and reduces the risk of exposure. Moreover, Terraform has built-in mechanisms to manage sensitive data, such as the ability to mark variables as sensitive and integration with secret management solutions, which allows for safer handling of sensitive information.

2. Which of the following commands will check the overall compatibility of code with existing infrastructure?

A. terraform apply

B. terraform validate

C. terraform plan

D. terraform init

The command that checks the overall compatibility of code with existing infrastructure is "terraform validate". This command is designed to analyze the configuration files in the Terraform project for syntax and internal consistency, ensuring that they are structured correctly and can be processed by Terraform. However, it does not connect to any infrastructure or validate against the existing state. Using "terraform validate" is crucial before deploying changes because it helps identify errors early in the development lifecycle. It can catch issues like misconfigurations or syntax errors before any interaction with real infrastructure, which can lead to changes that unintentionally disrupt current resources. While other commands serve important functions, they do not focus specifically on the code compatibility aspect. "Terraform apply" is used to apply changes to the infrastructure based on the configuration files but does not perform any checks against them prior to executing actions. "Terraform plan" generates an execution plan and shows what changes will be made, thus giving an idea of compatibility with existing infrastructure but only after validating the configuration. "Terraform init" initializes a new or existing Terraform working directory, allocating required backend settings and provider plugins but does not validate the configuration. Therefore, "terraform validate" is the most appropriate command for checking the compatibility of the code with the existing infrastructure.

SAMPLE

3. What is the use of the terraform output command?

A. To display the state of resources

B. To show the created resource configurations

C. To retrieve outputs defined in the configuration

D. To summarize changes made

The terraform output command is specifically designed to retrieve outputs that have been defined in your Terraform configuration. Outputs are a way to extract certain values from your Terraform state, making them easily accessible after running `terraform apply`. They can include important information such as IP addresses, resource IDs, or any other data that you want to be highlighted or passed to other configurations. When you define outputs in your Terraform configuration file—using the `output` block—you can access these outputs later through the terraform output command. This can be particularly useful when you want to pass information to other Terraform configurations or when you're working with multiple modules that require certain values from one another. The command does not display the state of resources directly, show full resource configurations, or summarize changes made during deployments, which distinguishes its purpose and functionality from the other options.

4. How does Terraform know the order to create resources?

A. Through user-defined order in configuration

B. Automatically based on dependencies

C. Using defined tags for resource grouping

D. Users must manually specify order

Terraform determines the order to create resources primarily through automatic dependency management. When Terraform analyzes the configuration files, it assesses the relationships between different resources. For instance, if one resource needs to reference the output of another resource (like an IP address or a database ID), Terraform understands these dependencies and constructs a dependency graph. This ensures that resources are created in the correct order, with each resource being provisioned only after its dependencies have been satisfied. This automatic handling of dependencies alleviates the requirement for users to manually specify the order in which resources should be created, as Terraform intelligently interprets the necessary relationships. As a result, Terraform can optimize the resource creation process while ensuring that everything is provisioned successfully without user intervention in the order of operations.

5. Is it necessary to initialize a Terraform backend before it can be configured?

A. True

B. False

C. Only in production environments

D. Only for remote backends

To initialize a Terraform backend before it can be configured is indeed necessary. This process involves establishing a connection between the Terraform configuration and the specified backend where the state file will be managed. The initialization step ensures that Terraform recognizes the backend and sets up any necessary configuration details, such as authentication requirements and communication protocols. During the initialization phase, Terraform will download the necessary provider plugins and prepare the working directory for deployment. This sets the foundation for managing and storing the state of resources created by Terraform. If the backend is not initialized, subsequent commands that interact with the backend, such as `terraform apply` or `terraform plan`, will fail since Terraform will not have the context or necessary information to manage state effectively. This is fundamental for both local and remote backends, and initiation must occur regardless of the environment, whether it is for development or production use.

SAMPLE

6. What does 'terraform apply' primarily do?

A. Creates infrastructure based on configurations

B. Changes the state file directly

C. Validates the configuration only

D. Destroys existing infrastructure

The command 'terraform apply' is primarily used to create or update infrastructure as defined in the Terraform configuration files. When you run this command, Terraform evaluates the current state of your infrastructure—stored in the state file—and compares it with the desired state defined in your configuration files. It then generates an execution plan that shows what actions will be taken to align the actual infrastructure with the desired state. If there are changes needed, 'terraform apply' will execute the necessary actions, such as creating new resources, modifying existing ones, or deleting resources that are no longer defined in the configuration. This makes it a crucial command for effectively managing and provisioning infrastructure in a declarative way. In contrast, directly changing the state file, validating configurations only, or destroying infrastructure are not the primary actions associated with this command. Although 'terraform apply' might result in modifying the state file as part of its process, its core function is to apply the desired changes to your infrastructure based on the configurations provided.

7. How does Terraform determine dependencies between resources?

A. Through user-defined settings in the configuration

B. By checking the state file only

C. By automatically building a resource graph

D. By executing resources in the order they are defined

Terraform determines dependencies between resources by automatically building a resource graph. This process involves analyzing the configuration files to understand how different resources relate to one another based on their configurations, inputs, and outputs. When creating a resource graph, Terraform is able to visualize and manage dependencies without requiring explicit definitions from the user. It takes into account various factors such as resource types, references made within the configuration, and the results of one resource's output that may be needed as an input for another resource. This capability is critical for ensuring that resources are created, updated, or destroyed in the correct order, respecting the defined dependencies. The automatic nature of this dependency graph helps Terraform efficiently manage provisioning, avoiding potential issues that could arise from manual intervention or an incorrect deployment order. This nuanced dependency management is one of Terraform's key features, enabling complex infrastructure configurations to be handled intelligently and effectively.

SAMPLE

8. Which statement is true about Terraform and Terraform providers concerning version numbers?

A. They can have different major version numbers in a configuration

B. They must use the same major version number in a single configuration

C. They can operate independently of each other

D. They do not have any versioning requirements

In a Terraform configuration, it is essential for all providers to operate within the same major version number. This requirement stems from the fact that major version changes in software typically introduce breaking changes. Therefore, if different providers within a configuration were to have different major version numbers, it could lead to inconsistencies and compatibility issues within the infrastructure code. By adhering to the same major version number, Terraform ensures that all components maintain compatibility and operate as expected, reducing the risk of encountering errors during execution and state management. This level of control allows users to manage dependencies effectively, ensuring smooth upgrades and modifications to infrastructure as code while minimizing disruptions caused by version conflicts. The other statements imply variances in versioning practices that Terraform does not support, especially around the idea of independent operations, which can lead to conflicts since providers depend on shared functionality and defined interfaces.

9. Can open source Terraform import publicly-accessible and open-source modules?

A. True

B. False

C. Only if configured correctly

D. Only for specific providers

The statement regarding whether open source Terraform can import publicly-accessible and open-source modules is indeed true. Terraform is designed to facilitate the management of infrastructure as code, and a key feature that supports this functionality is its ability to work with modules. Modules in Terraform enable users to encapsulate reusable configurations into manageable components. Because Terraform is open source, users have the flexibility to create, share, and use modules from various sources, including the Terraform Registry. This registry hosts many publicly-accessible modules created by the community which can be used freely. Importing these modules allows users to leverage existing work and best practices, promoting efficiency and standardization across infrastructure setups. By utilizing these open-source modules, practitioners can enhance their Terraform configurations without needing to reinvent the wheel, thereby speeding up development and deployment processes. The other options suggest limitations that do not apply to the general functionality of importing modules in Terraform. Hence, the answer underscores the versatile and collaborative aspects of Terraform as an open-source tool.

SAMPLE

10. What does the command 'terraform fmt' do when executed without flags in a directory with configuration files?

A. It checks the formatting without making changes

B. It rewrites Terraform files in a canonical format

C. It formats comments in the files

D. It only validates the files without changing them

The command 'terraform fmt' is designed to rewrite Terraform configuration files into a canonical format and style. When this command is executed without any flags, it scans the directory for files ending in .tf or .tf.json and automatically reformats them to adhere to the standard conventions recommended by Terraform. This helps ensure that the code is consistent and more readable across the board, which is especially important in collaborative environments where multiple users may be editing the same files. Reformatting includes changes to indentation, spacing, and other stylistic elements that do not affect the actual functionality of the configuration. The goal is to promote best practices in coding by ensuring uniformity in formatting, which can help developers quickly identify issues and understand the code structure. In contrast, other choices do not accurately reflect the function of 'terraform fmt'. For instance, while checking formatting or validating files is important, that specific functionality is not what 'terraform fmt' does by default when executed without flags. Instead, its primary role is to perform automatic formatting adjustments to maintain best practices in code styling.

SAMPLE

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!

SAMPLE