HASHICORP TERRAFORM INFRASTRUCTURE AS CODE (IAC) PRACTICE TEST (SAMPLE)

STUDY GUIDE



SAMPLE STUDY GUIDE WITH LIMITED QUESTIONS

VISIT US ONLINE FOR THE FULL VERSION STUDY GUIDE

https://hashicorpterraformiac.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. **How do you define dependencies manually if Terraform cannot automatically resolve them?**

   A. By using the depends_on argument within the resource or module block

   B. By specifying them in the Terraform state file

   C. By using comments to indicate dependencies

   D. By creating a separate dependency graph

2. **What method can be employed to ensure infrastructure changes are reviewed before being applied?**

   A. Using terraform validate before every apply

   B. Using terraform plan with version control systems for pull requests

   C. Directly editing the state file prior to applying

   D. Using a script to automate the apply process

3. **In the provided dynamic block, what is the issue that causes it to result in an error?**

   A. content block is not necessary

   B. for_each is missing

   C. An iterator argument is missing

   D. content should be called for_each

4. **How can you protect sensitive information in Terraform variables?**

   A. By marking variables as sensitive in the configuration

   B. By encrypting the configuration files

   C. By using version control to store variables

   D. By limiting access to the Terraform state file

5. **What is a workspace in Terraform?**

   A. A separate environment for different configurations

   B. A way to group related resources

   C. A method to store state files

   D. A command for managing modules

SAMPLE

6. How does Terraform determine changes needed for the infrastructure?

A. By analyzing logs for errors

B. By comparing local files

C. By comparing the current state to the desired state defined in the configuration

D. By checking external resource dependencies

7. What does the command 'terraform state rm' do?

A. Renames an item in the state

B. Adds an item to the state

C. Remove instances from the state

D. Displays the current state file

8. When Terraform crashes because Golang results in a panic, where will it log the results?

A. debug.log

B. panic.log

C. terraform.log

D. crash.log

9. What is the purpose of Terraform modules?

A. To indicate deprecated resources

B. To encapsulate and group related resources for reusability and organization

C. To increase Terraform configuration file size

D. To automate backups of the Terraform state

10. What is the benefit of implementing Terraform workspaces?

A. To facilitate easy coding

B. To manage different states for different environments

C. To create better documentation

D. To increase execution speed

SAMPLE

Answers

SAMPLE

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

SAMPLE

Explanations

SAMPLE

1. How do you define dependencies manually if Terraform cannot automatically resolve them?

A. By using the depends_on argument within the resource or module block

B. By specifying them in the Terraform state file

C. By using comments to indicate dependencies

D. By creating a separate dependency graph

Defining dependencies manually in Terraform is best accomplished using the `depends_on` argument within a resource or module block. This method is particularly useful when Terraform's automatic dependency resolution does not cover specific scenarios where the order of operations is crucial. The `depends_on` argument explicitly indicates to Terraform that one resource relies on another to be created or modified first. This is beneficial in cases where there are implicit dependencies that Terraform might not recognize, such as when one resource's configuration references another resource indirectly (for instance, by utilizing output values). By articulating dependencies in this way, you ensure that Terraform processes resources in the correct order, mitigating any potential errors that could arise from parallel processing. In contrast, other options are less effective for managing dependencies. Specifying dependencies in the Terraform state file could lead to confusion and is not a recommended approach, as the state file is meant to reflect the current state of resources rather than serve as a guide for dependencies. Using comments to indicate dependencies does not impact the execution plan and does not inform Terraform of any necessary ordering, making it a non-functional approach to dependency management. Creating a separate dependency graph may provide a visual representation of relationships between resources, but it cannot influence or control the execution order in Terraform itself, which

2. What method can be employed to ensure infrastructure changes are reviewed before being applied?

A. Using terraform validate before every apply

B. Using terraform plan with version control systems for pull requests

C. Directly editing the state file prior to applying

D. Using a script to automate the apply process

The correct approach to ensure that infrastructure changes are reviewed before being applied is by using Terraform plan along with version control systems for pull requests. This methodology facilitates a clear review process. When you run Terraform plan, it generates an execution plan that details what changes will be made to the infrastructure. This plan can be shared with team members, allowing them to review the proposed changes before they are executed. Integrating this process with a version control system enables collaboration among team members, where they can create pull requests with the planned changes. Each pull request can be meticulously reviewed, ensuring that all modifications go through a quality check before being merged and applied to the infrastructure. This practice minimizes the risk of errors and enhances accountability, as multiple eyes assess the planned alterations. While validating the configuration with `terraform validate` is a good practice, it does not inherently provide a review mechanism for proposed infrastructure changes. Directly editing the state file is not advisable due to the potential for introducing inconsistencies and errors, compromising the integrity of the infrastructure management. Automating the apply process with a script can streamline deployment but does not incorporate a review step, potentially leading to unvetted changes being applied directly. Therefore, the combination of Terraform plan and version control pull requests is the best method

SAMPLE

3. In the provided dynamic block, what is the issue that causes it to result in an error?

A. content block is not necessary

B. for_each is missing

C. An iterator argument is missing

D. content should be called for_each

In a dynamic block within Terraform, the purpose is to generate multiple nested blocks based on some collection. The use of the `for_each` argument is essential for iterating over that collection, which allows for the creation of multiple copies of the nested block with different configurations. If the `for_each` argument is missing, Terraform lacks the necessary information to know how many instances of the block are required or what data to populate each instance with. This results in an error because the dynamic block is fundamentally designed to operate on collections; without `for_each`, the dynamic block cannot generate any content. Therefore, the absence of `for_each` directly leads to the dynamic block being ineffective, resulting in an error message indicating that it cannot proceed with the block creation as intended. This highlights the critical role of iteration in defining dynamic content in Terraform configurations.

4. How can you protect sensitive information in Terraform variables?

A. By marking variables as sensitive in the configuration

B. By encrypting the configuration files

C. By using version control to store variables

D. By limiting access to the Terraform state file

Marking variables as sensitive in the Terraform configuration is a crucial practice for protecting sensitive information. When you declare a variable as sensitive, Terraform ensures that its value is not displayed in the command line output or logs, thus reducing the risk of leaking sensitive information such as passwords, API keys, or other confidential data. This feature helps manage sensitive data more securely by preventing accidental exposure during Terraform runs. It effectively informs Terraform and anyone using the code that the variable contains sensitive information. By correctly using the sensitive attribute, you maintain a level of confidentiality throughout the infrastructure provisioning process. While other methods such as encrypting configuration files, using version control, and limiting access to the Terraform state file contribute to security, they do not directly prevent sensitive data from being exposed during Terraform operations like marking a variable as sensitive does. Encrypting configuration files does offer a layer of security but doesn't inherently prevent logging of sensitive information during execution. Utilizing version control can create risks if sensitive values are committed without consideration, and limiting access to the Terraform state file is more about managing permissions than obscuring values during execution. Thus, the most effective approach within the context provided is to specifically mark variables as sensitive in the configuration.

SAMPLE

5. What is a workspace in Terraform?

A. A separate environment for different configurations

B. A way to group related resources

C. A method to store state files

D. A command for managing modules

A workspace in Terraform provides a separate environment for different configurations, allowing developers to manage multiple versions of infrastructure within the same configuration directory. Each workspace has its own state file, which means changes in one workspace do not affect the others. This is particularly useful for managing different environments such as development, staging, and production, all within a single project. By utilizing workspaces, you can easily switch between different environments while keeping the resource management organized. This feature enhances Terraform's capability to handle infrastructure as code more effectively, maintaining clear distinctions within projects and facilitating better lifecycle management of environments. The other options do not accurately reflect the definition or purpose of workspaces in Terraform. Grouping related resources refers to Terraform modules, while state file management is handled through backends, and commands for managing modules are part of Terraform's module system rather than its workspace feature.

6. How does Terraform determine changes needed for the infrastructure?

A. By analyzing logs for errors

B. By comparing local files

C. By comparing the current state to the desired state defined in the configuration

D. By checking external resource dependencies

Terraform determines changes needed for the infrastructure primarily by comparing the current state to the desired state defined in the configuration. This process involves checking the current state of the infrastructure, which Terraform manages in its state file, against the configuration files written in HashiCorp Configuration Language (HCL) or JSON that specify the desired end state. When a Terraform plan is executed, it retrieves the current state from the state file and then looks at the configuration files to understand what the infrastructure should look like. If there are any discrepancies between the current state and the desired state, Terraform generates an execution plan that outlines the necessary actions to reach the desired state. These actions can include creating, modifying, or destroying resources as needed, ensuring that the infrastructure is provisioned in accordance with the specifications set forth in the configuration files. This approach enables Terraform to manage infrastructure changes in a declarative manner, allowing infrastructure management to be consistent, predictable, and repeatable. By focusing on the desired state as defined in the configurations, Terraform can optimally make the required modifications to align the current infrastructure with what is specified, regardless of the order in which resources were created or modified.

SAMPLE

7. What does the command 'terraform state rm' do?

A. Renames an item in the state

B. Adds an item to the state

C. Remove instances from the state

D. Displays the current state file

The command 'terraform state rm' is specifically designed to remove instances or resources from the Terraform state file. This command is useful when you want to detach a resource from management by Terraform without actually destroying the resource in the underlying infrastructure. For example, a resource might be deleted from the state if you manually remove it from your infrastructure, or if you're certain you want Terraform to stop tracking that resource but don't want to delete it. By removing it from the state, Terraform no longer has control or awareness of that resource, which can be crucial in scenarios where manual changes were made or if a resource is being migrated to another management tool or process. This command provides flexibility in managing your infrastructure as code by allowing users to selectively manage which resources are tracked. In contrast, the options related to renaming items, adding items to the state, or displaying the current state file do not accurately reflect the functionality of this specific command, highlighting 'terraform state rm's unique purpose in the context of Terraform state management.

8. When Terraform crashes because Golang results in a panic, where will it log the results?

A. debug.log

B. panic.log

C. terraform.log

D. crash.log

When Terraform experiences a crash resulting from a panic in Golang, it logs the results to a file named "crash.log." This file is specifically designated to capture detailed information regarding the state of the program at the time of the crash, including stack traces and other critical data that can assist in troubleshooting. By storing this information in crash.log, Terraform allows developers and operators to review the conditions that led to the panic and to gather insights necessary for debugging. This logging mechanism is essential for maintaining stability and reliability in infrastructure management, as it provides a mechanism for diagnosing issues that may not be apparent during regular operation. The other options, while they may seem plausible, do not correspond to Terraform's behavior during a panic event. The crash.log specifically serves this purpose, ensuring that users and developers can address critical failures appropriately.

SAMPLE

9. What is the purpose of Terraform modules?

A. To indicate deprecated resources

B. To encapsulate and group related resources for reusability and organization

C. To increase Terraform configuration file size

D. To automate backups of the Terraform state

The purpose of Terraform modules is to encapsulate and group related resources for reusability and organization. By utilizing modules, you can create a set of resources that are commonly used together, thereby promoting best practices and reducing duplication in your infrastructure code. This modular approach allows teams to standardize configurations, making it easier to manage and understand complex Terraform projects. Modules enhance collaborative development as they allow different teams or individuals to share and use predefined configurations efficiently. This leads to improved maintainability, as you can update the resources within a module without having to change every instance where that module is utilized in your Terraform scripts. In contrast, the other options do not align with the actual function of Terraform modules. Indicating deprecated resources, increasing configuration file size, and automating backups of the Terraform state do not relate to the core purpose of modules, which fundamentally revolves around organizing and reusing infrastructure code effectively.

10. What is the benefit of implementing Terraform workspaces?

A. To facilitate easy coding

B. To manage different states for different environments

C. To create better documentation

D. To increase execution speed

Implementing Terraform workspaces offers the key benefit of managing different states for different environments, which is critical for maintaining organized and scalable infrastructure management. Each workspace can hold a separate state file, enabling you to work on different versions of your infrastructure simultaneously, such as development, testing, and production environments, while isolating configurations and resources relevant to each stage. This separation reduces the risk of unintended changes affecting other environments and allows teams to deploy and manage multiple infrastructure setups efficiently. By using workspaces, you can also switch between different configurations easily, which is particularly beneficial in CI/CD pipelines or during multi-environment deployments. The other choices do not directly relate to the core functionality and advantages offered by workspaces within Terraform. While coding and documentation have their own importance in the IaC practice, they are not specifically enhanced by the use of workspaces, nor does using workspaces inherently increase execution speed.

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://hashicorpterraformiac.examzify.com

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

SAMPLE