Up-to-date Practice Test with Latest Questions and Answers covering latest syllabus and topics of the exam. Makes you ready to face actual exam.

KILL EXAMS

Terraform-Associate-004 Practice Questions
Terraform-Associate-004 Practice Test
Terraform-Associate-004 Practice Exam
Terraform-Associate-004 Exam Questions
Terraform-Associate-004 Study Guide

PASS

killexams.com

HashiCorp

Terraform-Associate-004

Terraform Associate 004

ORDER FULL VERSION

https://killexams.com/pass4sure/exam-detail/Terraform-Associate-004

Question: 1247

For an AzureRM backend backend "azurerm" { resource_group_name = "rg-tfstate" storage_account_name = "tfstateprod" container_name = "tfstate" key = "terraform.tfstate" }, state locking requires no additional configuration because it uses blob leases natively.

A. True
B. False

Answer: A
Explanation: True. The azurerm backend natively implements locking via Azure Storage blob leases without extra parameters. During operations, Terraform acquires a lease on the state blob, blocking concurrent access until released, providing built-in multi-user safety.

Question: 1248

When configuring an S3 backend, the 'dynamodb_table' argument is used for state locking. In a scenario where the IAM user executing Terraform has 's3:PutObject' and 's3:GetObject' permissions but lacks DynamoDB permissions, what will be the result and behavior of Terraform? (Select 3-4)

A. The 'terraform apply' will fail with an error because it cannot write to the lock table.
B. Terraform will successfully read the state but fail to acquire a lock before an apply.
C. Terraform will automatically skip locking and proceed with the operation anyway.
D. The error message will explicitly state that the DynamoDB table is inaccessible.
E. State changes will be written to S3, but concurrent runs could occur simultaneously.

Answer: A,B,D
Explanation: If a backend is configured to use a locking mechanism like DynamoDB but the permissions are missing, Terraform will stop the operation to prevent unsafe writes. It will not "fail open" by skipping the lock; instead, it will throw a fatal error during the lock acquisition phase. The user will still be able to read the state (S3 GetObject), but the inability to write to the lock table prevents any modification command from starting. The error output will identify the permission failure on the DynamoDB resource.

Question: 1249

You are troubleshooting a failed terraform import operation. The error message is vague. Which logging configurations will help you see the exact raw response from the cloud provider's API?

A. TF_LOG_CORE=DEBUG
B. TF_LOG=TRACE
C. TF_LOG_PATH=terraform.log
D. Setting TF_LOG to a non-empty value like INFO or DEBUG
E. Enabling the log_response parameter in the provider block

Answer: B,C,D
Explanation: TRACE is the most detailed level and typically includes the body of API requests and responses. Setting the log path ensures the data is captured for review. Even lower levels like INFO or DEBUG will provide more context than the default output, which is often suppressed to keep the terminal clean.

Question: 1250

A root configuration uses module expansions with for_each on a set of VPC configurations, each calling an identical security group module from the registry. One instance requires an additional ingress rule passed via a variable. How is variable scope and evaluation handled uniquely for each expanded instance?

A. All instances share a single variable evaluation context.
B. for_each disables variable defaults in child modules.
C. Extra variables in one instance propagate to others.
D. Each module instance maintains separate variable values based on the for_each key; the child module evaluates its inputs, validations, and resources independently per instance.

Answer: D
Explanation: for_each creates distinct module instances, each receiving its own map of input variables derived from the for_each value; this per-instance scope ensures isolated evaluation of variables, resources, and outputs tailored to each key in the set or map.

Question: 1251

A lead engineer is setting up a new Terraform project and needs to ensure that the provider versions are locked to prevent breaking changes during automated CI/CD runs. Which files or blocks are essential for managing provider versions and ensuring environment consistency?

A. The version_constraints block in the provider block
B. The terraform.tfstate file
C. The .terraform.lock.hcl file
D. The required_providers block within the terraform block
E. The terraform.tfvars file

Answer: A,C,D
Explanation: Managing provider versions effectively requires the use of the required_providers block to define the allowed version ranges for each provider used in the configuration. The dependency lock file (.terraform.lock.hcl) is automatically generated or updated during initialization to record the specific

versions and hashes of the providers used, ensuring that every team member or CI runner uses the exact same provider binary. Version constraints within the configuration act as the source of truth for allowed versions during the init process.

Question: 1252

To onboard a brownfield Kubernetes cluster manually provisioned in EKS into HCP Terraform, your team creates a new workspace "eks-prod" linked to GitHub VCS, defines the aws_eks_cluster block with essential attributes, queues a plan run via the HCP UI which auto-imports the existing cluster using an import block during apply after PR merge.

A. False
B. True

Answer: A
Explanation: False. While HCP Terraform workspaces fully support import blocks in configuration for importing multiple existing resources during standard plan/apply workflows triggered by VCS events or UI queues, the import process requires explicit terraform import CLI execution or import block processing followed by terraform plan to generate config diffs, not automatic cluster detection.

Question: 1253

A team is managing a complex cloud environment and experiences "drift" where an external administrator manually changed a Security Group rule in the AWS Console. How does Terraform identify and handle this resource drift during its workflow?

A. The state file is automatically updated in real-time by cloud provider webhooks without running a command
B. The plan output highlights the differences between the current state and the desired configuration
C. The apply command updates the real-world infrastructure to match the configuration file
D. The refresh phase reads the current real-world properties of resources and updates the state file
E. The import command can be used to bring unmanaged manual changes into the configuration and state

Answer: B,C,D,E
Explanation: Terraform identifies drift by performing a refresh, which queries the cloud provider's API to see the actual current status of resources and reconciles that with the state file. Once the state is refreshed, the plan command compares that state to the code and generates a list of actions required to return the infrastructure to the desired state. Applying these changes ensures that the real-world infrastructure is brought back into alignment with the source code. If new resources were created manually, the import command allows the engineer to bring those resources under Terraform management so they are no longer considered drift.

Question: 1254

In Terraform 1.9.x, terraform state rm aws_instance.example removes the resource from state. If the real instance persists in AWS, running terraform plan shows it as a new resource to create, while terraform refresh repopulates it into state.

A. True
B. False

Answer: B
Explanation: state rm deletes the entry from state without touching infrastructure. Refresh queries providers for existing resources matching config addresses, repopulating aws_instance.example into state if found, preventing erroneous "create" plans.

Question: 1255

When a module sourced from the registry depends on another module (transitive dependency) that has its own version constraint, and the root specifies a conflicting range for the same transitive module, how does Terraform resolve the final version during init?

A. Only direct module constraints are enforced; transitive ones are not considered.
B. It fails with a version conflict error.
C. The root module's constraint takes precedence, and transitive constraints are ignored.
D. Terraform selects the highest version satisfying all constraints across the dependency graph.

Answer: D
Explanation: Terraform evaluates the entire module dependency graph and selects, for each unique module source, the newest version that satisfies every version constraint imposed by any module in the tree (direct or transitive). Conflicts that cannot be satisfied result in an error during terraform init.

Question: 1256

A company needs to import a large number of existing security groups into Terraform. They want to ensure that the import process is documented and repeatable. Which of the following are valid strategies for managing imports in modern Terraform versions?

A. Using the terraform state push command to overwrite the remote state with a local backup

B. Using the terraform state mv command to rename resources after they are imported
C. Using the -generate-config-out flag with the terraform plan command to create HCL code for imported resources
D. Using an import block within the configuration to define the mapping between ID and resource
E. Using the terraform import command in a shell script

Answer: B,C,D
Explanation: The import block introduced in newer versions allows for a declarative way to handle imports, making them part of the standard HCL code. When combined with config generation, Terraform can automatically produce the necessary resource blocks, reducing manual effort. Renaming resources via state mv after the import allows for better alignment with naming conventions without destroying the physical infrastructure.

Question: 1257

During Terraform initialization in a directory with partial state corruption from a prior failed apply, running terraform init -reconfigure followed by terraform init -migrate-state successfully migrates the state to a new Azure Blob backend while preserving all resources and automatically resolving the corruption without manual intervention.

A. True
B. False

Answer: A
Explanation: This is true because terraform init -reconfigure forces backend reconfiguration, and when combined with -migrate-state in a subsequent or chained init, Terraform automatically detects and handles partial state corruption by validating resource addresses and data during migration to the new backend like Azure Blob, ensuring continuity in Terraform 1.x workflows without requiring manual state repair.

Question: 1258

When writing Terraform configuration, you need to use a specific version of the Google Cloud (GCP) provider to avoid an issue found in the latest release. However, your configuration also uses a third-party module from the registry that has its own provider requirements. How does Terraform handle these overlapping provider version constraints?

A. If the root module requires version 4.0.0 and a child module requires >= 5.0.0, Terraform will return an error because no single version satisfies both
B. The required_providers block in the root module is the authoritative place to define the versions that will be used throughout the entire configuration
C. Terraform will install multiple versions of the same provider if different modules require different

versions, and it will use the correct version for each module
D. Terraform will attempt to find a single version of the GCP provider that satisfies all version constraints across the root module and all child modules
E. Version constraints can include operators like ~> (pessimistic constraint), >=, and <= to give Terraform a range of acceptable versions to choose from

Answer: A,B,D,E
Explanation: Terraform's provider discovery logic requires that a single version of a provider be selected for the entire configuration. It looks at all version constraints defined in the root module and any modules it calls, and searches for a version that satisfies the intersection of all those ranges. If the requirements are mutually exclusive, initialization will fail. This is why it is important for module authors to be flexible with their version constraints and for root module authors to explicitly manage versions to ensure compatibility.

Question: 1259

In HCP Terraform, to facilitate collaboration between infrastructure and application teams while maintaining separation of concerns, how should workspaces and integrations be set up for a microservices architecture?

A. Use only CLI-driven workflows without remote state
B. Put everything in one workspace with modules
C. Create projects for infrastructure and applications; use VCS integration for configuration changes, remote state data sources for cross-references, and appropriate team permissions
D. Duplicate configurations across teams

Answer: C
Explanation: Projects separate concerns logically. VCS integration drives reliable runs and collaboration. Remote state or data sources allow safe consumption of outputs. Permissions ensure teams can only modify their scope. This reflects real-world scalable HCP Terraform usage.

Question: 1260

version = ">= 2.0.0" in a reusable module allows consumers to use any 3.x version, but root modules should prefer this over ~> for maximum flexibility in upgrades.

A. False
B. True

Answer: A
Explanation: Reusable modules should use minimum constraints like >= 2.0.0 for broad compatibility,

while root modules use ~> to bound updates and avoid untested major versions.

Question: 1261

During terraform plan, several resources show as "drifted" because tags were added manually via the cloud console. The team wants to adopt a configuration-driven approach going forward and ignore minor tag differences that are managed externally. What lifecycle meta-argument can be used on affected resources to customize drift handling for specific attributes?

A. prevent_destroy = true
B. create_before_destroy = true
C. replace_triggered_by for tags
D. ignore_changes = ["tags"] (or specific tag keys) within the lifecycle block

Answer: D
Explanation: The lifecycle.ignore_changes list tells Terraform to ignore differences in specified attributes during plan/apply, preventing unnecessary updates for attributes managed outside Terraform (like certain tags). This is useful for drift scenarios involving non-critical or externally modified attributes while still managing core resource properties.

Question: 1262

An S3 backend config includes dynamodb_table = "locks" for locking. A hung apply leaves a stale lock; terraform force-unlock <ID> succeeds from the same credentials, but a different IAM role fails with permission denied on subsequent init.

A. False
B. True

Answer: B
Explanation: Force-unlock requires DynamoDB write permissions on the lock table; cross-role access demands explicit policy grants for dynamodb:DeleteItem on the lock objects, which the secondary role lacks.

Question: 1263

A company wants to use a module from a private GitHub repository. The repository is at github.com/org/terraform-aws-vpc. Which of the following are valid source strings for this module?

A. source = "git::https://github.com/org/terraform-aws-vpc.git"
B. source = "github.com/org/terraform-aws-vpc"
C. source = "org/vpc/aws"
D. source = "git@github.com:org/terraform-aws-vpc.git"
E. source = "git::ssh://git@github.com/org/terraform-aws-vpc.git"

Answer: A,B,D,E
Explanation: Terraform recognizes standard Git SSH and HTTPS URLs, as well as a shorthand for GitHub URLs. Using the git:: prefix explicitly defines the protocol, while the shorthand github.com/... is automatically inferred as a Git source. The three-part name org/vpc/aws is reserved for the Terraform Registry, not direct Git sources.

Question: 1264

When migrating modules from local paths to a Git-based source in a large repository with subdirectories, the source string changes from "./modules/vpc" to "git::https://github.com/org/repo.git//modules/vpc?ref=v1.0". What common pitfall in this transition affects how Terraform resolves further nested submodules inside the VPC module?

A. The // path separator is ignored for nested modules.
B. Git sources do not support subdirectories.
C. Version refs apply only to the top-level module.
D. Nested modules using relative paths inside the Git-sourced module resolve relative to the cloned repository root, potentially breaking if they assumed a flat local structure.

Answer: D
Explanation: In Git-sourced modules, the double-slash (//) denotes the subdirectory containing the module root; any relative sources used by nested modules inside it are resolved against the downloaded Git content's filesystem, requiring consistent path handling compared to pure local development.

Question: 1265

A backend "gcs" configuration for Google Cloud Storage with credentials set to a service account key file enables automatic state versioning by default, storing historical states in the same bucket.

A. True
B. False

Answer: A
Explanation: True. GCS backend supports object versioning if enabled on the bucket. Terraform

automatically creates versioned objects on state updates, allowing historical state retrieval via terraform state pull > versioned-state.tfstate for audits or rollbacks.

Question: 1266

A DevOps engineer is configuring Terraform for a project that interacts with both the AWS provider for EC2 instances and the Docker provider to build and push container images to a registry as part of the same deployment workflow. The configuration must ensure that provider plugins are installed correctly without conflicts, and resources reference the appropriate provider instances. How does Terraform handle and utilize multiple providers in this mixed infrastructure scenario?

A. Multiple providers require separate Terraform workspaces, with state files isolated per provider to prevent cross-provider dependency issues.
B. Each provider is declared with its own provider block (optionally using aliases), and the required_providers block lists all sources with versions; resources can specify a provider meta-argument to select a specific instance if multiple configurations of the same provider exist.
C. Terraform merges all provider configurations into a single global context during plan, ignoring any alias or source differences.
D. Providers are discovered automatically at runtime based on resource addresses; no explicit declaration is needed as long as terraform init succeeds.

Answer: C
Explanation: Each provider is declared with its own provider block (optionally using aliases), and the required_providers block lists all sources with versions; resources can specify a provider meta-argument to select a specific instance if multiple configurations of the same provider exist. Terraform downloads and executes providers as separate plugins during terraform init based on declarations in the required_providers block. It uses providers to translate resource declarations into API calls for the target systems. In configurations with multiple providers (different or aliased instances of the same), the provider meta-argument on resources or modules explicitly routes the management to the correct provider configuration, enabling seamless multi-cloud or mixed-tool workflows without runtime discovery or workspace separation.

Question: 1267

terraform destroy --exclude-types=aws_instance skips all instance destructions and their VPC dependencies, allowing selective teardown of non-excluded networking resources in a shared state.

A. True
B. False

Answer: B

Explanation: This is false because --exclude-types is not a valid flag for destroy; exclusion requires -target negation or state rm, and dependencies like VPCs would still be targeted if rooted.

Question: 1268

You need to deploy a set of resources to three different environments (Dev, QA, Prod) using the same code base but with different state files. Which of the following strategies are appropriate for this requirement using the Terraform CLI?

A. Using separate backend configuration files with -backend-config
B. Using Terraform CLI workspaces (e.g., terraform workspace new dev)
C. Creating separate directories with identical code files
D. Using separate variable files (e.g., dev.tfvars, prod.tfvars)
E. Using the count parameter on every resource to toggle environments

Answer: A,B,D
Explanation: Managing multiple environments with the same code is best achieved through separation of state and variables. Terraform CLI workspaces allow you to maintain multiple state files for a single configuration in the same directory. Alternatively, you can use the same configuration but initialize it with different backend configurations using the -backend-config flag to point to different storage locations. In both scenarios, environment-specific values are typically provided using different .tfvars files or environment variables.

Question: 1269

A large organization uses HCP Terraform with multiple projects and workspaces. One project contains shared networking modules consumed by application workspaces across teams. During a planned migration, the team needs to enforce that all consuming workspaces pull the latest approved module version from the private registry while preventing direct edits to the module source in individual workspace configurations. What HCP Terraform feature combination best achieves this governance?

A. Use variable sets at the project level combined with run triggers on the shared module workspace and Sentinel policies checking module sources
B. Set workspace permissions to read-only for non-owners and rely on VCS-driven runs only
C. Configure ephemeral variable sets scoped to the consuming workspaces
D. Enable health assessments on workspaces and configure change requests for module updates

Answer: A
Explanation: Variable sets defined at the project level propagate consistently to child workspaces, while run triggers automatically queue runs in consuming workspaces when the shared module workspace completes

successfully. Sentinel (or OPA) policies can then enforce that module sources point only to the private registry with pinned versions, providing governance over multi-team collaboration in HCP Terraform.

Question: 1270

You are building a module that needs to output the private IP addresses of a cluster of virtual machines. The number of machines is dynamic based on an input variable. Which of the following are necessary or valid steps to implement this output?

A. Defining an output block in the child module.
B. Mapping the child module's output to a resource in the root module.
C. Declaring the output as sensitive = true if the IPs are considered internal data.
D. Accessing the output in the root module using the module.<NAME>.<OUTPUT_NAME> syntax.
E. Using a for expression in the output's value to iterate over the instances.

Answer: A,C,D,E
Explanation: Outputs are the only way to expose internal module data. For multiple resources, a for loop is often needed to gather all attributes into a list or map. Marking an output as sensitive prevents it from being logged in the CLI (though it remains in the state). While these outputs can be used as inputs for other resources, they are not "mapped to a resource" directly; they are accessed by the root module using the specific module namespace syntax.

KILLEXAMS.COM

Killexams.com is a leading online platform specializing in high-quality certification exam preparation. Offering a robust suite of tools, including Exam Questions, practice tests, and advanced test engines, Killexams.com empowers candidates to excel in their certification exams. Discover the key features that make Killexams.com the go-to choice for exam success.

Practice Exam Questions Based on Current Exam Objectives

Killexams.com provides practice exam questions aligned with the latest official exam objectives and latest syllabus. Our content is reviewed and updated regularly to reflect recent changes announced by certification vendors. By studying these practice questions, candidates will cover the structure, difficulty level, and topics of the actual exam, helping them prepare more effectively and efficiently.

Comprehensive Practice Exams (PDF Format)

Killexams.com offers multiple-choice questions (MCQs) in easy-to-read PDF format, covering all major domains of the exam. Each PDF contains a structured collection of practice questions and verified answers designed to support focused study. These MCQs help candidates reinforce key concepts, identify knowledge gaps, and improve exam readiness through consistent practice.

Realistic Practice Tests (Online Test Engine & Desktop Test Engine)

To support hands-on preparation, Killexams.com provides practice tests through both an Online Test Engine and a Desktop Test Engine. These tools are designed to simulate a real exam environment, allowing candidates to practice under exam-like conditions, with latest syllbus and topics of the exam. Performance tracking, test history, and result analysis help users evaluate their progress and focus on areas that need improvement.

Risk-Free Purchase Policy

Killexams.com follows a transparent and customer-friendly purchase policy. If users are not satisfied with the study materials, they may request assistance or a refund in accordance with our published terms and conditions. This policy reflects our commitment to customer satisfaction, fairness, and confidence in our preparation resources.

Regularly Updated Content

Our practice question bank is reviewed and updated on an ongoing basis to stay aligned with the latest exam outlines and vendor updates. This ensures candidates are studying up-to-date, relevant material, and preparing with content that reflects current exam expectations, helping them stay confident and well-prepared.