perplexity

Ultimate Challenge: 60 Difficult HashiCorp Certified Terraform Associate Practice Questions

Terraform 1.8 - Exam coverage as of July 2025

Prepare to tackle sixty scenario-focused, higher-difficulty questions that probe deep understanding of Terraform language, functions, state management, meta-arguments, provider nuances, and Terraform Cloud workflows. Each item mirrors the complexity and wording style of the toughest items reported by recent test-takers while remaining fully within HashiCorp's public-domain blueprint.

How to Use These Questions

Read each scenario carefully, attempt an answer before checking the key, and then review the inline citations to locate official documentation for a refresher. Re-typing snippets in `terraform console` reinforces syntactic subtleties.

Tough Questions by Domain

Core HCL & Syntax Mastery

1. **A module input variable is declared with no default value and the following type constraint:**

   ```
   variable "team_map" {
     type = map(object({
       name    = string
       members = set(string)
       cost    = optional(number, 42)
     }))
   }
   ```

   Which CLI command will *fail* even when `-var-file=teams.tfvars` supplies valid JSON, and *why*?
   A. `terraform fmt` – type checking occurs during formatting.
   B. `terraform validate` – optional attributes are always ignored.
   C. `terraform plan -var team_map={}` – empty map violates object attribute rules.
   D. `terraform console` – console cannot evaluate complex types.
   Answer: C – an empty map violates the object's non-optional attributes at planning time[1] [2].

2. **Given the local value below, which expression correctly returns `"10.1.2.0/27"`?**

   ```
   locals {
     base_cidr = "10.1.0.0/22"
   ```

```
}
```

A. `cidrsubnet(local.base_cidr, 5, 4)`
B. `cidrsubnet(local.base_cidr, 3, 16)`
C. `cidrsubnet(local.base_cidr, 8, 2)`
D. `cidrsubnet(local.base_cidr, 2, 8)`
Answer: A – a new subnet mask of `/27` (original `/22` plus 5) and network index 4 yields the target range[3][2].

3. **Why does the following attempt to compute an MD5 checksum fail at `terraform plan`?**

```
resource "null_resource" "checksum" {
  triggers = {
    md5 = md5(file("${path.module}/init.sh"))
  }
}
```

A. `md5()` cannot accept file objects.
B. `file()` marks output sensitive; triggers cannot be sensitive.
C. The file is read at *apply* time; triggers need values during *plan*.
D. `md5()` is deprecated in Terraform 1.8.
Answer: C – file contents are unknown until apply, causing non-static trigger evaluation to fail[4][5].

4. **Which statement about provider-defined functions introduced in Terraform 1.8 is *true*?**
A. They can be authored in any language compliant with WASM.
B. They are invoked with a double colon prefix (`aws::cidrsubnet`).
C. They may be disabled via `provider "aws" { functions = false }`.
D. They share the same namespace as data sources, so naming collisions are disallowed.
Answer: D – provider-defined functions occupy the provider namespace and cannot clash with data/resource names[5].

5. **A `for` expression must produce a tuple when…**
A. brackets `[]` enclose the expression and no `:` token exists.
B. an `=>` token is present.
C. the iterator name is omitted.
D. the collection source is a map.
Answer: A – list/tuple result comes from `[ for ... ]` without key selector[6].

6. **In Terraform 0.12+ interpolation syntax, which two of the following are *functionally identical*?**

    1. `local.tags["env"]`
    2. `${local.tags["env"]}`
    3. `lookup(local.tags, "env", null)`
    4. `${lookup(local.tags,"env","")}`

    Answer: 1 & 2 – both exact attribute syntax, whereas `lookup` introduces a default argument difference[2].

7. **What is printed by `terraform console` for `jsondecode("[true, [false]]")[7]`?**
   A. `true`
   B. `false`
   C. `0`
   D. Error – nested list unsupported.
   Answer: B – the second element is `[false]`, index 0 gives `false`[3].

8. **Which direct reference is invalid inside a `dynamic "ingress"` block?**
   A. `ingress.value.description`
   B. `each.key`
   C. `var.port_list[count.index]`
   D. `outer_block.value.port`
   Answer: C – `count.index` is unavailable inside dynamic for-each constructs; use `ingress.key` instead[8][9].

9. **Why does the plan below produce `Force replacement` rather than an in-place update?**

```
resource "aws_launch_template" "app" {
  name_prefix = "blue-"

  lifecycle {
    create_before_destroy = true
  }

  block_device_mappings {
    ebs {
      volume_type = var.type
    }
  }
}
```

   A. `volume_type` is immutable in AWS; Terraform uses destroy/create[10].
   B. `create_before_destroy` always triggers replacement.
   C. Launch templates cannot update `name_prefix`.
   D. The provider sets `ForceNew` on any block attribute changes.
   Answer: D – AWS provider marks nested launch template blocks with `ForceNew`; Terraform must replace[10].

10. **Which CLI option speeds up a large workspace refresh when every resource has stable remote state?**
    A. `terraform plan -replace="*" -refresh=true`
    B. `terraform plan -destroy -parallelism=1`
    C. `terraform plan -refresh=false`
    D. `terraform init -upgrade`
    Answer: C – skipping refresh avoids remote reads yet still performs validation[10][11].

Meta-Arguments, Lifecycle & Dependency Traps

11. **A security team wants a critical production RDS instance blocked from deletion. Which configuration meets the goal *and* still lets engineers rename tags?**

```
resource "aws_db_instance" "prod" {
  # ...snip...
  tags = var.tags

  lifecycle {
    __?__
  }
}
```

    A. `prevent_destroy = true`
    B. `ignore_changes = ["*"]`
    C. `prevent_destroy = true`, `ignore_changes = [tags]`
    D. `prevent_destroy = true`, `ignore_changes = [parameter_group_name]`
    Answer: C – prevents deletion while allowing tag drift[12][13][14].

12. `replace_triggered_by` **will *not* trigger a replacement when the referenced entity is…**
    A. a single attribute of a resource instance.
    B. a `null_resource`.
    C. a module output value.
    D. any resource with multiple instances.
    Answer: C – variables, data sources, modules are unsupported because they lack planned actions[15][16].

13. **Which code forces an** `aws_instance` **to recreate whenever its associated** `aws_security_group`**'s inbound rules change, without modifying user-data?**
    A. Add `depends_on = [aws_security_group.web]`.
    B. Add `replace_triggered_by = [aws_security_group.web]`.
    C. Add `triggers = { sg = aws_security_group.web.id }`.
    D. Add `ignore_changes = [vpc_security_group_ids]`.
    Answer: B – leverages lifecycle replacement based on dependency changes[17][15].

14. **Why does adding** `prevent_destroy = true` *not* **block removal when the resource is commented out of configuration?**
    A. Prevent destroy only checks drift, not deletion via config removal.
    B. Terraform forgets lifecycle rules once the block is absent.
    C. Provider overrides lifecycle during graph walk.
    D. You must also set `create_before_destroy = false` for it to work.
    Answer: B – lifecycle meta data lives in config; once absent Terraform cannot honor it[14].

15. **Select the correct syntax to ignore *all* future attribute drift for an S3 bucket while still allowing** `terraform destroy` **to work.**
    A. `ignore_changes = ["*"]`
    B. `ignore_changes = all`
    C. `ignore_changes = true`
    D. Impossible; ignore implies lock-in.

Answer: B – the special keyword `all` ignores every attribute but still permits deletes when requested[18][19].

16. **A team embeds the snippet below and later adds an EC2 capacity provider to the cluster via the AWS console. What will `terraform apply` do?**

```
resource "aws_ecs_cluster" "main" {
  name = "blue"

  lifecycle {
    ignore_changes = [capacity_providers]  # added previously
  }
}
```

A. Remove the console-added capacity provider.
B. Fail with drift error; ignore doesn't cover lists.
C. Leave the new capacity provider untouched.
D. Force recreation of the cluster.
Answer: C – ignore_changes suppresses Terraform updates for that attribute[12][19].

17. **Which scenario is *appropriate* for using `create_before_destroy = true` to guarantee zero downtime?**
A. Rotating an RDS instance storage type from gp2 to gp3.
B. Replacing an IAM role assumed by running Lambda functions.
C. Updating an ASG launch template ID.
D. Switching a Route 53 record `type` from `A` to `AAAA`.
Answer: C – new launch template plus detached old one achieves seamless rollout; many AWS resources reject identical names, so create before destroy is required[20][21].

18. **When do precondition blocks execute relative to `terraform import`?**
A. Always before reading remote objects, so import may fail.
B. Only after import because no resource has a prior state.
C. They are skipped; import bypasses pre/post conditions.
D. Precondition must reference `self`, so import fails.
Answer: C – import bypasses plan/apply evaluation and thus condition blocks[22][1].

19. **Which postcondition example will *prevent* dependent resources from provisioning if a newly created S3 bucket lacks server-side encryption?**

```
resource "aws_s3_bucket" "secure" {
  bucket = "secure-${random_id.suffix.hex}"
  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "aws:kms"
      }
    }
  }

  lifecycle {
    postcondition {
      condition     = self.server_side_encryption_configuration[*].rule[0].apply_serv
      error_message = "SSE-KMS must remain enabled."
```

```
    }
  }
}
```

A. Valid as is.
B. Must reference `data.aws_s3_bucket.secure` instead of `self`.
C. Need `precondition`, not `postcondition`.
D. Key needs `` index outside splat.
Answer: A – correct structure and will block dependent graph nodes if SSE disabled[1][23].

20. **Which characteristic applies to both `precondition` and `variable validation` blocks?**
A. They may reference `each.key`.
B. They can only compare primitive types.
C. They display `error_message` on failure and halt planning.
D. They run after remote API calls.
Answer: C – both provide custom messages and halt plan on `false`[22][24][1].

Functions, Encoding & Collection Wizardry

21. **Select the expression that rounds *up* the floating-point variable `weight` to the next multiple of 5.**
A. `ceil(weight, 5)`
B. `ceil(weight / 5) * 5`
C. `floor(weight / 5) * 5`
D. `max(weight, 5)`
Answer: B – Terraform lacks multi-argument `ceil`; dividing then multiplying achieves rounding[2][3].

22. **A colleague attempts `merge(list1,list2)` and receives an error. What is the root cause?**
A. `merge()` only accepts maps.
B. `list1` is an unknown value.
C. `merge()` was removed in 1.5.
D. Lists must be converted via `toset()` first.
Answer: A – merge concatenates maps; for lists use `concat()`[2].

23. **To generate an HCL map of tag keys to uppercase values, which `for` expression is correct?**

```
locals {
  tags_upper = { for k,v in var.tags : k => upper(v) }
}
```

A. Valid as written.
B. Must wrap result in `tomap()`.
C. Should use `toset()`.
D. Needs `merge()` not `for`.
Answer: A – map for-expression syntax correct[2][6].

24. **Identify the *only* encoding function that can safely transform a base64 string back to its original binary form for use in an `archive_file` data source.**

A. `base64gzip()`
B. `base64decode()`
C. `base64sha256()`
D. `gzipdecode()`
Answer: B – decode returns raw bytes suitable for archiving[3].

25. **What does `templatefile("${path.module}/user_data.tpl", { uid = uuid() })` return at *plan* time?**
A. Rendered template with runtime UUID.
B. `null` – file read deferred.
C. Runtime error – functions cannot be nested.
D. Unknown value rendered at *apply* because `uuid()` changes each run.
Answer: D – `uuid()` yields non-deterministic value known only during apply; console shows `(known after apply)`[3].

26. **Given `setproduct(["us","eu"],["prod","dev"])`, what is the length of the resulting list?**
A. 2
B. 3
C. 4
D. 8
Answer: C – Cartesian product 2×2=4 elements[2].

27. **Which two functions can convert an *unknown* JSON string produced by `external` data source into a fully typed Terraform map?**
A. `jsondecode()` then `tomap()`
B. `yamldecode()` then `toset()`
C. `abspath()` then `tonumber()`
D. `cidrhost()` then `jsonencode()`
Answer: A – decode plus typing map[3].

28. **What does `zipmap(tolist(keys(var.map)), values(var.map))` guarantee?**
A. Reverses key–value mapping order.
B. Removes any duplicate values.
C. Produces list of tuples.
D. Produces deterministic order independent of original map.
Answer: D – `zipmap` with explicit key list stabilizes ordering[2].

29. **Why is `toset([1,true,"a"])` invalid?**
A. Sets cannot contain booleans.
B. Mixed element types violate set homogeneity.
C. `toset` requires a tuple.
D. Numbers must be quoted.
Answer: B – sets require uniform primitive type[2].

30. **Choose the expression that extracts the last segment `"bar"` from `var.arn = "arn:aws:s3:::foo/bar"` regardless of prefix depth.**
A. `split("/",var.arn)[-1]`
B. `element(split("/",var.arn), length(split("/",var.arn))-1)`
C. `regex("\\/([^\\/]+)$", var.arn)`

D. Both A and B are valid; C requires provider-defined regex.
Answer: D – negative index or element/length both work; `regex()` returns list not string[2].

Remote State, Backends & Workspaces

31. **Which backend attribute must be *omitted* when migrating an existing S3 backend to use DynamoDB locking?**
    A. `bucket`
    B. `encrypt`
    C. `workspace_key_prefix`
    D. `key`
    Answer: C – workspace prefix is incompatible with new table-based locking schema[2].

32. **In Terraform Cloud, a workspace using the VCS workflow is set to "queue plans manually". What happens if two pull requests modify the same file?**
    A. The second PR auto-cancels the first queued run.
    B. Both speculative plans execute; conflicts resolved at merge.
    C. Terraform Cloud locks the statefile and rejects the second commit.
    D. Only the default branch run can lock remote state.
    Answer: B – speculative runs work in isolation; true lock occurs only when merged[2].

33. **Which command promotes a `remote` state workspace named `test` to the default without editing `backend.tf`?**
    A. `terraform workspace select default`
    B. `terraform state push test`
    C. Not possible; backend block is authoritative.
    D. `terraform workspace delete default && terraform workspace new default`
    Answer: C – backend config determines default workspace; cannot promote via CLI alone[2].

34. **Why does `terraform state rm module.web.null_resource.delay` refuse when the state is stored in Terraform Cloud?**
    A. `state rm` is unsupported with remote backends.
    B. Feature flag `remote_ops` must be enabled.
    C. Must run `tfrun` wrapper.
    D. Requires `TFC_WORKSPACE` env var.
    Answer: A – remote state modifications need `terraform state pull` / push or TFC UI[2].

35. **A team sets the backend `workspace_key_prefix = "env"` and selects workspace `prod`. What is the final S3 object key if `key = "app.tfstate"`?**
    A. `env/prod/app.tfstate`
    B. `prod/env/app.tfstate`
    C. `env/app.tfstate`
    D. `prod/app.tfstate`
    Answer: A – workspace prefix inserted before workspace and key[2].

Providers, Modules & Registry Nuances

36. **What is the *only* acceptable way to pin an AWS provider to the latest patch of 5.78.x?**
    A. `required_version = "= 5.78.*"`
    B. `version = "~> 5.78.0"`
    C. `version = ">=5.78, <5.79"`
    D. Both B and C work.
    Answer: C – explicit >= and < avoids tilde lim- itations; `~>` cannot fix patch wildcard[2].

37. **Why can't a child module output sensitive value `db_password` be referenced directly in a parent's `locals` block?**
    A. Outputs marked sensitive are hidden until apply.
    B. Parent modules cannot read outputs.
    C. Sensitive values require `nonsensitive()` wrapper.
    D. Must use data source interpolation.
    Answer: C – need `nonsensitive()` to coerce inside locals[2].

38. **A versioned module tag `v2025-07-23` fails `terraform init` when declared in `source` argument. Why?**
    A. Hyphenated tags unsupported.
    B. Non-semantic versions require `?ref=` syntax.
    C. Registry must host *x.y.z* tags only.
    D. The provider lock file overrides tag.
    Answer: B – non-semver tags need explicit `?ref=` pointer[2].

39. **When publishing a public module, which file is mandatory in the root for Registry discovery?**
    A. `variables.tf`
    B. `outputs.tf`
    C. `README.md`
    D. `.terraform.lock.hcl`
    Answer: C – README triggers registry UI listing[2].

40. **What is the primary reason to set `module.example.sensitive_output` to `true` in `outputs.tf`?**
    A. Prevent output from being displayed in CLI and state show.
    B. Encrypt output inside state file.
    C. Block module consumers from reading variable.
    D. Skip output during `terraform refresh`.
    Answer: A – sensitive flag redacts console/UI but state remains plaintext[2].

State CLI, Drift & Advanced Manipulations

41. **Which `terraform import` syntax correctly maps two CloudFront distributions into a single `aws_cloudfront_distribution` resource using for-each?**

```
resource "aws_cloudfront_distribution" "dist" {
  for_each = toset(["E123","E456"])
  # ...
}
```

A. `terraform import aws_cloudfront_distribution.dist E123`
B. `terraform import aws_cloudfront_distribution.dist E123` twice
C. `terraform import 'aws_cloudfront_distribution.dist["E123"]' E123`
D. Not possible; must separate blocks.
Answer: C – for-each index requires quoted key[2][6].

42. **After running `terraform taint module.db.aws_db_instance.primary`, what happens on the next plan when `prevent_destroy = true` is set?**
A. Plan fails; taint implies destroy which is blocked.
B. Taint converts to in-place update.
C. Taint ignored; resource stays.
D. Provider forces new resource only if `skip_final_snapshot = true`.
Answer: A – taint schedules replacement; prevent_destroy stops it[14][13].

43. **Why does `terraform state mv` require editing the configuration afterward?**
A. The move is ephemeral; next plan reverts.
B. Graph builder detects mismatch and deletes resource.
C. State now differs from config and will drift on next apply.
D. Providers include resource address checksum.
Answer: C – state must match resource addresses defined in HCL or they'll be removed[2].

44. **What behavior changed in Terraform 1.3 regarding the `-lock` flag?**
A. Default switched to `-lock=false`.
B. CLI now retries locks for 20 minutes by default.
C. `terraform destroy` overrides `-lock=false`.
D. Remote backends ignore the flag entirely.
Answer: B – default retry length increased; documented in release notes[2].

Pre/Post Conditions, Null Resource & terraform_data

45. **Which two resource types remain after HashiCorp announced deprecation of `null_resource`?**
A. `terraform_data` (core), `local_file` (provider)
B. `null_resource` replaced by `terraform_null`
C. Only `terraform_data`
D. `external` data source upgrades automatically
Answer: C – `terraform_data` supersedes `null_resource` as upgrade path[4][15].

46. **The following `terraform_data` block is intended to hash a script for a trigger, but always appears as `updates in-place`. Pick the fix.**

```
resource "terraform_data" "hash" {
  input = filesha256("${path.module}/user.sh")
}
```

A. Wrap in `nonsensitive()`.
B. Use attribute `triggers_replace`.
C. Use `lifecycle { replace_triggered_by = null }`.
D. Set `timeouts { create = "0m" }`.
Answer: B – `triggers_replace` holds value and triggers recreation on change[4][15].

47. **Which limitation still applies to `null_resource` despite using `triggers`?**
    A. Cannot depend on other resources via `depends_on`.
    B. Always treated as `tainted` on every plan.
    C. Provisioners execute only during create, never during replace.
    D. Lacks arguments for `timeouts`.
    Answer: C – replace does not rerun provisioner after initial create[25][26].

48. **Why is using `resource_meta` constructs safer than bash `local-exec` for data generation?**
    A. Meta arguments produce zero side effects outside state.
    B. Local-exec cannot produce outputs.
    C. Statefile size limited to 4 MiB for external commands.
    D. Local-exec runs on Terraform Cloud workers only.
    Answer: A – encapsulate effects; local-exec may drift[27][25].

Dynamic Blocks, Loops & Advanced Collections

49. **Choose the correct iterator override for multi-level dynamic blocks to avoid name shadowing.**

```
dynamic "listener" {
  for_each = var.listeners
  iterator = l
  content {
    port = l.value.port
  }
}
```

    A. Valid; iterator overrides default.
    B. Must set `iterator = listener`.
    C. Iterator allowed only on nested dynamic.
    D. `iterator` attribute deprecated.
    Answer: A – custom iterator prevents ambiguity[8][28][29].

50. **What is the outcome of omitting `for_each` inside a `dynamic` block?**
    A. Terraform duplicates the block for every element of `each.value`.
    B. The configuration is invalid and fails syntax check.
    C. A single block renders with computed values.
    D. Terraform silently ignores the dynamic block.
    Answer: B – `for_each` mandatory[8].

51. **Which list comprehension removes `null` entries from `var.ips`?**
    A. `[for ip in var.ips : ip if ip != null]`
    B. `[for ip in var.ips : ip if ip]`
    C. `compact(var.ips)`
    D. Both A and C.
    Answer: D – comprehension with `if` or built-in `compact()`[2].

52. **In nested `for` expressions, how do you avoid collision with `count.index` when mixing `for_each` and `count` resources?**
    A. Use `count = length(...)` then refer to `each.key`.

B. Use `for ip,key in ...`.
C. Set `for_each` on parent and reference `each.value`.
D. Encapsulate expression in separate module.
Answer: D – module encapsulation isolates meta-arg variable namespaces[6][2].

53. **Which expression merges a list of maps into one map while giving precedence to later items?**
A. `merge([maps]...)`
B. `reduce(var.list, {}, merge)`
C. `merge({for idx,v in var.list : idx => v}...)`
D. `merge(values(var.list))`
Answer: B – `reduce` iteratively merges maintaining later overwrite[2].

54. **Why does `terraform console` output `[sensitive]` after evaluating `local.secret` even without `sensitive(true)` annotation?**
A. Any value returned from `random_password` inherently sensitive.
B. Console hides any string > 16 chars.
C. `terraform console` redacts values marked sensitive in state.
D. HashiCorp Cloud policies enforce redaction.
Answer: A – random provider marks attribute sensitive; console inherits[2].

55. **What is the maximum number of nested `dynamic` levels officially supported?**
A. 1
B. 2
C. Unlimited; limited only by provider schema.
D. 4
Answer: C – dynamic recursion limited only by resource schema depth[8].

Terraform Cloud & Sentinel Guards

56. **Which Sentinel rule ensures cost estimates block a plan if the delta exceeds USD 500?**
A. `import "tfplan"`
B. `import "cost"`
C. `main = rule { tfplan.cost_changes.over_threshold }`
D. `main = rule { cost.delta <= 500 }`
Answer: B – sentinel-cost import provides `run_cost_estimate` object[2].

57. **Why might a run in Terraform Cloud stay in "Queued" despite available agents?**
A. Variable set pending approval.
B. An active run holds the global workspace lock.
C. The workspace is set to local execution mode.
D. SSH key missing for private module source.
Answer: B – only one run per workspace can hold lock[2].

58. **Which environment variable allows overriding Terraform binary version used by the Terraform Cloud agent?**
A. `TERRAFORM_VERSION`
B. `TF_CLI_VERSION`
C. `TFC_TERRAFORM_VERSION`

D. Not supported.
Answer: C – agent uses TFC_ prefix[2].

59. **When using run tasks, which exit code signals "soft-fail" (allow plan) versus "hard-fail" (reject run)?**
A. 0 vs 1
B. 101 vs 1
C. 2 vs 3
D. 0 vs 2
Answer: B – 101 soft-fail, 1 hard-fail[2].

60. **A Sentinel policy sets `tfplan.restricted_resources` but it always fails after moving to Terraform 1.8. What changed?**
A. JSON plan output switched from `planned_values` to `planned`.
B. Sentinel uses v3 plan representation; attribute renamed.
C. Resource addresses now relative to module root only.
D. No change; bug due to provider upgrade.
Answer: B – v3 plan in Terraform 1.5+ altered sentinel import paths[2].

Key Takeaways

- Tough questions test not only syntax but *behavioral edge cases* such as lifecycle pitfalls, pre/post conditions, and Terraform Cloud operational nuances.
- Recreate each answer in `terraform console` or a throw-away workspace to cement understanding.
- Consult the cited docs to confirm answers and explore contextual examples. Continual reference to primary documentation is essential for passing the Associate exam and mastering Terraform in production environments.