Terraform Infrastructure as Code Quiz (100 Questions)

DevOps Learning Module

This quiz covers fundamental concepts related to Terraform (Infrastructure as Code). Choose the best answer for each question.

1. What is the primary purpose of Terraform?

   A. To configure software on existing servers (like Ansible).

   B. To build container images (like Docker).

   C. To provision and manage infrastructure as code (IaC) in a declarative way.

   D. To manage CI/CD pipelines (like Jenkins).

   Answer: C
   Explanation: Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. It uses a declarative language to describe your desired state.

2. What command must be run first in a new or cloned Terraform directory?

   A. `terraform plan`

   B. `terraform apply`

   C. `terraform init`

   D. `terraform validate`

   Answer: C
   Explanation: `terraform init` initializes the working directory. It downloads the required provider plugins and sets up the backend.

3. What is the purpose of the `terraform plan` command?

   A. It creates a "dry-run" execution plan, showing what changes will be made, without actually making them.

   B. It applies the changes to the infrastructure.

   C. It destroys the infrastructure.

   D. It checks the syntax of the configuration files.

   Answer: A
   Explanation: `plan` is a crucial safety step. It compares the desired state (your code) to the current state (in the `.tfstate` file) and shows you the plan (create, update, destroy).

4. What command is used to apply the changes to the infrastructure?

A. `terraform push`

B. `terraform run`

C. `terraform build`

D. `terraform apply`

Answer: D
Explanation: `terraform apply` executes the actions proposed in the plan. It will ask for confirmation ("yes") before proceeding unless you use `-auto-approve`.

5. What is the "state file" (`terraform.tfstate`)?

A. A file that defines your variables.

B. A file (usually JSON) that stores the "state" of the infrastructure managed by Terraform.

C. A file that defines your providers.

D. A log file.

Answer: B
Explanation: The state file is critical. It's how Terraform maps the resources in your code to the real-world resources (e.g., EC2 instance ID, S3 bucket name).

6. Why is it critical to store the `.tfstate` file in a remote, shared, and locked backend (like S3) when working in a team?

A. To make `plan` run faster.

B. To prevent two people from running `apply` at the same time (state locking) and to have a single source of truth.

C. To automatically validate the state file.

D. It is not critical; it should be in Git.

Answer: B
Explanation: A remote backend provides state locking to prevent conflicts and ensures everyone on the team is working with the same, most up-to-date state.

7. What command is used to destroy all infrastructure managed by the current Terraform configuration?

A. `terraform delete`

B. `terraform rm -rf`

C. `terraform destroy`

D. `terraform apply -destroy`

Answer: C
Explanation: `terraform destroy` generates a plan to destroy all resources and, after confirmation, executes it.

8. What is a "Provider" in Terraform?

A. The `terraform.tfstate` file.

B. A plugin that interfaces with an API (e.g., AWS, Azure, GCP, GitHub) to manage its resources.

C. A variable.

D. A module.

Answer: B
Explanation: The provider (e.g., `hashicorp/aws`) is the plugin that understands the API of a specific service and makes its resources available to Terraform.

9. What is a "Resource" block?

A. A block that defines a variable.

B. The main block that defines a piece of infrastructure (e.g., `resource "aws_instance" "web" { ... }`).

C. A block that defines a provider.

D. A block that defines an output.

Answer: B
Explanation: The `resource` block is the most important. It defines one infrastructure object (like an `aws_instance`) and its configuration.

10. In `resource "aws_instance" "web"`, what is `"web"`?

A. The type of resource.

B. The name of the instance in AWS.

C. The local name (identifier) used to refer to this resource within Terraform.

D. The provider.

Answer: C
Explanation: This is the "local name." `"aws_instance"` is the *type*. You use the local name to reference this resource (e.g., `aws_instance.web.id`).

11. What is a "Module" in Terraform?

A. A file with the `.tfm` extension.

B. A set of Terraform configuration files (`.tf`) in a single directory, used for reusable components.

C. A provider.

D. A state file.

Answer: B
Explanation: Modules are the primary way to package and reuse Terraform code. A simple module might create a "VPC," and a complex one might create an entire "Kubernetes Cluster."

12. What is the "root module"?

A. The main provider module.

B. The `main.tf` file.

C. The top-level directory of `.tf` files where you run `terraform apply`.

D. The module that manages `root` user permissions.

Answer: C
Explanation: The "root module" is simply the directory of `.tf` files that you are currently working in.

13. How do you use a module from the Terraform Registry?

A. `module "my_vpc" { source = "terraform-aws-modules/vpc/aws" }`

B. `import "terraform-aws-modules/vpc/aws"`

C. `provider "my_vpc" { ... }`

D. `resource "module" "my_vpc" { ... }`

Answer: A
Explanation: The `module` block is used to call a child module. The `source` attribute points to its location (Registry, Git, local path).

14. What is the purpose of a `variable` block?

A. To declare an input variable (e.g., `variable "aws_region" { ... }`).

B. To declare an output value.

C. To declare a local variable.

D. To set the value of a variable.

Answer: A
Explanation: The `variable` block *defines* the input, including its type, description, and default value. It does *not* set the value (that's done via `.tfvars`).

15. What is the purpose of an `output` block?

A. To define an input variable.

B. To print a value to the console during `plan`.

C. To declare a value that will be displayed to the user after `apply` (e.g., a server's IP address).

D. To send output to a log file.

Answer: C
Explanation: `output` blocks are used to export structured data from your configuration. These are shown at the end of an `apply` and can be queried.

16. What is a `.tfvars` file?

A. A file that defines variables.

B. A file used to *set the values* for input variables (e.g., `aws_region = "us-east-1"`).

C. A file that stores the state.

D. A file that stores provider configurations.

Answer: B
Explanation: This is how you pass values *into* your configuration. Terraform automatically loads `terraform.tfvars` or `*.auto.tfvars`.

17. How do you reference the value of an input variable named `aws_region`?

A. `$var.aws_region`

B. `var.aws_region`

C. `aws_region`

D. `variable.aws_region`

Answer: B
Explanation: All input variables are accessed using the `var.` prefix (e.g., `region = var.aws_region`).

18. How do you reference the `id` attribute of `resource "aws_instance" "web"`?

A. `aws_instance.web.id`

B. `resource.aws_instance.web.id`

C. `web.id`

D. `var.web.id`

Answer: A
Explanation: The syntax is `<resource_type>.<local_name>.<attribute>`.

19. What is a "local value" (`locals` block)?

A. The same as an input variable.

B. A variable that is set on the local machine.

C. A named, intermediate variable used within a module to simplify complex expressions.

D. A variable that is not stored in the state file.

Answer: C
Explanation: The `locals { ... }` block is for creating intermediate, "local" variables. They are not set by the user (like `variable`) but are used to make code cleaner (DRY).

20. What is the `terraform.tfvars` file?

A. A special `.tfvars` file that Terraform *automatically* loads if it exists.

B. A file that must be loaded manually with `-var-file`.

C. The state file.

D. The file that defines variables.

Answer: A
Explanation: Terraform will automatically load values from any file named `terraform.tfvars` or `*.auto.tfvars`.

21. What does the `terraform validate` command do?

    A. It checks the syntax and internal consistency of the configuration files.

    B. It checks connectivity to the cloud provider.

    C. It runs a `plan`.

    D. It validates the `.tfstate` file.

    Answer: A
    Explanation: This is a static check. It's very fast and useful to run in a CI pipeline to catch syntax errors before running a `plan`.

22. What does the `terraform fmt` command do?

    A. It formats the output of `plan`.

    B. It formats the state file.

    C. It rewrites your Terraform configuration files to a standard, canonical format and style.

    D. It "forces merge" (fmt) of a branch.

    Answer: C
    Explanation: This command (`fmt`) is used to enforce a standard code style, ensuring consistent spacing, indentation, and alignment.

23. What is a "backend" in Terraform?

    A. The cloud provider.

    B. The `terraform` binary.

    C. The configuration that defines *where* Terraform's state file is stored.

    D. A provider plugin.

    Answer: C
    Explanation: The backend (e.g., `backend "s3" { ... }`) configures Terraform to store its state file remotely (like in an S3 bucket) instead of on the local disk.

24. What is "state locking"?

    A. A feature of remote backends that prevents two people from running `apply` at the same time.

    B. A command to make the state file read-only.

    C. An error when the state file is corrupt.

    D. A feature of `git`.

Answer: A
Explanation: When you run `apply`, Terraform "locks" the state. If another person tries to run `apply` at the same time, they will get an error, preventing state corruption.

25. What is a "provisioner" (e.g., `remote-exec`) in Terraform?

    A. The main provider.

    B. A tool to run commands (like configuration scripts) on a resource *after* it is created.

    C. A way to provision the state file.

    D. A type of module.

    Answer: B
    Explanation: Provisioners (like `remote-exec` to run a script via SSH) are a "last resort." They are not declarative and should be avoided in favor of tools like Ansible or cloud-init.

26. Why are provisioners generally discouraged in modern Terraform?

    A. They are not secure.

    B. They are declarative, but slow.

    C. They are not declarative and can make the "state" unreliable.

    D. They only work with AWS.

    Answer: C
    Explanation: Terraform (declarative) is for provisioning. Tools like Ansible (procedural) are for configuration. Mixing them (with provisioners) can lead to a fragile setup.

27. What is `terraform import`?

    A. A command to import a module.

    B. A command to read an *existing, manually-created* resource and bring it into the Terraform state file.

    C. A command to import a `.tfvars` file.

    D. A command to import a provider.

    Answer: B
    Explanation: This is used when you have "brownfield" infrastructure and want to start managing it with Terraform. You must write the `resource` block, then run `import` to link them.

28. What is the `count` meta-argument used for?

    A. To create multiple, similar resources from a single resource block.

    B. To count the number of resources.

    C. To create a loop.

    D. To set a timeout.

Answer: A
Explanation: `count = 3` on an `aws_instance` block will create three identical instances.

29. What is the `for_each` meta-argument used for?

    A. To create multiple resources, but based on a map or a set of strings.

    B. To create a `for` loop.

    C. To count each resource.

    D. To run a command on each resource.

    Answer: A
    Explanation: `for_each` is more flexible than `count`. It creates one resource *for each item* in a map, and uses the map key as the resource identifier (e.g., `aws_instance.web["dev"]`).

30. What is "HCL"?

    A. A security protocol.

    B. The "HashiCorp Configuration Language" - the declarative language used to write `.tf` files.

    C. A CI/CD tool.

    D. A provider for GCP.

    Answer: B
    Explanation: HCL is the human-readable, declarative syntax used by Terraform, Packer, and Vault.

31. What is an "implicit dependency"?

    A. A dependency you must define with `depends_on`.

    B. A dependency that Terraform *automatically* infers when you reference one resource in another.

    C. A dependency on a provider.

    D. A dependency on a module.

    Answer: B
    Explanation: If you set `subnet_id = aws_subnet.private.id`, Terraform knows it must create the subnet *before* it creates the resource that references it. This is implicit.

32. What is an "explicit dependency"?

    A. A dependency that Terraform infers automatically.

    B. A dependency that you must manually define using the `depends_on` meta-argument.

    C. A dependency on a provider.

    D. A dependency on a `.tfvars` file.

Answer: B
Explanation: `depends_on` is used for "hidden" dependencies, where Terraform can't see the link (e.g., an S3 bucket that must be created before an application that *assumes* it exists).

33. What is the `data` block used for?

    A. To create a new resource.

    B. To *read* information about an *existing* resource that is *not* managed by this Terraform configuration.

    C. To define a data bag.

    D. To store the `.tfstate`.

Answer: B
Explanation: A `data` source (e.g., `data "aws_ami" "latest_ubuntu" { ... }`) is a "read-only" query. It's used to fetch data (like the latest Ubuntu AMI ID) to use in your `resource` blocks.

34. What is `terraform workspace`?

    A. A tool for managing multiple, separate state files for the *same* configuration.

    B. A text editor for Terraform.

    C. A directory of modules.

    D. A tool for managing providers.

Answer: A
Explanation: Workspaces allow you to use one set of `.tf` files to manage multiple environments (e.g., `dev`, `staging`, `prod`), each with its own separate state file.

35. Why are Terraform Workspaces (for dev/staging/prod) often discouraged?

    A. They are not secure.

    B. They make it too easy to accidentally run a `destroy` in the wrong environment.

    C. They are too simple.

    D. They don't use HCL.

Answer: B
Explanation: While useful, they can be dangerous. A common best practice is to use separate directories (and separate state files) for each environment.

36. What is the `terraform refresh` command?

    A. It's an alias for `plan`.

    B. It's a command that updates the state file to match the *real-world* infrastructure.

    C. It's a command that re-downloads providers.

    D. It's a command that refreshes the console output.

Answer: B
Explanation: This command checks the status of all resources in the state file against the cloud provider. `terraform plan` automatically runs a refresh first.

37. What is `terraform taint`?

    A. A command that marks a resource as "tainted" (unhealthy), forcing Terraform to destroy and recreate it on the next `apply`.

    B. A command to add a "taint" to a Kubernetes node.

    C. A command to mark a resource for deletion.

    D. A command to validate a resource.

    Answer: A
    Explanation: This is used to force a resource to be recreated, even if its configuration hasn't changed.

38. What is the modern replacement for `terraform taint`?

    A. `terraform apply -taint`

    B. `terraform apply -replace=<resource>`

    C. `terraform taint` is not deprecated.

    D. `terraform force-apply`

    Answer: B
    Explanation: `taint` is now deprecated. The new method is to add the `-replace` flag to your `apply` or `plan` command (e.g., `plan -replace=aws_instance.web`).

39. What is `terraform graph`?

    A. It outputs the resource graph in the "DOT" graphviz format.

    B. It shows a graph of your spending.

    C. It graphs the state file size.

    D. It is not a real command.

    Answer: A
    Explanation: This is used to visualize the "dependency graph" of all your resources, showing how they connect.

    Example HCL for a variable:

```
variable "instance_type" {
  description = "The EC2 instance type"
  type        = string
  default     = "t3.micro"
}
```

40. In the HCL above, what is `type = string`?

    A. A "type constraint" that ensures the variable's value is a string.

B. A "type conversion".

C. A comment.

D. A default value.

Answer: A
Explanation: Type constraints (like `string`, `number`, `bool`, `list(string)`, `map(string)`) are a key feature for validating input.

41. What is the `sensitive = true` argument in a variable or output?

A. It encrypts the value in the state file.

B. It tells Terraform to *redact* (hide) the value from all `plan` and `apply` console output.

C. It marks the variable as optional.

D. It marks the variable as required.

Answer: B
Explanation: This is crucial for secrets. It prevents passwords or API keys from being printed to the console logs.

42. What is "Terraform Cloud"?

A. A cloud provider, like AWS.

B. A set of Terraform modules.

C. A hosted, SaaS platform by HashiCorp that provides remote state, locking, and a CI/CD pipeline for Terraform.

D. The Terraform Registry.

Answer: C
Explanation: Terraform Cloud is an "all-in-one" solution for running Terraform in a team. It provides a remote backend, locking, and a "VCS-driven" (Git) workflow.

43. What is the `terraform login` command?

A. To log into your AWS/Azure account.

B. To log into Terraform Cloud or a private registry.

C. To log into the state file.

D. To log into your Git repository.

Answer: B
Explanation: This command is used to retrieve an API token to authenticate with Terraform Cloud or other remote services.

44. What is the `local-exec` provisioner?

A. A provisioner that runs a command on the *remote* resource.

B. A provisioner that runs a command on the *local machine* (where `terraform apply` is being run).

C. A provisioner that runs a local script.

D. A provisioner that runs at compile time.

Answer: B
Explanation: This is used for tasks that need to happen on the machine running Terraform, *after* a resource is created (e.g., running `kubectl` to configure a new K8s cluster).

45. How can you reference the IP address of an `aws_instance.web` in an `output` block?

A. `output "ip" { value = aws_instance.web.public_ip }`

B. `output "ip" { value = aws_instance.web.ip }`

C. `output "ip" { value = "aws_instance.web.public_ip" }`

D. `output "ip" { value = var.web_ip }`

Answer: A
Explanation: The `value` attribute is set to the expression that references the resource's "exported attribute" (`public_ip`).

46. What is a "splat" expression (e.g., `aws_instance.web.*.id`)?

A. It's not valid syntax.

B. It's a "for-each" loop.

C. It's a syntax for getting a list of attributes from a resource that was created with `count`.

D. It's a syntax for joining strings.

Answer: C
Explanation: If `aws_instance.web` was created with `count = 3`, then `aws_instance.web.*.id` will return a *list* of all three instance IDs.

47. What is a `for` expression (e.g., `[for s in aws_subnet.private : s.id]`)?

A. It's a `for_each` meta-argument.

B. It's a powerful way to iterate over a list or map to transform it into a new list or map.

C. It's a provisioner.

D. It's not valid HCL.

Answer: B
Explanation: This is a "list comprehension." The example takes a list of subnet objects and returns a *new list* containing only their IDs.

48. What is the `required_providers` block?

A. A block that defines which providers are required and their source/version.

B. A block that defines which providers to install.

C. A block that configures the provider (e.g., with region/credentials).

D. A block that is not necessary.

Answer: A
Explanation: This block (inside a `terraform { ... }` block) is required by modern Terraform. It tells `terraform init` *which* providers to download (e.g., `hashicorp/aws`).

49. What is the `provider` block?

A. A block that defines which providers are required.

B. A block that configures a *specific* provider (e.g., `provider "aws" { region = "us-east-1" }`).

C. A block that is the same as the `required_providers` block.

D. A block that defines a module.

Answer: B
Explanation: The `provider` block is where you pass "configuration" to the provider, such as the AWS region, credentials, or other settings.

50. What is the difference between `terraform.tfstate` and `terraform.tfstate.backup`?

A. There is no difference.

B. `.backup` is the state file from the *previous* successful `apply`.

C. `.backup` is the state file from the `plan`.

D. `.backup` is for the `dev` environment.

Answer: B
Explanation: Before Terraform writes a new state file, it saves the *previous* state as `.backup`. This is a safety mechanism in case the `apply` fails.

51. Should you ever commit `terraform.tfstate` to Git?

A. Yes, it's the only way to share it with your team.

B. No, because it contains the state, and may contain sensitive data.

C. Only if the repository is private.

D. Yes, it's a best practice.

Answer: B
Explanation: Never commit the state file. It can contain passwords and other secrets in plain text. Always use a `.gitignore` rule and a remote backend.

52. What does `terraform workspace new dev` do?

A. It creates a new `.tfvars` file.

B. It creates a new workspace named `dev` and switches to it.

C. It creates a new directory named `dev`.

D. It is not a valid command.

Answer: B
Explanation: This command creates a new, separate state file for the `dev` environment.

53. How can you use the *current* workspace name in your code?

    A. `var.workspace`

    B. `terraform.workspace`

    C. `workspace.name`

    D. `$workspace`

    Answer: B
    Explanation: The `terraform.workspace` interpolation can be used to dynamically change things based on the workspace (e.g., `count = terraform.workspace == "prod" ? 5 : 1`).

54. What does the `terraform show` command do?

    A. It shows the current `plan`.

    B. It shows the `README.md` of a module.

    C. It shows the contents of the state file in a human-readable format.

    D. It shows the `.tf` files.

    Answer: C
    Explanation: This command is useful for inspecting the current state and seeing all the resource attributes that Terraform has saved.

55. What is "Terragrunt"?

    A. A Terraform provider.

    B. A thin wrapper for Terraform that provides extra tools for managing multiple modules and environments.

    C. The old name for Terraform.

    D. A security tool for Terraform.

    Answer: B
    Explanation: Terragrunt is a popular third-party tool that helps keep your Terraform code DRY (Don't Repeat Yourself) by managing remote state, locking, and variable configurations.

56. What is the `null_resource`?

    A. A resource that does nothing.

    B. A resource that acts as a placeholder, often used to trigger a provisioner.

    C. A resource that deletes other resources.

    D. A data source.

Answer: B
Explanation: A `null_resource` doesn't create any infrastructure, but it can have provisioners and triggers. It's often used as a "trigger" for a `local-exec` provisioner.

57. What is `terraform output`?

    A. It shows the `plan` output.

    B. It shows the log output.

    C. It prints the values of the `output` blocks from the state file.

    D. It saves the output to a file.

    Answer: C
    Explanation: After an `apply`, you can run `terraform output <name>` to retrieve a specific output value (e.g., `terraform output web_ip`).

58. What is `terraform state list`?

    A. It lists all `.tfstate` files.

    B. It lists all resources that are currently tracked in the state file.

    C. It lists all variables.

    D. It lists all modules.

    Answer: B
    Explanation: This command is useful for seeing the full, addressable names of all resources in your state.

59. What is `terraform state rm`?

    A. It destroys the resource in the cloud.

    B. It removes a resource from the *state file* without destroying it.

    C. It removes the state file.

    D. It is not a valid command.

    Answer: B
    Explanation: This is a dangerous, manual command. It tells Terraform to "stop tracking" a resource, but leaves the real-world resource untouched.

60. What is `terraform state mv`?

    A. It moves a resource to a different cloud.

    B. It "moves" (renames) a resource *within* the state file.

    C. It moves the state file.

    D. It's an alias for `mv terraform.tfstate`.

    Answer: B
    Explanation: This is used when you rename a `resource` block in your code (e.g., from

`aws_instance.web` to `aws_instance.app`). You use `state mv` to tell Terraform the new name.

61. What is `terraform state pull`?

    A. It pulls updates from Git.

    B. It downloads the latest state file from the remote backend and prints it to stdout.

    C. It is an alias for `terraform refresh`.

    D. It pulls in a new resource.

    Answer: B
    Explanation: This command is a manual way to download the current state from your remote backend (like S3) to inspect it locally.

62. What is `terraform state push`?

    A. It pushes changes to Git.

    B. A dangerous command to manually upload a local state file to the remote backend, overwriting what's there.

    C. It pushes a plan to Terraform Cloud.

    D. It is not a valid command.

    Answer: B
    Explanation: This is a "last resort" command, used when you need to manually fix a corrupt remote state by uploading a local, corrected version.

63. How do you define a list of strings as a variable?

    A. `variable "subnets" { type = list(string) }`

    B. `variable "subnets" { type = array[string] }`

    C. `variable "subnets" { type = string[] }`

    D. `variable "subnets" { type = list }`

    Answer: A
    Explanation: `list(string)` is the HCL type for a list of strings. `map(string)` is for a map of strings.

64. How do you access the first element of `var.subnets`?

    A. `var.subnets[0]`

    B. `var.subnets.0`

    C. `var.subnets(0)`

    D. `var.subnets[1]`

    Answer: A
    Explanation: HCL uses zero-based (0) indexing for lists, just like most programming languages.

65. What is the `terraform workspace default`?

    A. The name of the workspace you should use for production.

    B. The name of the "main" workspace that exists before you create any new ones.

    C. A workspace that is not used.

    D. A command to set the default workspace.

    Answer: B
    Explanation: The `default` workspace is the one you are in before you run `workspace new`. Its state file is named `terraform.tfstate`.

66. What does `terraform init -upgrade` do?

    A. It upgrades the Terraform binary.

    B. It upgrades the provider plugins to the latest versions allowed by the version constraints.

    C. It upgrades the state file format.

    D. It upgrades all modules.

    Answer: B
    Explanation: This is used to "un-pin" a provider from the version in the lock file and get the latest version that matches your `required_providers` block.

67. What is the `.terraform.lock.hcl` file?

    A. The state lock file.

    B. A file that locks your modules.

    C. A dependency lock file that records the exact versions of the providers used.

    D. A file that should be in `.gitignore`.

    Answer: C
    Explanation: This file (like `Gemfile.lock` or `package-lock.json`) ensures that every `init` will install the *exact same* provider versions, making builds reproducible. It *should* be committed to Git.

68. What does the `dynamic` block (inside a `resource`) do?

    A. It makes the resource dynamic.

    B. It creates a `for_each` loop.

    C. It dynamically generates nested blocks (like `ingress`) for a resource, based on a list or map.

    D. It runs a dynamic provisioner.

    Answer: C
    Explanation: This is an advanced feature. If you need to create a variable number of `ingress` rules for a security group, you use a `dynamic "ingress"` block.

69. What is the `lifecycle` block?

    A. A block that defines a resource's state.

    B. A meta-argument block that customizes a resource's lifecycle (e.g., `prevent_destroy`, `create_before_destroy`).

    C. A block that defines how long a resource should live.

    D. A block that controls the `plan` lifecycle.

    Answer: B
    Explanation: The `lifecycle` block is used to change Terraform's default behavior for a specific resource.

70. What does `lifecycle { prevent_destroy = true }` do?

    A. It causes Terraform to fail the `plan` or `apply` if it tries to destroy this resource.

    B. It prevents the resource from being created.

    C. It prevents the resource from being updated.

    D. It makes the resource immortal.

    Answer: A
    Explanation: This is a critical safety lock for "singleton" resources, like a production database, to prevent accidental deletion.

71. What does `lifecycle { create_before_destroy = true }` do?

    A. It creates a new resource, then deletes it.

    B. It forces a replacement, but ensures the new resource is created and healthy *before* the old one is destroyed.

    C. It creates a resource but prevents it from being destroyed.

    D. It is not valid syntax.

    Answer: B
    Explanation: This ensures zero-downtime for "rolling updates" of resources that must be replaced (e.g., an EC2 Launch Configuration).

72. How do you pass a variable's value on the command line?

    A. `terraform apply -var "region=us-east-1"`

    B. `terraform apply -var="region=us-east-1"`

    C. `terraform apply -var 'region=us-east-1'`

    D. All of the above.

    Answer: D
    Explanation: The `-var` flag (and `-var-file`) is used to pass input variables on the command line, which overrides `.tfvars` files.

73. What is the "Terraform Registry"?

A. The `docker.io` for Terraform.

B. A public, centralized repository (run by HashiCorp) for finding and sharing modules and providers.

C. The `.terraform.lock.hcl` file.

D. The `.tfstate` file.

Answer: B
Explanation: The Registry (registry.terraform.io) is the main source for all public providers and modules.

74. What is the `templatefile` function?

A. A function that renders a template file, substituting variables into it.

B. A function that defines a resource.

C. A function that creates a `.tfvars` file.

D. A deprecated function.

Answer: A
Explanation: This function (`templatefile("config.tpl", { port = 8080 })`) is used to render simple config files.

75. What is a `data "terraform_remote_state" "vpc"` block?

A. A block to move state.

B. A data source that allows one Terraform configuration to read the `output` values from *another* configuration's state file.

C. A block to configure the remote state.

D. A block to pull the state locally.

Answer: B
Explanation: This is the primary way to share data between separate Terraform configurations (e.g., your "application" configuration can read the `vpc_id` from your "VPC" configuration).

76. What is the `tflint` tool?

A. The same as `terraform fmt`.

B. The same as `terraform validate`.

C. A third-party "linter" that checks for syntax, best practices, and potential errors.

D. A tool to clean `.tfstate` files.

Answer: C
Explanation: `tflint` is a popular, more advanced linter than the built-in `validate`. It can find provider-specific issues (e.g., "don't use a `t2.micro` for production").

77. What is the `checkov` tool?

A. A Terraform provider.

B. A static analysis *security* scanner for Infrastructure as Code.

C. A tool for checking `.tfvars` files.

D. A tool for checking out code.

Answer: B
Explanation: `checkov` (and `tfsec`) is a DevSecOps tool. You run it in your CI pipeline to scan your `.tf` files for security misconfigurations (e.g., "S3 bucket is public").

78. What does `terraform output -json` do?

A. It prints the state file as JSON.

B. It prints all output values in a machine-readable JSON format.

C. It validates a JSON file.

D. It converts HCL to JSON.

Answer: B
Explanation: This is very useful for scripting, as it allows you to pipe the output of one Terraform configuration to another tool.

79. What is the "interpolation syntax" in Terraform?

A. The `provider` block.

B. The syntax used to reference variables and resource attributes (e.g., `"${var.name}"`).

C. The `depends_on` argument.

D. The `output` block.

Answer: B
Explanation: Interpolation is the `${...}` syntax used inside strings to embed the value of an expression.

80. What is the `main.tf` file?

A. A required file that must contain all resources.

B. A conventional file, but not required. It's just a common name for the main configuration.

C. The file that defines the `main` module.

D. The state file.

Answer: B
Explanation: Terraform combines all `*.tf` files in a directory. Naming your file `main.tf` (and `variables.tf`, `outputs.tf`) is just a convention, not a requirement.

81. What is the `count.index` variable?

A. The total number of resources.

B. A special variable, available *inside* a resource block that uses `count`, which gives the 0-based index of the current resource.

C. The index of a module.

D. The index of a provider.

Answer: B
Explanation: If you have `count = 3`, this allows you to name your instances `web-0`, `web-1`, and `web-2` (e.g., `name = "web-${count.index}"`).

82. What are the `each.key` and `each.value` variables?

A. Variables for `count`.

B. Special variables, available *inside* a resource block that uses `for_each`, which give the key and value of the current map item.

C. Variables for modules.

D. Variables for data sources.

Answer: B
Explanation: If you have `for_each = { dev = "t3.micro", prod = "m5.large" }`, `each.key` would be "dev" (then "prod") and `each.value` would be "t3.micro" (then "m5.large").

83. What is a "state-locking" failure?

A. When Terraform fails to get a "lock" on the remote state file, usually because someone else is running `apply`.

B. When the state file is corrupt.

C. When the state file is in Git.

D. When the state file is read-only.

Answer: A
Explanation: This is not an error, but a safety feature. It tells you to "wait until the other operation is complete."

84. What is a "state-drift"?

A. When the `.tfstate` file is deleted.

B. When the *real-world* infrastructure (in AWS, etc.) no longer matches the `.tfstate` file, because someone made a manual change.

C. When the `.tf` files do not match the `.tfstate` file.

D. When a `plan` fails.

Answer: B
Explanation: State drift is a major problem. If someone manually deletes an S3 bucket, the `.tfstate` file still *thinks* it exists. A `terraform plan` will detect this drift.

85. What does `terraform workspace select default` do?

A. It creates a new workspace.

B. It selects the `dev` workspace.

C. It switches you back to the main, `default` workspace.

D. It deletes all other workspaces.

Answer: C
Explanation: This command changes your active workspace back to the one named `default`.

86. What is a "provider alias"?

A. A way to use the *same* provider (e.g., "aws") multiple times with different configurations (e.g., one for `us-east-1` and one for `us-west-2`).

B. A shorter name for a provider.

C. A provider that is deprecated.

D. A provider for DNS.

Answer: A
Explanation: You define `provider "aws" { alias = "west" region = "us-west-2" }`. Then, you tell a resource to use it: `resource "aws_instance" "w" { provider = aws.west }`.

87. What is the "Terraform Graph"?

A. The dependency graph (DAG) that Terraform builds to understand the relationships between resources.

B. A dashboard of resource usage.

C. A visual tool.

D. A block in HCL.

Answer: A
Explanation: Terraform builds this graph (using implicit and explicit dependencies) to determine the *order* of operations. This is how it knows to create a VPC before a subnet.

88. What is the `file()` function?

A. A resource for creating files.

B. A function that reads the contents of a file on the *local disk* (where `plan` is run) and returns it as a string.

C. A function that reads a file from a remote URL.

D. A function that creates a file.

Answer: B
Explanation: This is used to "slurp" a file's content into your configuration, e.g., `policy = file("my_policy.json")`.

89. What is the `join()` function?

    A. A function that joins two modules.

    B. A function that takes a separator string and a list, and joins the list items into a single string.

    C. A function that joins two resource blocks.

    D. A function that joins two strings.

    Answer: B
    Explanation: Example: `join(", ", ["a", "b", "c"])` would return the string `"a, b, c"`.

90. What is the `lookup()` function?

    A. A function to look up a resource.

    B. A function to look up a value in a *map*, with a fallback default value.

    C. A function to look up a provider.

    D. A function to look up a module.

    Answer: B
    Explanation: `lookup(var.my_map, "key", "default_value")` will return `var.my_map["key"]`, but if "key" doesn't exist, it will return "default_value" instead of failing.

91. What is `terraform login` used for?

    A. To log into your cloud provider (e.g., AWS).

    B. To retrieve an API token for Terraform Cloud or a private registry.

    C. To log into a remote server via SSH.

    D. To unlock a state file.

    Answer: B
    Explanation: This command authenticates your CLI with a remote service like Terraform Cloud, allowing it to pull modules or use the remote backend.

92. What is the `required_version` setting?

    A. It sets the version of the provider.

    B. It sets the version of the module.

    C. It sets a constraint on the version of the *Terraform CLI* that can be used with this code.

    D. It sets the version of your application.

    Answer: C
    Explanation: This is set in the `terraform { ... }` block (e.g., `required_version = ">= 1.0.0"`). It prevents someone from running your code with an old, incompatible Terraform binary.

93. What is `terraform graph | dot -Tpng > graph.png`?

    A. A command that fails.

    B. A command that pipes the DOT-formatted graph output to the `dot` (Graphviz) tool to render it as a PNG image.

    C. A command that creates a file named `dot`.

    D. A command that outputs a text-based graph.

    Answer: B
    Explanation: This is the standard way to visualize the dependency graph that `terraform graph` produces.

94. What is "Sentinel"?

    A. A Terraform provider.

    B. A monitoring tool.

    C. HashiCorp's "Policy as Code" framework, often used with Terraform Cloud.

    D. A state file encryption tool.

    Answer: C
    Explanation: Sentinel is a policy language. In Terraform Cloud, you can write Sentinel policies like "No security group can have an inbound rule for 0.0.0.0/0 on port 22."

95. What is the `file` provisioner?

    A. A provisioner that runs a script.

    B. A provisioner that copies a file from the local machine to the newly created remote resource.

    C. A provisioner that creates a file.

    D. A provisioner that reads a file.

    Answer: B
    Explanation: This is used to upload files (e.g., a config file) to the new resource. Like `remote-exec`, it's often better to use a dedicated configuration management tool.

96. How do you define a complex variable type, like an object?

    A. `variable "user" { type = object({ name = string, age = number }) }`

    B. `variable "user" { type = map }`

    C. `variable "user" { type = hash }`

    D. `variable "user" { type = json }`

    Answer: A
    Explanation: The `object(...)` type constructor allows you to define a specific, complex shape for your variable, ensuring the input is structured correctly.

97. What is the `terraform force-unlock` command?

A. A command to force `apply`.

B. A command to manually remove a "lock" on the state file if it was left by a failed or crashed process.

C. A command to unlock a module.

D. A command to unlock a provider.

Answer: B
Explanation: This is a dangerous command. If an `apply` fails and the lock is *not* released, you can use this command to manually break the lock.

98. What is the `depends_on` argument used for?

A. To define an "explicit" dependency that Terraform cannot see.

B. To define an "implicit" dependency.

C. To define a module dependency.

D. To define a provider dependency.

Answer: A
Explanation: You should always prefer implicit dependencies. You only use `depends_on` when one resource depends on another, but there is no attribute reference linking them.

99. What is the `zipmap` function?

A. It zips a file.

B. It takes a list of keys and a list of values and "zips" them together into a map.

C. It unzips a map.

D. It maps a zip file.

Answer: B
Explanation: `zipmap(["a", "b"], [1, 2])` would return the map `{ "a" = 1, "b" = 2 }`.

100. What does "declarative" mean in the context of Terraform?

A. You describe the *steps* to take.

B. You describe the *end result* you want, and Terraform figures out the steps.

C. The code is written in YAML.

D. The code is run in a CI/CD pipeline.

Answer: B
Explanation: This is the core concept. You don't write "if server exists, update it, else create it." You just write the *definition* of the server, and Terraform's reconciliation loop does the rest.