1. **A user runs terraform init on their RHEL-based server, and per the output, two provider plugins are downloaded:**

**$ terraform init**
**Initializing the backend...**
**Initializing provider plugins...**
**- Checking for available provider plugins...**
**- Downloading plugin for provider "aws" (hashicorp/aws) 2.44.0...**
**- Downloading plugin for provider "random" (hashicorp/random) 2.2.1...**
**Terraform has been successfully initialized!**

**Where are these plugins downloaded and stored on the server?:** The .terraform/providers directory in the current working directory
2. **When multiple arguments with single-line values appear on consecutive lines at the same nesting level, HashiCorp recommends that you::** align the equals signs
3. **When you add a new module to a configuration, Terraform must download it before it can be used. What two commands can be used to download and update modules? (select two):** terraform init: This command downloads and updates the required modules for the Terraform configuration. It also sets up the backend for state storage if specified in the configuration.

terraform get: This command is used to download and update modules for a Terraform configuration. It can be used to update specific modules by specifying the module name and version number, or it can be used to update all modules by simply running the command without any arguments.
4. **From the code below, identify the implicit dependency:**

**resource "aws_eip" "public_ip" {**
**vpc = true**
**instance = aws_instance.web_server.id**
**}**
**resource "aws_instance" "web_server" {**
**ami = "ami-2757f631"**
**instance_type = "t2.micro"**
**depends_on = [aws_s3_bucket.company_data]**
**}:** The implicit dependency in the code is the EC2 instance labeled "web_server" because the aws_eip resource depends on the aws_instance.web_server.id for its instance attribute.

5. **Terraform Cloud is more powerful when you integrate it with your version control system (VCS) provider. Select all the supported VCS providers from the answers below. (select four):** GitHub Enterprise, Azure DevOps Server, GitHub.com, Bitbucket Cloud
6. **Which of the following statements represents the most accurate statement about the Terraform language?:** Terraform is a mutable, imperative, Infrastructure as Code provisioning language based on Hashicorp Configuration Language, or optionally YAML.
7. **In the example below, the depends_on argument creates what type of dependency?**

**resource "aws_instance" "example" {**
**ami = "ami-2757f631"**
**instance_type = "t2.micro"**
**depends_on = [aws_s3_bucket.company_data]**
**}:** The `depends_on` argument in Terraform creates an explicit dependency between resources. This means that Terraform will wait for the specified resource to be created or updated before proceeding with the dependent resource.
8. **You are developing a new Terraform module to demonstrate features of the most popular HashiCorp products. You need to spin up an AWS instance for each tool, so you create the resource block as shown below using the for_each meta-argument.**

**resource "aws_instance" "bryan-demo" {**
**# ...**
**for_each = {**
**"terraform": "infrastructure",**
**"vault": "security",**
**"consul": "connectivity",**
**"nomad": "scheduler",**
**}**
**}**
**After the deployment, you view the state using the terraform state list command. What resource address would be displayed for the instance related to vault?:** The correct resource address format for the instance related to "vault" when using the for_each meta-argument in Terraform is aws_instance.bryan-demo["vault"]. This format allows Terraform to uniquely identify and manage each instance based on the key-value pair provided in the for_each block.

9. **A user has created three workspaces using the command line - prod, dev, and test. The user wants to create a fourth workspace named stage.**
**Which command will the user execute to accomplish this task?:** The correct command to create a new workspace in Terraform is "terraform workspace new ". This command will create a new workspace named "stage" as requested by the user.
10. **What do the declarations, such as name, cidr, and azs, in the following Terraform code represent and what purpose do they serve?**

**module "vpc" {**
**source = "terraform-aws-modules/vpc/aws"**
**version = "5.7.0"**
**name = var.vpc_name**
**cidr = var.vpc_cidr**
**azs = var.vpc_azs**
**private_subnets = var.vpc_private_subnets**
**public_subnets = var.vpc_public_subnets**
**enable_nat_gateway = var.vpc_enable_nat_gateway**
**tags = var.vpc_tags**
**}:** these are variables that are passed into the child module likely used for resource creation
Explanation
The declarations like name, cidr, and azs are variables that are being passed into the child module for resource creation. These variables allow for customization and flexibility in configuring the VPC module according to specific requirements.
11. **Emma is a Terraform expert, and she has automated all the things with Terraform. A virtual machine was provisioned during a recent deployment, but a local script did not work correctly. As a result, the virtual machine needs to be destroyed and recreated.**
**How can Emma quickly have Terraform recreate the one resource without having to destroy everything that was created?:** Using `terraform apply -replace=aws_instance.web` allows Emma to mark the specific virtual machine resource for replacement without affecting other resources that were created. This command is useful for quickly recreating a single resource.
12. **In Terraform, most resource dependencies are handled automatically. Which of the following statements best describes how Terraform resource dependencies are handled?:** Terraform automatically analyzes expressions within a resource block to identify dependencies on other resources. This allows Terraform to determine the correct order of operations when creating, updating, or destroying

resources.

Terraform analyzes any expressions within a resource block to find references to other objects and treats those references as implicit ordering requirements when creating, updating, or destroying resources.
13. **Oscar is modifying his Terraform configuration file but isn't 100% sure it's correct. He is afraid that changes made could negatively affect production workloads.**
**How can Oscar validate the changes that will be made without impacting existing workloads?:** Running a terraform plan allows Oscar to preview the changes that will be made to the infrastructure without actually applying them. This way, he can validate the changes and ensure they won't negatively impact existing workloads before making any modifications.
14. **True or False? The following code is an example of an implicit dependency in Terraform**

```
resource "aws_instance" "web" {
ami = "ami-0c55b159cbfafe1f0"
instance_type = "t2.micro"
}
resource "aws_ebs_volume" "data" {
availability_zone = "us-west-2a"
size = 1
tags = {
Name = "data-volume"
}
}
resource "aws_volume_attachment" "attach_data_volume" {
device_name = "/dev/xvdf"
volume_id = aws_ebs_volume.data.id
instance_id = aws_instance.web.id
```

**}:** The code snippet provided shows an implicit dependency in Terraform.
The resource "aws_volume_attachment" "attach_data_volume" depends on both "aws_ebs_volume.data" and "aws_instance.web" resources without explicitly specifying the dependency using the "depends_on" attribute. Terraform automatically detects this relationship and ensures that the dependencies are resolved in the correct order during the execution.
15. **When using constraint expressions to signify a version of a provider, which of the following are valid provider versions that satisfy the expression found**

**in the following code snippet: (select two)**

```
terraform {
required_providers {
aws = {
source = "hashicorp/aws"
version ~> "5.36.0"
}
}
```

**:** The version "5.36.9" satisfies the constraint expression "~> 5.36.0" as it falls within the same minor version range (5.36.x).

The version "5.36.3" satisfies the constraint expression "~> 5.36.0" as it falls within the same minor version range (5.36.x).

16. **You are adding a new variable to your configuration. Which of the following is NOT a valid variable type in Terraform?:** In Terraform, the variable type `float` is not a valid type. Terraform supports variable types such as `string`, `map`, `bool`, and `number`, but not `float`.

17. **What Terraform command can be used to remove the lock on the state for the current configuration?:** The correct Terraform command to remove the lock on the state for the current configuration is `terraform force-unlock`. This command is specifically designed to force unlock the state file and allow modifications to be made.

18. **Terraform is distributed as a single binary and available for many different platforms. Select all Operating Systems that Terraform is available for. (select five):** Solaris, Windows, FreeBSD, macOS, Linux

19. **Environment variables can be used to set the value of input variables. The environment variables must be in the format "____"_<variablename>.**
**Select the correct prefix string from the following list.:** The correct prefix string for setting input variables using environment variables in Terraform is TF_VAR. This prefix is recognized by Terraform to assign values to variables.

20. **Sara has her entire application automated using Terraform, but she needs to start automating more infrastructure components, such as creating a new subnet, DNS record, and load balancer. Sara wants to create these new resources using modules so she easily reuse the code. However, Sara is having problems getting the subnet_id from the subnet module to pass to the load balancer module.**
**modules/subnet.tf:**

```
resource "aws_subnet" "bryan" {
```

**vpc_id = aws_vpc.krausen.id**
**cidr_block = "10.0.1.0/24"**
**tags = {**
**Name = "Krausen Subnet"**
**}**
**}:** add an output block to the subnet module and retrieve the value using module.subnet.subnet_id for the load balancer module
Explanation
Adding an output block to the subnet module allows the subnet_id to be exposed as an output variable. This output variable can then be retrieved using module.subnet.subnet_id in the load balancer module, enabling Sara to pass the subnet_id between modules.
21. **Understanding how indexes work is essential when working with different variable types and resource blocks that use count or for_each. Therefore, what is the output value of the following code snippet?**

**variable "candy_list" {**
**type = list(string)**
**default = ["snickers", "kitkat", "reeces", "m&ms"]**
**}**
**output "give_me_candy" {**
**value = element(var.candy_list, 2)**
**}:** Explanation
The output value of the code snippet is "reeces" because the element function is used to access the element at index 2 in the candy_list variable, which is "reeces".
22. **True or False? Rather than use a state file, Terraform can inspect cloud resources on every run to validate that the real-world resources match the desired state.:** This choice is correct. Terraform requires a state file to store information about the current state of infrastructure resources. By inspecting this state file, Terraform can determine the necessary changes to bring the real-world resources in line with the desired state specified in the configuration files. Without a state file, Terraform would not be able to perform this validation.
23. **What Terraform command will launch an interactive console to evaluate and experiment with expressions?:** The correct Terraform command to launch the Interactive console is "terraform console". This command allows users to evaluate and experiment with expressions in an interactive manner.
24. **In the following code snippet, the type of Terraform block is identified by which string?**

**resource "aws_instance" "db" {**
**ami = "ami-123456"**
**instance_type = "t2.micro"**
**}:** The type of Terraform block is identified by the keyword "resource" in this code snippet. This keyword indicates that a new AWS instance resource is being defined.
25. **True or False? Each Terraform workspace uses its own state file to manage the infrastructure associated with that particular workspace.:** True
26. **Anyone can publish and share modules on the Terraform Public Registry, and meeting the requirements for publishing a module is extremely easy. What are some of the requirements that must be met in order to publish a module on the Terraform Public Registry? (select three):** The requirement for module repositories to follow the terraform-- naming format is accurate, as this naming convention helps organize and categorize modules on the Terraform Public Registry.

The requirement for release tag names to follow the x.y.z format and optionally be prefixed with a 'v' is valid, as it ensures consistency and clarity in versioning for modules on the Terraform Public Registry.

The requirement for the module to be on GitHub and a public repository is correct, as the Terraform Public Registry relies on GitHub for module hosting and version control.
27. **What tasks can the terraform state command be used for in Terraform?: -** The `terraform state` command can indeed be used to modify the current state by removing items. This is useful for managing the state of resources in Terraform.
28. **Why might a user opt to include the following snippet in their configuration file?**

**terraform {**
**required_version = ">= 1.7.5"**
**}:** The snippet specifies the minimum version of Terraform required to run the configuration, ensuring compatibility and preventing potential issues that may arise from using older versions.
29. **When writing Terraform code, how many spaces between each nesting level does HashiCorp recommend that you use?:** HashiCorp recommends using 2 spaces between each nesting level in Terraform code for better readability and maintainability.
30. **After many years of using Terraform Community (Free), you decide to migrate to Terraform Cloud. After the initial configuration, you create a work-**

**space and migrate your existing state and configuration. What Terraform version would the new workspace be configured to use after the migration?:** the same Terraform version that was used to perform the migration

31. **A user creates three workspaces from the command line: prod, dev, and test. Which of the following commands will the user run to switch to the dev workspace?:** The correct command to switch workspaces in Terraform is "terraform workspace select ". Therefore, terraform workspace select dev is the correct command to switch to the "dev" workspace.

32. **Henry has been working on automating his Azure infrastructure for a new application using Terraform. His application runs successfully, but he has added a new resource to create a DNS record using the new Infoblox provider. He has added the new resource but gets an error when he runs a terraform plan.**
**What should Henry do first before running a plan and apply?:** Running `terraform init` is necessary when a new provider is introduced to download the required plugin. This ensures that Terraform has access to the Infoblox provider and can properly manage the DNS record resource.

33. **You are writing Terraform to deploy resources, and have included provider blocks as shown below:**

```
provider "aws" {
region = "us-east-1"
}
provider "aws" {
region = "us-west-1"
}
```

**When you validate the Terraform configuration, you get the following error:**

```
Error: Duplicate provider configuration
on main.tf line 5:
5: provider "aws" {
A default provider configuration for "aws" was already given at
main.tf:1,1-15. If multiple configurations are required, set the xxxx
argument for alternative configurations.
```

**What additional parameter is required to use multiple provider blocks of the same type, but with distinct configurations, such as cloud regions, namespaces, or other desired settings?:** Explanation

The correct additional parameter required to use multiple provider blocks of the same type with distinct configurations is the "alias" parameter. This allows you to differentiate between the different configurations of the same provider type.
34. **Where does Terraform Community (Free) store the local state for workspaces?:** Terraform Community (Free) stores the local state for workspaces in a directory called `terraform.tfstate.d/`. This directory structure allows for separate state files for each workspace, making it easier to manage and maintain the state data.
35. **True or False? The terraform plan -refresh-only command is used to create a plan whose goal is only to update the Terraform state to match any changes made to remote objects outside of Terraform.:** The statement is true because the terraform plan -refresh-only command is specifically designed to only refresh the Terraform state to match any changes made to remote objects outside of Terraform. It does not apply those changes to the state.
36. **Provider dependencies are created in several different ways. Select the valid provider dependencies from the following list: (select three):** When any resource block or data block is used in the configuration, it indicates a dependency on the provider associated with those blocks, as they are responsible for managing those specific resources.

The existence of any resource instance belonging to a particular provider in the current state signifies a dependency on that provider, as Terraform needs access to the provider to manage the state of those resources.

The explicit use of a provider block in the configuration, along with an optional version constraint, establishes a direct dependency on that specific provider for the resources being managed.
37. **What is the correct syntax for defining a list of strings for a variable in Terraform?**
**variable "public_subnets" {**
**description = "The number of public subnets for VPC"**
**type = list(string)**
**default = 2**
**}:** variable "public_subnet_cidr_blocks" {
type = list(string)
default = [
"10.0.1.0/24",
"10.0.1.0/24",
"10.0.1.0/24",

"10.0.1.0/24",
]
}

38. **You and a colleague are working on updating some Terraform configurations within your organization. You need to follow a new naming standard for the local name within your resource blocks. However, you don't want Terraform to replace the object after changing your configuration files.**
**As an example, you want to change data-bucket to now be prod-encrypted-data-s3-bucket in the following resource block:**

**resource "aws_s3_bucket" "data-bucket" {**
**bucket = "corp-production-data-bucket"**
**tags = {**
**Name = "corp-production-data-bucket"**
**Environment = "prod"**
**}**
**}**

**After updating the resource block, what command would you run to update the local name while ensuring Terraform does not replace the existing resource?:** terraform state mv aws_s3_bucket.data-bucket aws_s3_bucket.prod-encrypted-data-s3-bucket

The correct command to update the local name without replacing the existing resource is to use the `terraform state mv` command. This command will move the existing state object to the new local name specified, ensuring that Terraform does not replace the resource.

39. **In order to reduce the time it takes to provision resources, Terraform uses parallelism. By default, how many resources will Terraform provision concurrently during a terraform apply?:** Terraform by default provisions 10 resources concurrently during a `terraform apply` command to speed up the provisioning process and reduce the overall time taken.

40. **True or False? By default, the terraform destroy command will prompt the user for confirmation before proceeding.:** By default, the terraform destroy command does prompt the user for confirmation before proceeding to ensure that the user is aware of the resources that will be deleted. This is a safety measure to prevent accidental deletion of important infrastructure.

True. By default, Terraform will prompt for confirmation before proceeding with the

terraform destroy command. This prompt allows you to verify that you really want to destroy the infrastructure that Terraform is managing before it actually does so. Terraform destroy will always prompt for confirmation before executing unless passed the -auto-approve flag.

$ terraform destroy
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value: yes
41. **Which are some of the benefits of using Infrastructure as Code in an organization? (select three):** IaC code is platform-agnostic and can be used to manage infrastructure across various cloud platforms, providing flexibility and scalability in managing resources.

IaC utilizes a human-readable configuration language, making it easier for developers and operators to understand, write, and maintain infrastructure code efficiently.

Using Infrastructure as Code (IaC) allows for configurations to be stored in version control, enabling collaboration, tracking changes, and ensuring consistency in infrastructure deployment.
42. **Which Terraform command will check and report errors within modules, attribute names, and value types to ensure they are syntactically valid and internally consistent?:** The correct choice is terraform validate because this command is specifically designed to check and report errors within modules, attribute names, and value types to ensure they are syntactically valid and internally consistent.
43. **Why might users want to utilize Sentinel or OPA with Terraform Cloud in their infrastructure workflow? (select four):** Organizations can use Sentinel and OPA to enforce resource naming conventions and approved machine images, which helps in maintaining consistency and clarity across the infrastructure. This can prevent naming conflicts and ensure that only approved resources are used.

Sentinel and OPA can enhance security by allowing users to define and enforce policies that prevent unauthorized changes to infrastructure. This helps in maintaining the integrity and security of the managed infrastructure.

Sentinel and OPA can provide real-time feedback on potential security risks present in Terraform configurations during the development process. This allows developers to identify and address security issues early on, reducing the risk of vulnerabilities

in the infrastructure.

Sentinel and OPA enable automated policy checks that can be used to enforce compliance standards before any changes are applied to production environments. This ensures that all changes meet the required standards and regulations.
44. **What environment variable can be set to enable detailed logging for Terraform?:** The correct environment variable to enable detailed logging for Terraform is `TF_LOG`. Setting this variable will provide detailed logs for troubleshooting and debugging purposes.
45. **Harry has deployed resources on Azure using Terraform. However, he has discovered that his co-workers Ron and Ginny have manually created a few resources using the Azure console. Since it is company policy to manage production workloads using IaC, how can Harry bring these resources under Terraform management without negatively impacting the availability of the deployed resources?:** Using `terraform import` or the `import` block allows Harry to bring the existing resources under Terraform management without disrupting the availability of the deployed resources. This method ensures that the resources are managed by Terraform while preserving their current state.
46. **A "backend" in Terraform determines how state is loaded and how an operation such as apply is executed. Which of the following is not a supported backend type?:** The "github" backend type is not a supported backend type in Terraform. Terraform does not have built-in support for storing state in a GitHub repository.
47. **You are performing a code review of a colleague's Terraform code and see the following code. Where is this module stored?**

**module "vault-aws-tgw" {**
**source = "terraform-vault-aws-tgw/hcp"**
**version = "1.0.0"**
**client_id = "4djlsn29sdnjk2btk"**
**hvn_id = "a4c9357ead4de"**
**route_table_id = "rtb-a221958bc5892eade331"**
**}:** the Terraform public registry
Explanation
The code specifies a source from "terraform-vault-aws-tgw/hcp", which is a typical format for modules stored in the Terraform public registry. This choice is correct based on the information provided in the code snippet.
48. **What feature of Terraform Cloud allows you to publish and maintain a set of custom modules that can only be used within your organization?:** private

registry
Explanation
The private registry feature in Terraform Cloud allows users to publish and maintain custom modules within their organization, providing a secure and controlled environment for sharing infrastructure configurations.
49. **Which code snippet would allow you to retrieve information about existing resources and use that information within your Terraform configuration?:** data "aws_ami" "aws_instance" {
most_recent = true
owners = ["self"]
tags = {
Name = "app-server"
Tested = "true"
}
}
Explanation
This code snippet defines a data block for retrieving information about an AWS AMI (Amazon Machine Image) based on specific criteria like owners and tags. This data can then be used within the Terraform configuration to make decisions or set attributes based on the retrieved information.
50. **You are using a Terraform Cloud workspace linked to a GitHub repo to manage production workloads in your environment. After approving a merge request, what default action can you expect to be triggered on the workspace?:** A speculative plan will be run to show the potential changes to the managed environment and validate the changes against any applicable Sentinel policies
Explanation
After approving a merge request, Terraform Cloud will run a speculative plan to show the potential changes that will be applied to the managed environment. This allows users to review and validate the changes against any applicable Sentinel policies before applying them.
51. **You are using Terraform to deploy some cloud resources and have developed the following code. However, you receive an error when trying to provision the resource. Which of the following answers fixes the syntax of the Terraform code?**
**resource "aws_security_group" "vault_elb" {**
**name = "${var.name_prefix}-vault-elb"**
**description = Vault ELB**
**vpc_id = var.vpc_id**
**}:** resource "aws_security_group" "vault_elb" {

name = "${var.name_prefix}-vault-elb"
description = "Vault ELB"
vpc_id = var.vpc_id
}
Explanation
The syntax error in the original code is that the description value is missing quotation marks. By adding the quotes around "Vault ELB", the code will be corrected and the provisioning process will not throw an error.
52. **Which of the following variable declarations is going to result in an error?-:** variable "example" {
description = "This is a variable description"
type = list(string)
default = {}
}
Explanation
The declaration is incorrect because the default value is set to an empty object, which is not a valid value for a list of strings. The type should match the default value.
53. **Freddy and his co-worker Jason are deploying resources in GCP using Terraform for their team. After resources have been deployed, they must destroy the cloud-based resources to save on costs. However, two other team members, Michael and Chucky, are using a Cloud SQL instance for testing and request to keep it running.**
**How can Freddy and Jason destroy all other resources without negatively impacting the database?:** run a terraform state rm command to remove the Cloud SQL instance from Terraform management before running the terraform destroy command
Explanation
Removing the Cloud SQL instance from Terraform management using the `terraform state rm` command ensures that the instance is not included in the resources to be destroyed when running `terraform destroy`. This allows Freddy and Jason to delete all other resources without impacting the database.
54. **Which of the following is a valid variable name in Terraform?:** invalid
Explanation
This is a valid variable name in Terraform as it follows the naming conventions for variables, which allow alphanumeric characters and underscores, and must start with a letter or underscore.
55. **True or False? Using the latest versions of Terraform, terraform init cannot automatically download community providers.:** False
Explanation

The statement "False" is correct because using the latest versions of Terraform, the command `terraform init` can automatically download community providers. This functionality simplifies the process of integrating community providers into Terraform configurations, enhancing the overall user experience.
56. **True or False? When using the Terraform provider for Vault, the tight integration between these HashiCorp tools provides the ability to mask secrets in the state file.:** False
Explanation
Correct. The statement is false because the tight integration between Terraform and Vault does not automatically mask secrets in the state file. Developers need to implement secure practices to handle secrets effectively.
57. **While Terraform is generally written using the HashiCorp Configuration Language (HCL). What other syntax can Terraform be expressed in?:** JSON
Explanation
Terraform can be expressed in JSON syntax in addition to HCL. JSON is a popular choice for configuration files due to its simplicity and compatibility with various systems.
58. **What is Terraform?:** An open source provisioning declarative tool that based on Infrastructure as a Code paradigm
designed on immutable infrastructure principles
Written in Golang and uses own syntax - HCL (Hashicorp Configuration Language), but also supports JSON
Helps to evolve the infrastructure, safely and predictably
Applies Graph Theory to IaaC and provides Automation, Versioning and Reusability
Terraform is a multipurpose composition tool:Ë Composes multiple tiers (SaaS/PaaS/IaaS)Ë A plugin-based architecture model
Terraform is not a cloud agnostic tool. It embraces all major Cloud Providers and provides common language to orchestrate the infrastructure resources
Terraform is not a configuration management tool and other tools like chef, ansible exists in the market.
59. **Terraform Architecture - Terraform Providers (Plugins):** provide abstraction above the upstream API and is responsible for understanding API interactions and exposing resources.
Invoke only upstream APIs for the basic CRUD operations
Providers are unaware of anything related to configuration loading, graphtheory, etc.
supports multiple provider instances using alias for e.g. multiple aws provides with different region
can be integrated with any API using providers framework
Most providers configure a specific infrastructure platform (either cloud or self-host-

ed).
can also offer local utilities for tasks like generating random numbers for unique resource names
60. **Terraform Provisioners:** run code locally or remotely on resource creationlocal exec executes code on the machine running terraformremote execruns on the provisioned resourcesupports ssh and winrmrequires inline list of commands should be used as a last resort
are defined within the resource block.
support types - Create and Destroyif creation time fails, resource is tainted if provisioning failed, by default. (next apply it will be re-created)behavior can be overridden by setting the on_failure to continue, which means ignore and continuefor destroy, if it fails - resources are not removed
61. **Terraform Workspaces:** helps manage multiple distinct sets of infrastructure resources or environments with the same code.
just need to create needed workspace and use them, instead of creating a directory for each environment to manage
state files for each workspace are stored in the directory terraform.tfstate.d
terraform workspace new dev creates a new workspace and switches to it as well
terraform workspace select dev helps select workspace
terraform workspace list lists the workspaces and shows the current active one with *
does not provide strong separation as it uses the same backend
62. **Terraform Workflow - init:** initializes a working directory containing Terraform configuration files.
performsbackend initialization , storage for terraform state file.modules installation, downloaded from terraform registry to local pathprovider(s) plugins installation, the plugins are downloaded in the sub-directory of the present working directory at the path of .terraform/plugins
supports -upgrade to update all previously installed plugins to the newest version that complies with the configuration's version constraints
is safe to run multiple times, to bring the working directory up to date with changes in the configuration
does not delete the existing configuration or state
63. **Terraform Workflow - validate:** validates syntactically for format and correctness.
is used to validate/check the syntax of the Terraform files.
verifies whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state.

A syntax check is done on all the terraform files in the directory, and will display an error if any of the files doesn't validate.

64. **Terraform Update - plan:** create a execution plan
traverses each vertex and requests each provider using parallelism
calculates the difference between the last-known state andthe current state and presents this difference as the output of the terraform plan operation to user in their terminal
does not modify the infrastructure or state.
allows a user to see which actions Terraform will perform prior to making any changes to reach the desired state
will scan all *.tf files in the directory and create the plan
will perform refresh for each resource and might hit rate limiting issues as it calls provider APIs
all resources refresh can be disabled or avoided using-refresh=false ortarget=xxxx orbreak resources into different directories.
supports -out to save the plan

65. **Terraform Workflow - apply:** apply changes to reach the desired state.
scans the current directory for the configuration and applies the changes appropriately.
can be provided with a explicit plan, saved as out from terraform plan
If no explicit plan file is given on the command line, terraform apply will create a new plan automatically and prompt for approval to apply it
will modify the infrastructure and the state.
if a resource successfully creates but fails during provisioning,Terraform will error and mark the resource as "tainted".A resource that is tainted has been physically created, but can't be considered safe to use since provisioning failed.Terraform also does not automatically roll back and destroy the resource during the apply when the failure happens, because that would go against the execution plan: the execution plan would've said a resource will be created, but does not say it will ever be deleted.
does not import any resource.
supports -auto-approve to apply the changes without asking for a confirmation
supports -target to apply a specific module

66. **Terraform Workflow - refresh:** used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure
does not modify infrastructure, but does modify the state file

67. **Terraform Workflow - destroy:** destroy the infrastructure and all resources
modifies both state and infrastructure
terraform destroy -target can be used to destroy targeted resources
terraform plan -destroy allows creation of destroy plan

68. **Terraform Workflow - import:** helps import already-existing external resources, not managed by Terraform, into Terraform state and allow it to manage those resources
Terraform is not able to auto-generate configurations for those imported modules, for now, and requires you to first write the resource definition in Terraform and then import this resource
69. **Terraform Workflow - taint:** marks a Terraform-managed resource as tainted, forcing it to be destroyed and recreated on the next apply.
will not modify infrastructure, but does modify the state file in order to mark a resource as tainted. Infrastructure and state are changed in next apply.
can be used to taint a resource within a module
70. **Terraform Workflow - fmt:** format to lint the code into a standard format
71. **Terraform Workflow - console:** command provides an interactive console for evaluating expressions.
72. **Terraform Modules:** enables code reuse
supports versioning to maintain compatibility
stores code remotely
enables easier testing
enables encapsulation with all the separate resources under one configuration block
modules can be nested inside other modules, allowing you to quickly spin up whole separate environments.
can be referred using source attribute
supports Local and Remote modules
Local modules are stored alongside the Terraform configuration (in a separate directory, outside of each environment but in the same repository) with source path ./ or ../Remote modules are stored externally in a separate repository, and supports versioning
supports following backendsLocal pathsTerraform RegistryGitHubBitbucketGeneric Git, Mercurial repositoriesHTTP URLsS3 bucketsGCS buckets
Module requirements
must be on GitHub and must be a public repo, if using public registry.must be named terraform-<PROVIDER>-<NAME>, where <NAME> reflects the type of infrastructure the module manages and <PROVIDER> is the main provider where it creates that infrastructure. for e.g. terraform-google-vault or terraform-aws-ec2-instance.must maintain x.y.z tags for releases to identify module versions. Release tag names must be a semantic version, which can optionally be prefixed with a v for example, v1.0.4 and 0.9.2. Tags that don't look like version numbers are ignored.must maintain a Standard module structure, which allows the registry to

inspect the module and generate documentation, track resource usage, parse submodules and examples, and more.
73. **Terraform Read and write configuration - Resources:** Resourcesresource are the most important element in the Terraform language that describes one or more infrastructure objects, such as compute instances etcresource type and local name together serve as an identifier for a given resource and must be unique within a module for e.g. aws_instance.local_name
74. **Terraform Read and Write configuration - Data Sources:** Resourcesresource are the most important element in the Terraform language that describes one or more infrastructure objects, such as compute instances etcresource type and local name together serve as an identifier for a given resource and must be unique within a module for e.g. aws_instance.local_name
75. **Terraform Read and Write configuration - Variables:** Variablesvariable serve as parameters for a Terraform module and act like function argumentsallows aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurationscan be defined through multiple wayscommand line for e.g.-var="image_id=ami-abc123"variable definition files .tfvars or .tfvars.json. By default, terraform automatically loadsFiles named exactly terraform.tfvars or terraform.tfvars.json.Any files with names ending in .auto.tfvars or .auto.tfvars.jsonfile can also be passed with -var-fileenvironment variables can be used to set variables using the format TF_VAR_nameEnvironment variablesterraform.tfvars file, if present.terraform.tfvars.json file, if present.Any *.auto.tfvars or *.auto.tfvars.json files, processed in lexical order of their filenames.Any -var and -var-file options on the command line, in the order they are provided.Terraform loads variables in the following order, with later sources taking precedence over earlier ones:
76. **Terraform Read and Write Configurations - Local Values:** locals assigns a name to an expression, allowing it to be used multiple times within a module without repeating it.
are like a function's temporary local variables.
helps to avoid repeating the same values or expressions multiple times in a configuration.
77. **Terraform Read and Write Configurations - Output:** are like function return values.
output can be marked as containing sensitive material using the optional sensitive argument, which prevents Terraform from showing its value in the list of outputs. However, they are still stored in the state as plain text.
In a parent module, outputs of child modules are available in expressions as module.<MODULE NAME>.<OUTPUT NAME>.

78. **Terraform Read and Write Configurations - Named Values:** is an expression that references the associated value for e.g. aws_instance.local_name, data.aws_ami.centos, var.instance_type etc.
support Local named values for e.g count.index
79. **Terraform Read and Write Configurations - Dependencies:** identifies implicit dependencies as Terraform automatically infers when one resource depends on another by studying the resource attributes used in interpolation expressions for e.g aws_eip on resource aws_instance
explicit dependencies can be defined using depends_on where dependencies between resources that are not visible to Terraform
80. **Terraform Read and Write Configurations - Data Types:** supports primitive data types ofstring, number and boolTerraform language will automatically convert number and bool values to string values when needed
supports complex data types oflist - a sequence of values identified by consecutive whole numbers starting with zero.map - a collection of values where each is identified by a string label.set - a collection of unique values that do not have any secondary identifiers or ordering.
supports structural data types ofobject - a collection of named attributes that each have their own typetuple - a sequence of elements identified by consecutive whole numbers starting with zero, where each element has its own type.
81. **Terraform Read and Write Configurations - Built In Functions:** includes a number of built-in functions that can be called from within expressions to transform and combine values for e.g. min, max, file, concat, element, index, lookup etc.
does not support user-defined functions
82. **Terraform Read and Write Configurations - Dynamic Blocks:** acts much like a for expression, but produces nested blocks instead of a complex typed value. It iterates over a given complex value, and generates a nested block for each element of that complex value.
83. **Terraform Read and Write Configurations - Terraform Comments:** supports three different syntaxes for comments:#///* and */
84. **Terraform Backends:** determines how state is loaded and how an operation such as apply is executed
are responsible for storing state and providing an API for optional state locking needs to be initialized
if switching the backed for the first time setup, Terraform provides a migration option helpscollaboration and working as a team, with the state maintained remotely and state lockingcan provide enhanced security for sensitive datasupport remote operations
supports local vs remote backendslocal (default) backend stores state in a local

JSON file on diskremote backend stores state remotely like S3, OSS, GCS, Consul and support features like remote operation, state locking, encryption, versioning etc.
supports partial configuration with remaining configuration arguments provided as part of the initialization process
Backend configuration doesn't support interpolations.
GitHub is not the supported backend type in Terraform.
85. **Terraform State Management:** state helps keep track of the infrastructure Terraform manages
stored locally in the terraform.tfstate
recommended not to edit the state manually
Use terraform state commandmv - to move/rename modulesrm - to safely remove resource from the state. (destroy/retain like)pull - to observe current remote statelist & show - to write/debug modules
86. **State Locking:** happens for all operations that could write state, if supported by backend
prevents others from acquiring the lock & potentially corrupting the state
backends which support state locking areazurermHashicorp consulTencent Cloud Object Storage (COS)etcdv3Google Cloud Storage GCSHTTP endpointsKubernetes Secret with locking done using a Lease resourceAliCloud Object Storage OSS with locking via TableStorePostgreSQLAWS S3 with locking via DynamoDBTerraform Enterprise
Backends which do not support state locking areartifactoryetcd
can be disabled for most commands with the -lock flag
use force-unlock command to manually unlock the state if unlocking failed
87. **State Security:** can contain sensitive data, depending on the resources in use for e.g passwords and keys
using local state, data is stored in plain-text JSON files
using remote state, state is held in memory when used by Terraform. It may be encrypted at rest, if supported by backend for e.g. S3, OSS
88. **Terraform Logging:** debugging can be controlled using TF_LOG , which can be configured for different levels TRACE, DEBUG, INFO, WARN or ERROR, with TRACE being the more verbose.
logs path can be controlled TF_LOG_PATH. TF_LOG needs to be specified.
89. **Terraform Cloud and Terraform Enterprise:** Terraform Cloud provides Cloud Infrastructure Automation as a Service. It is offered as a multi-tenant SaaS platform and is designed to suit the needs of smaller teams and organizations. Its smaller plans default to one run at a time, which prevents users from executing multiple runs concurrently.
Terraform Enterprise is a private install for organizations who prefer to self-manage.

It is designed to suit the needs of organizations with specific requirements for security, compliance and custom operations.
90. **Remote Terraform Execution:** supports Remote Operations for Remote Terraform execution which helps provide consistency and visibility for critical provisioning operations.
91. **Workspaces:** organizes infrastructure with workspaces instead of directories. Each workspace contains everything necessary to manage a given collection of infrastructure, and Terraform uses that content whenever it executes in the context of that workspace.
92. **Remote State Management:** acts as a remote backend for the Terraform state. State storage is tied to workspaces, which helps keep state associated with the configuration that created it.
93. **Version Control Integration:** is designed to work directly with the version control system (VCS) provider.
94. **Private Module Registry:** provides a private and central library of versioned & validated modules to be used within the organization
95. **Team based Permission System:** can define groups of users that match the organization's real-world teams and assign them only the permissions they need
96. **Sentinel Policies:** embeds the Sentinel policy-as-code framework, which lets you define and enforce granular policies for how the organization provisions infrastructure. Helps eliminate provisioned resources that don't follow security, compliance, or operational policies.
97. **Cost Estimation:** an display an estimate of its total cost, as well as any change in cost caused by the proposed updates
Security - encrypts state at rest and protects it with TLS in transit.
98. **Terraform Enterprise features:** includes all the Terraform Cloud features with Audit - supports detailed audit logging and tracks the identity of the user requesting state and maintains a history of state changes.
SSO/SAML - SAML for SSO provides the ability to govern user access to your applications.
99. **Terraform Enterprise currently supports running under the following operating systems for a Clustered deployment::** Ubuntu 16.04.3 - 16.04.5 / 18.04
Red Hat Enterprise Linux 7.4 through 7.7
CentOS 7.4 - 7.7
Amazon Linux
Oracle Linux
Clusters currently don't support other Linux variants.
100. **Terraform Cloud currently supports following VCS Provider:** GitHub.com
GitHub.com (OAuth)

GitHub Enterprise
GitLab.com
GitLab EE and CE
Bitbucket Cloud
Bitbucket Server
Azure DevOps Server
Azure DevOps Services

101. **Terraform - air-gapped install:** A Terraform Enterprise install that is provisioned on a network that does not have Internet access is generally known as an air-gapped install. These types of installs require you to pull updates, providers, etc. from external sources vs. being able to download them directly.