Set 2 - Advanced Questions

Date: - 13/05/2020

Version: - 1.0

Total: - 50 Questions

Question 1:

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?

- ○ s3
- ○ github
- ○ consul
- ○ artifactory
- ○ terraform enterprise

Answer**: B**

Explanation

github is not a supported backend type.

https://www.terraform.io/docs/backends/types/index.html

Question 2:

The Terraform language supports a number of different syntaxes for comments. Select all that are supported. (select three)

- [ ] //
- [ ] <* and *>
- [ ] #
- [ ] /* and */

Answer**: A, C, D**

Explanation

https://www.terraform.io/docs/configuration/syntax.html#comments

Question 3:

In terraform, most resource dependencies are handled automatically. Which of the following statements describes best how terraform resource dependencies are handled?

- ○ Resource dependencies are handled automatically by the `depends_on` meta_argument, which is set to true by default.
- ○ Terraform analyses 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.
- ○ The terraform binary contains a built-in reference map of all defined Terraform resource dependencies. Updates to this dependency map are reflected in terraform versions. To ensure you are working with the latest resource dependency map you much be running the latest version of Terraform.
- ○ Resource dependencies are identified and maintained in a file called `resource.dependencies`. Each terraform provider is required to maintain a list of all resource dependencies for the provider and it's included with the plugin during initialization when `terraform init` is executed. The file is located in the `terraform.d` folder.

Answer**: B**

Explanation

https://www.terraform.io/docs/configuration/resources.html

Question 4:

Which of the following statements best describes the Terraform `list(...)` type?

- a sequence of values identified by consecutive whole numbers starting with zero.
- a collection of named attributes that each have their own type.
- a collection of values where each is identified by a string label.
- a collection of unique values that do not have any secondary identifiers or ordering.

Answer**: A**

Explanation

A terraform list is a sequence of values identified by consecutive whole numbers starting with zero.

https://www.terraform.io/docs/configuration/types.html#structural-types

Question 5:

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]
}
```

- ○ implicit dependency
- ○ internal dependency
- ○ non-dependency resource
- ○ explicit dependency

Answer**: D**

Explanation

Sometimes there are dependencies between resources that are *not* visible to Terraform. The `depends_on` argument is accepted by any resource and accepts a list of resources to create *explicit dependencies* for.

Question 6:

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 to?

- ○ The .terraform.d directory in the directory terraform init was executed in.
- ○ The .terraform/plugins directory in the directory terraform init was executed in.
- ○ /etc/terraform/plugins
- ○ The .terraform.plugins directory in the directory terraform init was executed in.

Answer**: B**

Explanation

By default, `terraform init` downloads plugins into a subdirectory of the working directory, .terraform, so that each working directory is self-contained.

Question 7:

The following is a snippet from a Terraform configuration file:

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

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

which, when validated, results in 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 "_____"
argument for alternative configurations.
```

Fill in the blank in the error message with the correct string from the list below.

- label
- alias
- version
- multi

Answer**: B**

Explanation

An alias meta-arguement is used when using the same provider with different configurations for different resources.

https://www.terraform.io/docs/configuration/providers.html#alias-multiple-provider-instance

Question 8:

Which of the following is an *invalid* variable name?

- count
- instance_name
- var1
- web

Answer**: A**

Explanation

`count` is a reserved word. The `count` parameter on resources can simplify configurations and let you scale resources by simply incrementing a number.

https://www.terraform.io/intro/examples/count.html

Question 9:

Environment variables can be used to set variables. The environment variables must be in the format `"____"_<variablename>`. Select the correct prefix string from the following list.

- TF_ENV
- TF_VAR_NAME
- TF_VAR
- TF_ENV_VAR

Answer**: C**

Explanation

Environment variables can be used to set variables. The environment variables must be in the format `TF_VAR_name` and this will be checked last for a value. For example:

```
export TF_VAR_region=us-west-1
export TF_VAR_ami=ami-049d8641
export TF_VAR_alist='[1,2,3]'
export TF_VAR_amap='{ foo = "bar", baz = "qux" }'
```

https://www.terraform.io/docs/commands/environment-variables.html

Question 10:

What is the result of the following terraform function call?

```
> zipmap(["a", "b"], [1, 2])
```

- {
  "a" = 1
  "b" = 2
  }
- [
  "a" = 1
  "b" = 2
  ]
- {
  "a",
  "b",
  "1",
  "2",
  }
- [
  "a",
  "b",
  "1",
  "2",
  ]

Answer**: A**

Explanation

`zipmap` constructs a map from a list of keys and a corresponding list of values. A map is denoted by { } whereas a list is donated by [ ].

https://www.terraform.io/docs/configuration/functions/zipmap.html

Question 11:

What is the result of the following terraform function call?

```
> index(["a", "b", "c"], "c")
```

- ○ true
- ○ 2
- ○ 0
- ○ 1

Answer**: B**

Explanation

`index` finds the element index for a given value in a list starting with index 0.

https://www.terraform.io/docs/configuration/functions/index.html

Question 12:

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 `terraform plan` and state files.

- ○ True
- ○ False

Answer**: FALSE**

Explanation

Currently, Terraform has no mechanism to redact or protect secrets that are returned via data sources, so secrets read via this provider will be persisted into the Terraform state, into any plan files, and in some cases in the console output produced while planning and applying. These artifacts must, therefore, all be protected accordingly.

Question 13:

True or False? Each Terraform workspace uses its own state file to manage the infrastructure associated with that particular workspace.

- ○ False
- ○ True

Answer**: TRUE**

Explanation

The persistent data stored in the backend belongs to a *workspace*. Initially, the backend has only one workspace, called "default", and thus there is only one Terraform state associated with that configuration.

Question 14:

Terraform has detailed logs which can be enabled by setting the _____ environmental variable.

- TF_DEBUG
- TF_LOG
- TF_INFO
- TF_TRACE

Answer**: B**

Explanation

Terraform has detailed logs that can be enabled by setting the `TF_LOG` environment variable to any value. This will cause detailed logs to appear on stderr.

You can set `TF_LOG` to one of the log levels `TRACE`, `DEBUG`, `INFO`, `WARN` or `ERROR` to change the verbosity of the logs. `TRACE` is the most verbose and it is the default if `TF_LOG` is set to something other than a log level name.

https://www.terraform.io/docs/internals/debugging.html

Question 15:

Which of the following is not a valid Terraform string function?

- join
- tostring
- replace
- format

Answer**: B**

Explanation

`tostring` is not a string function, it is a type conversion function. `tostring` converts its argument to a string value.
https://www.terraform.io/docs/configuration/functions/tostring.html

Question 16:

Which Terraform command will check and report errors within modules, attribute names, and value types to make sure they are syntactically valid and internally consistent?

- `terraform format`
- `terraform fmt`
- `terraform show`
- `terraform validate`

Answer**: D**

Explanation

The `terraform validate` command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

Validate runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state. It is thus primarily useful for general verification of reusable modules, including the correctness of attribute names and value types.

Question 17:

While Terraform is generally written using the HashiCorp Configuration Language (HCL), what other syntax can Terraform be expressed in?

- ○ TypeScript
- ○ JSON
- ○ YAML
- ○ XML

Answer**: B**

Explanation

The constructs in the Terraform language can also be expressed in JSON syntax, which is harder for humans to read and edit but easier to generate and parse programmatically.

Question 18:

When using providers that require the retrieval of data, such as the HashiCorp Vault provider, in what phase does Terraform actually retrieve the data required?

- `terraform delete`
- `terraform apply`
- `terraform init`
- `terraform plan`

Answer**: D**

Explanation

It is important to consider that Terraform reads from data sources during the `plan` phase and writes the result into the plan. For something like a Vault token which has an explicit TTL, the `apply` must be run before the data, or token, in this case, expires, otherwise, Terraform will fail during the apply phase.

Question 19:

Why might a user opt to include the following snippet in their configuration file?

```
terraform {
  required_version = ">= 0.12"
}
```

- this ensures that all Terraform providers are above a certain version to match the application being deployed
- Terraform 0.12 introduced substantial changes to the syntax used to write Terraform configuration
- The user wants to ensure that the application being deployed is a minimum version of 0.12
- versions before Terraform 0.12 were not approved by HashiCorp to be used in production

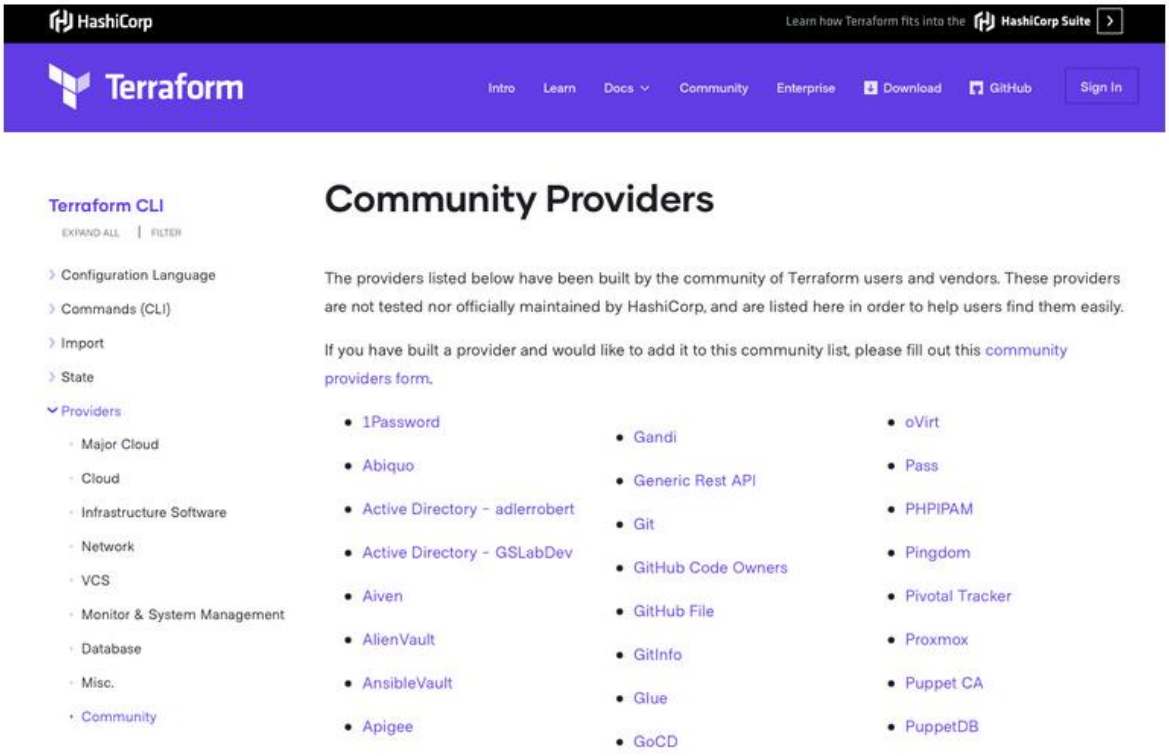
Answer**: B**

Explanation

You can use `required_version` to ensure that a user deploying infrastructure is using Terraform 0.12 or greater, due to the vast number of changes that were introduced. As a result, many previously written configurations had to be converted or rewritten.

Question 20:

True or False? `terraform init` cannot automatically download Community providers.



- ◯ False
- ◯ True

Answer**: TRUE**

Explanation

Anyone can develop and distribute their own Terraform providers. (See Writing Custom Providers for more about provider development.) These third-party providers must be manually installed, since `terraform init` cannot automatically download them.

https://www.terraform.io/docs/configuration/providers.html#third-party-plugins

Question 21:

Select the most accurate statement to describe the Terraform language from the following list.

- Terraform is a mutable, declarative, Infrastructure as Code configuration management language based on Hashicorp Configuration Language, or optionally JSON.
- Terraform is a mutable, procedural, Infrastructure as Code provisioning language based on Hashicorp Configuration Language, or optionally YAML.
- Terraform is an immutable, procedural, Infrastructure as Code configuration management language based on Hashicorp Configuration Language, or optionally JSON.
- Terraform is an immutable, declarative, Infrastructure as Code provisioning language based on Hashicorp Configuration Language, or optionally JSON.

Answer**: D**

Explanation

Terraform is not a configuration management tool - https://www.terraform.io/intro/vs/chef-puppet.html

Terraform is a declarative language - https://www.terraform.io/docs/configuration/index.html

Terraform supports a syntax that is JSON compatible - https://www.terraform.io/docs/configuration/syntax-json.html

Terraform is primarily designed on immutable infrastructure principles - https://www.hashicorp.com/resources/what-is-mutable-vs-immutable-infrastructure

Question 22:

When Terraform needs to be installed in a location where it does not have internet access to download the installer and upgrades, the installation is generally known as to be _____.

- ○ non-traditional
- ○ air-gapped
- ○ disconnected
- ○ a private install

Answer**: B**

Explanation

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.

Question 23:

In order to reduce the time it takes to provision resources, Terraform uses parallelism. By default, how many resources will Terraform provision concurrently?

- ○ 10
- ○ 5
- ○ 50
- ○ 20

Answer**: A**

Explanation

Terraform can limit the number of concurrent operations as Terraform walks the graph using the `-parallelism=n` argument. The default value for this setting is 10. This setting might be helpful if you're running into API rate limits.

Question 24:

Select all features which are exclusive to Terraform Enterprise. (select three)

- [ ] Sentinel
- [ ] Audit Logs
- [ ] Cost Estimation
- [ ] Clustering
- [ ] SAML/SSO

Answer**: B, D, E**

Explanation

Sentinel and Cost Estimation are both available in Terraform Cloud, though not at the free tier level.

https://www.hashicorp.com/products/terraform/pricing/

Question 25:

Select the operating systems which are supported for a clustered Terraform Enterprise: (select three)

- [ ] Red Hat
- [ ] Ubuntu
- [ ] CentOS
- [ ] Amazon Linux

Answer**: A, B, C**

Explanation

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

Clusters currently don't support other Linux variants.

https://www.terraform.io/docs/enterprise/before-installing/index.html#operating-system-requirements

Question 26:

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?

- terraform workspace create stage
- terraform workspace -new stage
- terraform workspace new stage
- terraform workspace -create stage

Answer**: C**

Explanation

The terraform workspace new command is used to create a new workspace.
https://www.terraform.io/docs/commands/workspace/new.html

Question 27:

Which of the following variable declarations is going to result in an error?

```
variable "example" {}
```

```
variable "example" {
  description = "This is a variable description"
  type        = list(string)
  default     = {}
}
```

```
variable "example" {
  type = object({})
}
```

```
variable "example" {
  description = "This is a test"
  type        = map
  default     = {"one" = 1, "two" = 2, "Three" = "3"}
}
```

Answer**: B**

Explanation

Lists are defined with [ ], maps are defined with { }.

https://www.terraform.io/docs/configuration/types.html#structural-types

Question 28:

Complete the following sentence:

The terraform `state` command can be used to ___

- ○ refresh state
- ○ There is no such command
- ○ modify state
- ○ view state

Answer**: C**

Explanation

The `terraform state` command is used for advanced state management. Rather than modify the state directly, the `terraform state` commands can be used in many cases instead.

https://www.terraform.io/docs/commands/state/index.html

Question 29:

Complete the following sentence:

For local state, the workspaces are stored directly in a...

- ○ a file called terraform.tfstate
- ○ directory called terraform.workspaces.tfstate
- ○ a file called terraform.tfstate.backup
- ○ directory called terraform.tfstate.d

Answer**: D**

Explanation

For local state, Terraform stores the workspace states in a directory called `terraform.tfstate.d`. https://www.terraform.io/docs/state/workspaces.html#workspace-internals

Question 30:

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 = "~> 1.2.0"
  }
}
```

- [ ] 1.3.0
- [ ] 1.2.9
- [ ] 1.2.3
- [ ] 1.3.1

Answer**: B, C**

Explanation

~> 1.2.0 will match any non-beta version of the provider between >= 1.2.0 and < 1.3.0. For example, 1.2.X

Question 31:

When multiple arguments with single-line values appear on consecutive lines at the same nesting level, HashiCorp recommends that you:

place all arguments using a variable at the top

```
ami = var.aws_ami
instance_type = var.instance_size
subnet_id = "subnet-0bb1c79de3EXAMPLE"
tags = {
  Name = "HelloWorld"
}
```

place a space in between each line

```
type = "A"

ttl = "300"

zone_id = aws_route53_zone.primary.zone_id
```

align their equals signs

```
ami           = "abc123"
instance_type = "t2.micro"
```

put arguments in alphabetical order

```
name = "www.example.com"
records = [aws_eip.lb.public_ip]
type = "A"
ttl = "300"
zone_id = aws_route53_zone.primary.zone_id
```

Answer**: C**

Explanation

HashiCorp style conventions suggest you that align the equals sign for consecutive arguments for easing readability for configurations.

```
ami           = "abc123"
instance_type = "t2.micro"
```

Question 32:

Which statements best describes what the local variable assignment is doing in the following code snippet:

```
variable "subnet_details" {
  type = list(object({
    cidr             = string
    subnet_name      = string
    route_table_name = string
    aznum            = number
  }))
}

locals {
  route_tables_all = distinct([for s in var.subnet_details : s.route_table_name ])
}
```

- ○ Create a map of route table names from a list of subnet names
- ○ Create a map of route table names to subnet names
- ○ Create a list of route table names eliminating duplicates
- ○ Create a distinct list of route table name objects

Answer**: C**

Explanation

`route_tables_all` is assigned a list of unique route table names filtered from a list of objects describing subnet details, one of those object attributes being `route_table_name`.

Question 33:

Select all Operating Systems that Terraform is available for. (select five)

- [ ] Solaris
- [ ] Unix
- [ ] Linux
- [ ] FreeBSD
- [ ] macOS
- [ ] Windows

Answer**: A, C, D, E, F**

Explanation

Terraform is available for macOS, FreeBSD, OpenBSD, Linux, Solaris, Windows

https://www.terraform.io/downloads.html

Question 34:

In the following code snippet, the `block type` is identified by which string?

```
resource "aws_instance" "db" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
}
```

- instance_type
- "db"
- "aws_instance"
- resource

Answer**: 4**

Explanation

The format of resource block configurations is as follows:

<block type> "<resource type>" "<local name/label>"

Question 35:

When writing Terraform code, HashiCorp recommends that you use how many spaces between each nesting level?

- 4
- 5
- 2
- 1

Answer**: C**

Explanation

HashiCorp style conventions state that you should use 2 spaces between each nesting level to improve the readability of Terraform configurations.

Check this link for more information

Question 36:

During a `terraform plan`, a resource is successfully created but eventually fails during provisioning. What happens to the resource?

- ◯ Terraform attempts to provision the resource up to three times before exiting with an error
- ◯ the terraform plan is rolled back and all provisioned resources are removed
- ◯ it is automatically deleted
- ◯ the resource is marked as tainted

Answer**: D**

Explanation

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.

Question 37:

Choose the correct answer which fixes the syntax of the following Terraform code:

```
resource "aws_security_group" "vault_elb" {
  name        = "${var.name_prefix}-vault-elb"
  description = Vault ELB
  vpc_id      = var.vpc_id
}
```

A.

```
resource "aws_security_group" "vault_elb" {
  name        = "${var.name_prefix}-vault-elb"
  description = "Vault ELB"
  vpc_id      = var.vpc_id
}
```

B.

```
resource "aws_security_group" "vault_elb" {
  name        = "${var.name_prefix}-vault-elb"
  description = var_Vault ELB
  vpc_id      = var.vpc_id
}
```

C.

```
resource "aws_security_group" "vault_elb" {
  name        = "${var.name_prefix}-vault-elb"
  description = "${Vault ELB}"
  vpc_id      = var.vpc_id
}
```

D.

```
resource "aws_security_group" "vault_elb" {
  name        = "${var.name_prefix}-vault-elb"
  description = [Vault ELB]
  vpc_id      = var.vpc_id
}
```

Answer**: A**

Explanation

When assigning a value to an argument, it must be enclosed in quotes ("...") unless it is being generated programmatically.

Question 38:

What feature of Terraform Cloud and/or Terraform Enterprise can you publish and maintain a set of custom modules which can be used within your organization?

- ○ remote runs
- ○ Terraform registry
- ○ private module registry
- ○ custom VCS integration

Answer**: C**

Explanation

You can use modules from a private registry, like the one provided by Terraform Cloud. Private registry modules have source strings of the form `<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>`. This is the same format as the public registry, but with an added hostname prefix.

Question 39:

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?

- `terraform workspace dev`
- `terraform workspace -switch dev`
- `terraform workspace switch dev`
- `terraform workspace select dev`

Answer**: D**

Explanation

The `terraform workspace select` command is used to choose a different workspace to use for further operations.

https://www.terraform.io/docs/commands/workspace/select.html

Question 40:

Anyone can publish and share modules on the `Terraform Public Module Registry`, and meeting the requirements for publishing a module is extremely easy. Select from the following list all valid requirements. (select three)

- [ ] The registry uses tags to identify module versions. Release tag names must be for the format *x.y.z*, and can optionally be prefixed with a `v`.
- [ ] The module must be on GitHub and must be a public repo.
- [ ] The module must be PCI/HIPPA compliant.
- [ ] Module repositories must use this three-part name format, *terraform-<PROVIDER>-<NAME>*.

Answer**: A, B, D**

Explanation

The list below contains all the requirements for publishing a module. Meeting the requirements for publishing a module is extremely easy. The list may appear long only to ensure we're detailed, but adhering to the requirements should happen naturally.

GitHub. The module must be on GitHub and must be a public repo. This is only a requirement for the public registry. If you're using a private registry, you may ignore this requirement.

Named terraform-<PROVIDER>-<NAME>. Module repositories must use this three-part name format, where <NAME> reflects the type of infrastructure the module manages and <PROVIDER> is the main provider where it creates that infrastructure. The <NAME> segment can contain additional hyphens. Examples: terraform-google-vault or terraform-aws-ec2-instance.

Repository description. The GitHub repository description is used to populate the short description of the module. This should be a simple one-sentence description of the module.

Standard module structure. The module must adhere to the standard module structure. This allows the registry to inspect your module and generate documentation, track resource usage, parse submodules and examples, and more.

x.y.z tags for releases. The registry uses tags 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`. To publish a module initially, at least one release tag must be present. Tags that don't look like version numbers are ignored.

https://www.terraform.io/docs/registry/modules/publish.html#requirements

Question 41:

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 EC2 instance labeled web_server
- ○ The EIP with an id of ami-2757f631
- ○ The AMI used fo the EC2 instance
- ○ The S3 bucket labeled company_data

Answer**: A**

Explanation

The EC2 instance labeled *web_server* is the implicit dependency as the `aws_eip` cannot be created until the `aws_instance` labeled `web_server` has been provisioned and the `id` is available.

Note that `aws_s3_bucket.example` is an explicit dependency.

Question 42:

Which of the following commands will launch the Interactive console for Terraform interpolations?

- terraform
- terraform cli
- terraform cmdline
- terraform console

Answer**: D**

Explanation

The `terraform console` command provides an interactive console for evaluating expressions.

https://www.terraform.io/docs/commands/console.html

Question 43:

True or False. The `terraform refresh` command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. If drift is detected between the real-world infrastructure and the last known-state, it will modify the infrastructure to correct the drift.

- ○ False
- ○ True

Answer**: FALSE**

Explanation

The `terraform refresh` command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file.

This does not modify infrastructure but *does* modify the state file. If the state is changed, this may cause changes to occur during the next plan or apply.

https://www.terraform.io/docs/commands/refresh.html

Question 44:

Terraform Enterprise (also referred to as pTFE) requires what type of backend database for a clustered deployment?

- MSSQL
- PostgreSQL
- MySQL
- Cassandra

Answer**: B**

Explanation

External Services mode stores the majority of the stateful data used by the instance in an external PostgreSQL database and an external S3-compatible endpoint or Azure blob storage. There is still critical data stored on the instance that must be managed with snapshots. Be sure to check the PostgreSQL Requirements for information that needs to be present for Terraform Enterprise to work. This option is best for users with expertise managing PostgreSQL or users that have access to managed PostgreSQL offerings like AWS RDS.

Question 45:

Which of the following terraform subcommands could be used to remove the lock on the state for the current configuration?

- unlock
- force-unlock
- Removing the lock on a state file is not possible
- state-unlock

Answer**: B**

Explanation

`terraform force-unlock` removes the lock on the state for the current configuration.
https://www.terraform.io/docs/commands/force-unlock.html

Question 46:

What is the result of the following terraform function call?

```
> lookup({a="hello", b="goodbye"}, "c", "what?")
```

- goodbye
- what?
- c
- hello

Answer**: B**

Explanation

`lookup` retrieves the value of a single element from a map, given its key. If the given key does not exist, a the given default value is returned instead. In this case, the function call is searching for the key "c". Because there there is no key "c", the default vault "what?" is returned.

https://www.terraform.io/docs/configuration/functions/lookup.html

Question 47:

A user has created a module called "*my_test_module*" and committed it to GitHub. Over time, several commits have been made with updates to the module, each tagged in GitHub with an incremental version number. Which of the following lines would be required in a module configuration block in terraform to select tagged version v1.0.4?

- source = "git::https://example.com/my_test_module.git?ref=v1.0.4"
- source = "git::https://example.com/my_test_module.git#tag=v1.0.4"
- source = "git::https://example.com/my_test_module.git@tag=v1.0.4"
- source = "git::https://example.com/my_test_module.git&ref=v1.0.4"

Answer**: A**

Explanation

By default, Terraform will clone and use the default branch (referenced by `HEAD`) in the selected repository. You can override this using the `ref` argument:

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

The value of the ref argument can be any reference that would be accepted by the git checkout command, including branch and tag names.

https://www.terraform.io/docs/modules/sources.html#selecting-a-revision

Question 48:

Provider dependencies are created in several different ways. Select the valid provider dependencies from the following list: (select three)

- [ ] Explicit use of a provider block in configuration, optionally including a version constraint.
- [ ] Use of any resource belonging to a particular provider in a resource or data block in configuration.
- [ ] Existence of any resource instance belonging to a particular provider in the current *state*.
- [ ] Existence of any provider plugins found locally in the working directory.

Answer**: A, B, C**

Explanation

The existence of a provider plugin found locally in the working directory does not itself create a provider dependency. The plugin can exist without any reference to it in the terraform configuration.

https://www.terraform.io/docs/commands/providers.html

Question 49:

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)

- [ ] CVS Version Control
- [ ] Bitbucket Cloud
- [ ] Azure DevOps Server
- [ ] GitHub
- [ ] GitHub Enterprise

Answer**: B, C, D, E**

Explanation

Terraform Cloud supports the following VCS providers:

- GitHub
- GitHub.com (OAuth)
- GitHub Enterprise
- GitLab.com
- GitLab EE and CE
- Bitbucket Cloud
- Bitbucket Server
- Azure DevOps Server
- Azure DevOps Services

https://www.terraform.io/docs/cloud/vcs/index.html#supported-vcs-providers

Question 50:

True or False? By default, Terraform `destroy` will prompt for confirmation before proceeding.

- ○ True
- ○ False

Answer**: TRUE**

Explanation

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:
```