GSS Terraform Quiz 1

1. A resource I’m looking to create has a parameter that seems like we might want to adjust in the future, however, for now, the value being used is always the same. What would be a good approach to minimize the need to configure this parameter each time we create the resource, but also enable easy adjustment in the future if we decide to change the parameter?

To allow you to make your code more DRY and more configurable, Terraform allows you to define input variables.

description

It’s always a good idea to use this parameter to document how a variable is used. Your teammates will not only be able to see this description while reading the code, but also when running the plan or apply commands.

default

There are a number of ways to provide a value for the variable, including passing it in at the command line (using the -var option), via a file (using the -var-file option), or via an environment variable (Terraform looks for environment variables of the name TF\_VAR\_<variable\_name>). If no value is passed in, the variable will fall back to this default value. If there is no default value, Terraform will interactively prompt the user for one.

type

This allows you enforce type constraints on the variables a user passes in. Terraform supports a number of type constraints, including string, number, bool, list, map, set, object, tuple, and any. If you don’t specify a type, Terraform assumes the type is any.

In addition to input variables, Terraform also allows you to define output variables

The NAME is the name of the output variable and VALUE can be any Terraform expression that you would like to output. The CONFIG can contain two additional parameters, both optional:

description

It’s always a good idea to use this parameter to document what type of data is contained in the output variable.

sensitive

Set this parameter to true to tell Terraform not to log this output at the end of terraform apply. This is useful if the output variable contains sensitive material or secrets, such as passwords or private keys.

Input and output variables are also essential ingredients in creating configurable and reusable infrastructure code

1. If a secret (password, ssh key, etc) is needed to create a resource in an environment, what is the correct approach to create and use this secret within Terraform?

To keep secrets out of version control is to define them as variables in a separate tfvars file. The secret variables can now be sourced from a secret tfvars file that is added to your .gitignore and typically contains all the secrets for your Terraform project

Most of the remote backends natively support encryption in transit and encryption on disk of the state file. Moreover, those backends usually expose ways to configure access permissions (e.g., using IAM policies with an S3 bucket), so you can control who has access to your state files and the secrets the may contain.

Encrypting secrets within the state file, but these remote backends reduce most of the security concerns, as at least the state file isn’t stored in plaintext on disk anywhere.

If you’re using Terraform with AWS, Amazon S3 which is Amazon’s managed file store, is typically your best bet as a remote backend

One option for handling secrets is to use a Terraform data source to read the secrets from a secret store. For example, you can store secrets, such as database passwords, in AWS Secrets Manager, a managed service AWS offers specifically for storing sensitive data. You could use the AWS Secrets Manager UI to store the secret and then read the secret back out in your Terraform code using the aws\_secretsmanager\_secret\_version data source

The second option for handling secrets is to manage them completely outside of Terraform (e.g., in a password manager such as 1Password, LastPass, or OS X Keychain) and to pass the secret into Terraform via an environment variable. To do that, declare a variable in variables.tf

Some of the supported secret stores and data source combos you could look into are:

* AWS Secrets Manager and the aws\_secretsmanager\_secret\_versiondata source.
* AWS Systems Manager Parameter Store and the aws\_ssm\_parameter data source.
* AWS KMS and the aws\_kms\_secrets data source.

1. I’m setting up a new application. Some resources needed by the application only need to be created once, while others need to have a resource created each time, we stand up a new environment. What are some options we have available to allow for the correct resources to be created?

Using separate Terraform folders (and therefore separate state files) for each environment (staging, production, etc.) and for each component (vpc, services, databases).

By defining infrastructure as code in modules is the best practices to your infrastructure. To validate each change made to a module through code and reviews and automated tests;

We can create semantically versioned releases of each module; and we can safely try out different versions of a module in different environments and roll back to previous versions if we hit a problem.

1. Give an example of the steps needed to run the command terraform plan after opening a GSS Terraform project in VSCode.

Check the init.vars and check terraform version which the code support and ensure that we have that particular version of terraform

. init.sh dev, prod, qa or terraform init

terraform validate

terraform plan