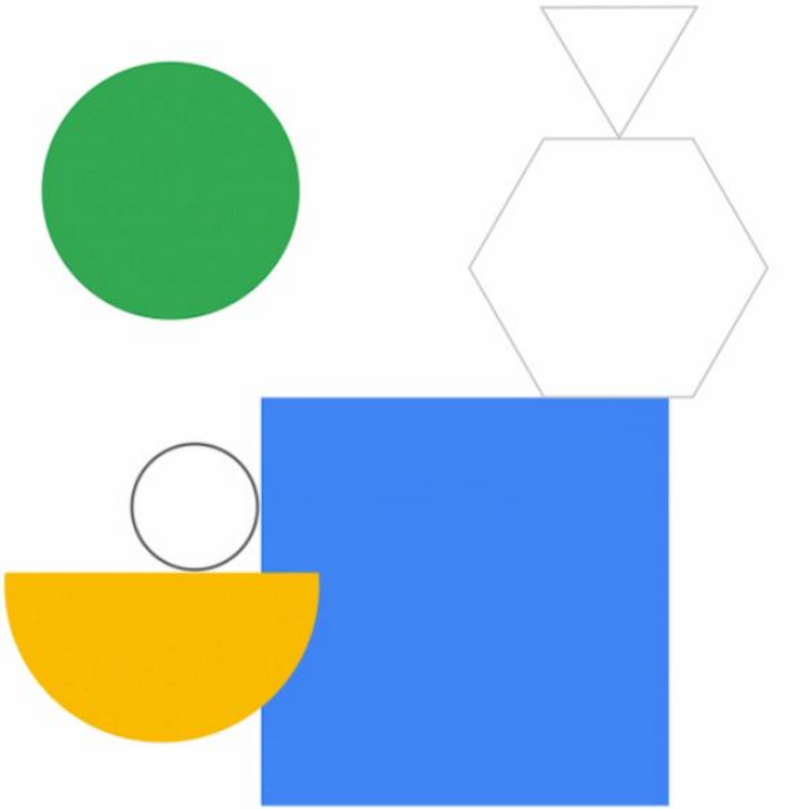
Google Cloud

Organizing and Reusing Configuration with Terraform Modules



Objectives

After you complete this module, you will be able to:

01 Define Terraform modules.

02 Use modules to reuse configurations.

03 Use modules from the public registry.

04 Use input variables to parameterize configurations.

05 Use output values to access resource attributes outside the module.

Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	Use variables to parameterize a module
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

Problem: Updating repeated code

Web Server

- VM image
- Machine type
- Static IP
- Service account

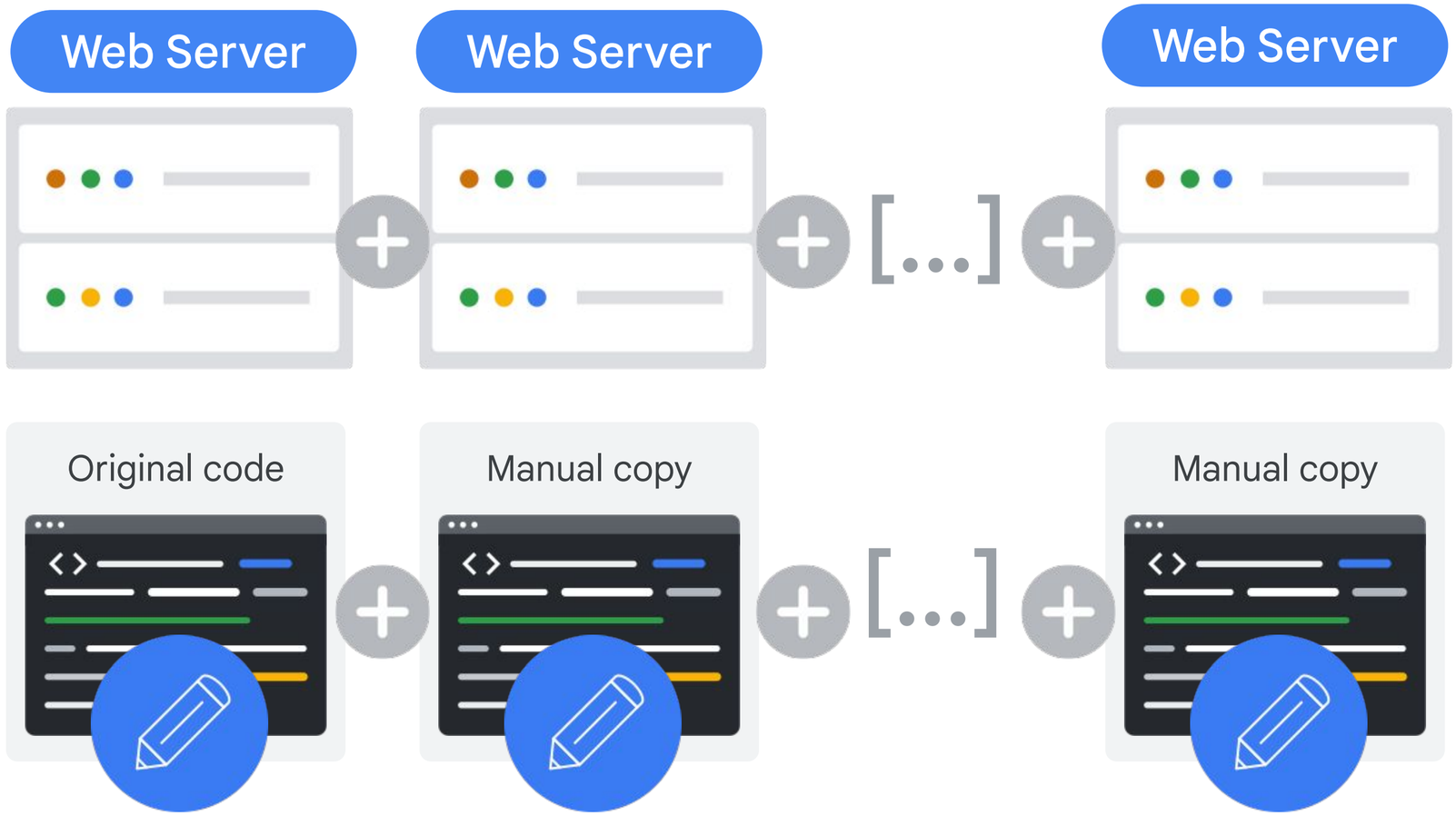
```
resource "google_compute_instance" "serverVM" {
#All necessary parameters defined
    machine_type = "f1-micro"
}

resource “google_compute_address” “static_ip”{
..
}
resource "google_compute_disk" "server_disk" {
..
}

resource "google_service_account" "service_account" {
..
}
```

Disadvantages of code repetition

- Unmanageable
- Error prone
- Inefficient



Solution: Create modules

```
-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf
```

- DRY (don’t repeat yourself)
- Define the reusable code within a module named server.

```
resource "google_compute_instance" "serverVM" {
#All necessary parameters defined
    machine_type = "f1-micro"
    boot_disk {
     initialize_params {
       image = "debian-cloud/debian-9"
     }
  }
}

resource “google_compute_address” “static_ip”{
..
}
resource "google_compute_network" "mynetwork" {
..
}

resource "google_compute_firewall" "default" {
..
}
```

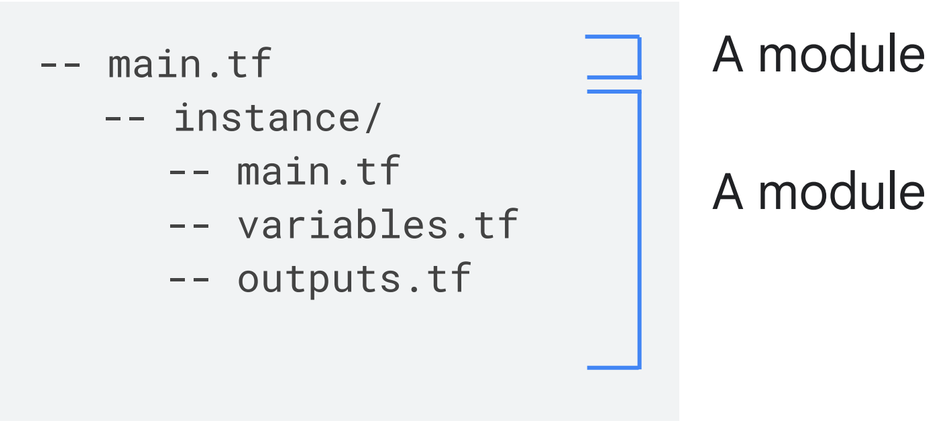
Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	Use variables to parameterize a module
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

A module is collection of configuration files

- One or more Terraform configuration files (.tf) in a directory can form a module.
- Modules let you group a set of resources together and reuse them later.
- The root module consists of the .tf files that are stored in your working directory where you run `terraform plan` or `terraform apply`.

Note: The root module is where other modules and resources are instantiated.



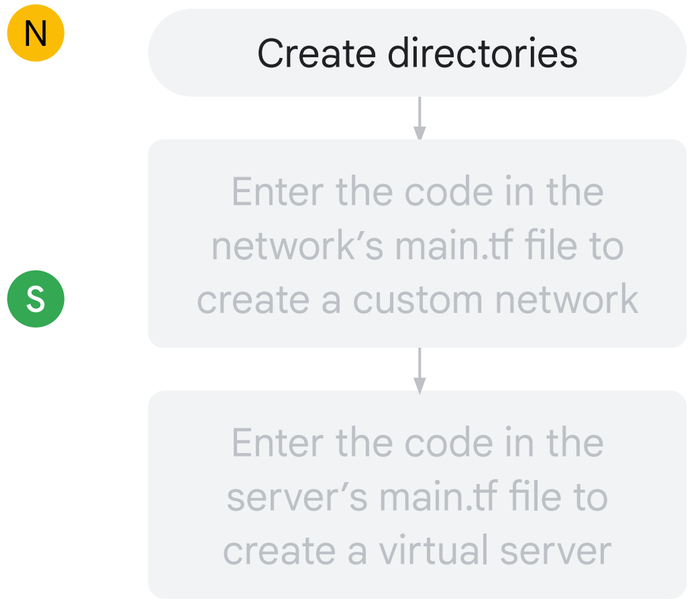
Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	Use variables to parameterize a module
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

Write your first module

```
-- network/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf
```



N

The network module contains networking resources to host the server.

S

The server module contains necessary compute resources that define a server in our organization.

Write your first module

```
-- network/
   -- main.tf
   -- outputs.tf
   -- variables.tf

-- server/
   -- main.tf
   -- outputs.tf
   -- variables.tf
```

N M

Create directories

Enter the code in the network's main.tf file to create a custom network

Enter the code in the server's main.tf file to create a virtual server

N M

```
resource "google_compute_network" "mynetwork" {
  name                    = “mynetwork”
  auto_create_subnetworks  = true
  routing_mode             = global
  mtu                     = 1460
}

resource "google_compute_firewall" "default" {
#All necessary parameters defined
}
```

Notice that multiple resources are grouped in the network module.

Write your first module

```
-- network/
   -- main.tf
   -- outputs.tf
   -- variables.tf

-- server/
   -- main.tf
   -- outputs.tf
   -- variables.tf
```

NM

SM

Create directories

Enter the code in the network's main.tf file to create a custom network

Enter the code in the server's main.tf file to create a virtual server

NM

```
resource "google_compute_network" "dev_network" {
  name                    = “mynetwork”
  auto_create_subnetworks  = true
  routing_mode             = global
  mtu                     = 1460
}
resource "google_compute_firewall" "default" {
#All necessary parameters defined
}
```

SM

```
resource "google_compute_instance" "server_VM" {
#All necessary parameters defined
}
```

Write your first module

```
-- network/
   -- main.tf
   -- outputs.tf
   -- variables.tf

-- server/
   -- main.tf
   -- outputs.tf
   -- variables.tf
```

NM

SM

Create directories

Enter the code in the network's main.tf file to create a custom network

Enter the code in the server's main.tf file to create a virtual server

We created two modules!

NM

```
resource "google_compute_network" "dev_network" {
  name                    = “mynetwork”
  auto_create_subnetworks  = true
  routing_mode             = global
  mtu                     = 1460
}
resource "google_compute_firewall" "default" {
#All necessary parameters defined
}
```

SM

```
resource "google_compute_instance" "server_VM" {
#All necessary parameters defined
}
```

Modules Use Cases

Modularize code

When you want to organize your Terraform configuration in modules so that it's readable and manageable.

Eliminate repetition

When a fair bit of code is repeated multiple times.

Standardize resources

When a set of resources has to be created in a specific way.

Benefits of using modules

Readable

Modules eliminate many lines of code with a call to the source module.

Reusable

You can use modules to write a code once and reuse it multiple times.

Abstract

By using modules, you can separate configurations into logical units.

Consistent

Modules help you package the configuration of a set of resources.

Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	**Reuse configurations by using modules**
05	Use variables to parameterize a module
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

Calling the module to reuse the configuration

```
-- network/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- main.tf
```

N

S

M

Root `main.tf`

```
provider "google" {
region = us-central-1
}

module "web_server" {
source = "./server"
}

module "server_network" {
source = "./network"
}
```

S

N

M

You can use the module by calling it in your main configuration.

`source` meta argument

- `source` is a meta argument, whose value provides the path to the configuration code.
- The value can be a local or remote path.
- There are several supported remote source types, such as Terraform Registry, GitHub, Bitbucket, HTTP URLs, and Cloud Storage buckets.

Syntax for calling the module

```
module "<NAME>" {
 source = "<source_location>"
 [CONFIG ...]
}
```

Module source: Local path

```
-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- main.tf
```

S

M

Source: Local path

```
module "web_server" {
  source = "./server"
}
```

M

S

Module source: Terraform Registry

```
-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- main.tf
```

S

M

Local Source: Local path

```
module "web_server" {
  source = "./server"
}
```

S

M

Remote Source: Terraform Registry

```
module "web_server" {
    source = "terraform-google-modules/vm/google//modules/compute_instance"
}
```

Terraform

Terraform Registry: Version Constraint

```
-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- main.tf
```

S

M

Source: Local path

```
module VMserver {
  source = "./server"
}
```

S

M

Remote Source: Terraform Registry

```
module "web_server" {
    source = "terraform-google-modules/vm/google//modules/compute_instance"
    version = "0.0.5"
}
```

Terraform

Module source: GitHub

```
-- server/
    -- main.tf
    -- outputs.tf
    -- variables.tf

-- main.tf
```

S

M

Source: Local path

```
module VMserver {
  source = "./server"
}
```

S

M

Remote Source: Terraform Registry

```
module "web_server" {
    source = "terraform-google-modules/vm/google//modules/compute_instance"
    version = "0.0.5"
}
```

Terraform

Remote source: GitHub

```
module VMserver {
    source = "github.com/terraform-google-modules/terraform-google-vm//modules/compute_instance"
}
```

GitHub

Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	**Use variables to parameterize a module**
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

Eliminate hard coding by using variables

```
-- main.tf
-- network/
    -- main.tf
    -- outputs.tf
    -- variables.tf
```

M

N

```
..
module "dev_network" {
    source = "./network"
}

module "prod_network" {
    source = "./network"
}
```

M

N

N

Source **main.tf**

```
Error: Error creating Network:
Error 409: The resource
'projects/<project-id>/global/networks/mynetwork' already
exists.
```

Name conflict errors

```
resource "google_compute_network" "vpc_network"
{

name    =  "my-network"
…
}
```

N

Parameterize your configuration with input variables

```
-- network/
   -- main.tf
   -- outputs.tf
   -- variables.tf

-- main.tf
```

N M

Replace the hard coded arguments with a variable.

Declare the variables in the variables.tf file

Pass the value for the input variable when you call the module.

N M

```
resource "google_compute_network" "vpc_network"
{
  name     = var.network_name
..
}
```

Parameterize your configuration with input variables

```
-- network/
   -- main.tf
   -- outputs.tf
   -- variables.tf

-- main.tf
```

N V

Replace the hard coded arguments with a variable.

↓

Declare the variables in the variables.tf file

↓

Pass the value for the input variable when you call the module.

N V

```
variable "network_name" {
type          = string
description   = “name of the network”
}
```

Parameterize your configuration with input variables

```
-- network/
   -- main.tf
   -- outputs.tf
   -- variables.tf

-- main.tf
```

M

Replace the hard coded arguments with a variable.

↓

Declare the variables in the variables.tf file

↓

Pass the value for the input variable when you call the module.

M

```
..
module "dev_network" {
    source            = "./network"
    network_name      = "my-network1"
}

module "prod_network" {
    source            = "./network"
    network_name      = "my-network2"
}
```

You cannot pass values to variables for modules at run time.

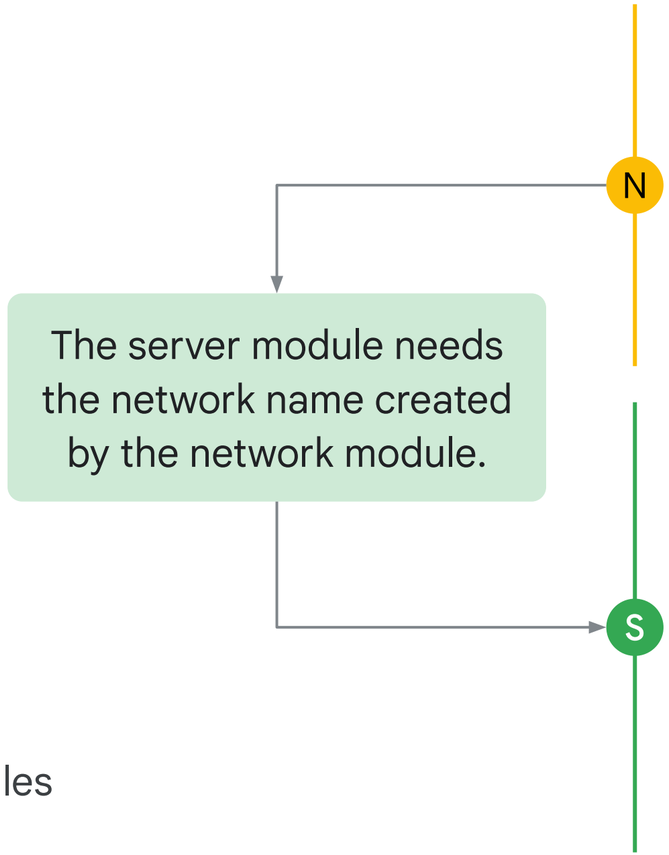
Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	Use variables to parameterize a module
06	**Pass resource attribute outside the module by using output values**
07	Modules best practices
08	A real time scenario

Pass resource attributes outside the module

```
-- main.tf
   -- network/
      -- main.tf
      -- outputs.tf
      -- variables.tf

   -- server/
      -- main.tf
      -- outputs.tf
      -- variables.tf
```




```
resource "google_compute_network" "my_network"
{
  name                          = “mynetwork”
  auto_create_subnetworks       = true
  routing_mode                  = “GLOBAL”
  mtu                           = 1460
}
```

```
resource "google_compute_instance" "server_VM"
{
network = <network created by network module>
#All necessary parameters defined
}

resource "google_compute_firewall" "default" {
#All necessary parameters defined
}
```

Note: Use output values to pass resources attributes between modules

Using output values

```
-- main.tf
   -- network/
      -- main.tf
      -- outputs.tf
      -- variables.tf

   -- server/
      -- main.tf
      -- outputs.tf
      -- variables.tf
```

M

N

S

Declare the output value in the network module.

Declare the argument as a variable in the server module.

Refer the output value when calling the server module.

N

```
# /network/output.tf
output "network_name" {
  value = google_compute_network.my_network.name
}
```

Using output values

```
-- main.tf
    -- network/
        -- main.tf
        -- outputs.tf
        -- variables.tf

    -- server/
        -- main.tf
        -- outputs.tf
        -- variables.tf
```

M

N

S

Declare the output value in the network module.

Declare the argument as a variable in the server module.

Refer the output value when calling the server module.

S

```
# /server/main.tf
resource "google_compute_instance" "server_VM"
{
..
    network = var.network_name
}
# /server/variables.tf
variable "network_name" {
}
```

Using output values

```
-- main.tf
   -- network/
      -- main.tf
      -- outputs.tf
      -- variables.tf

   -- server/
      -- main.tf
      -- outputs.tf
      -- variables.tf
```

M

N

S

Declare the output value in the network module.

Declare the argument as a variable in the server module.

Refer the output value when calling the server module.

M

```
## Root Config
# main.tf
…
module "server_VM1" {
  source = "./server"
  network_name = module.my_network_1.network_name
}
module "my_network_1" {
source = "./network"
}
```

Using output values

```
-- main.tf
    -- network/
        -- main.tf
        -- outputs.tf
        -- variables.tf

    -- server/
        -- main.tf
        -- outputs.tf
        -- variables.tf
```

M

N

S

Declare the output value in the network module.

Declare the argument as a variable in the server module.

Refer the output value when calling the server module.

Run **`terraform init`**

M

```
## Root Config
# main.tf
…
module "server_VM1" {
  source = "./server"
  network_name = module.my_network_1.network_name
}
module "my_network_1" {
source = "./network"
}
```

Topics

01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	Use variables to parameterize a module
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

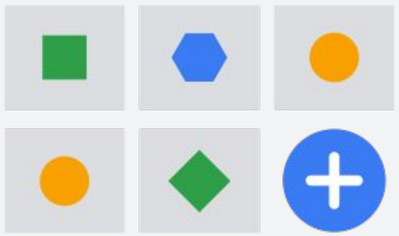
Best Practices

01	Modularize your code for keeping your codebase DRY and encapsulating best practices.
02	Parameterize modules intelligently only if they make sense for end users to change.
03	Use local modules to organize and encapsulate your code.
04	Use the public Terraform Registry for complementing complex architecture confidently.
05	Publish and share your module with your team.

Topics

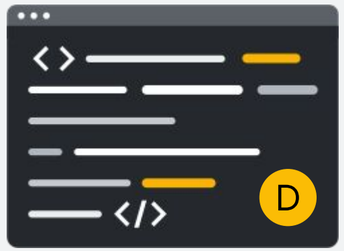
01	Need for modules
02	Modules overview
03	Example of a module, use cases and benefits
04	Reuse configurations by using modules
05	Use variables to parameterize a module
06	Pass resource attribute outside the module by using output values
07	Modules best practices
08	A real time scenario

Managing complex infrastructure in Terraform



Feature test

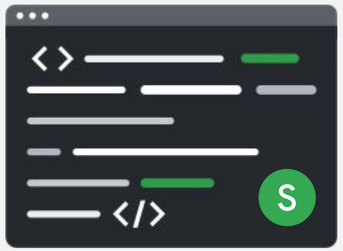
New resources
are added



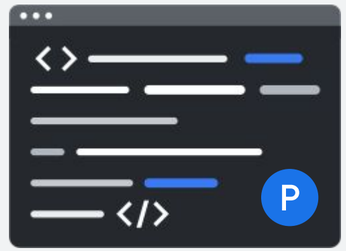
Development

Feature is
tested in dev.

Code is
approved



Staging



Production

The code is manually copied
across staging and production.

Define all resources that belong to the server in a .tf file

```
-- servers/
   -- main.tf
-- environments/
   -- development/
      -- main.tf
   -- production/
      -- main.tf
   -- staging/
      -- main.tf
```

M

M

```
resource "google_compute_instance" "server" {
#All necessary parameters defined
}
resource “google_compute_address” “static_ip”{
#All necessary parameters defined
}
```

Reuse configuration with modules

```
-- servers/
   -- main.tf
-- environments/
   -- development/
      -- main.tf
   -- staging/
      -- main.tf
   -- production/
      -- main.tf
```

M

```
resource "google_compute_instance" "serverVM" {
#All necessary parameters defined
}
resource “google_compute_address” “static_ip”{
#All necessary parameters defined
}
```

D

```
module "dev-server" {
 source = "../servers/main.tf"
 name = “dev_server”
 num_vm = 2
}
```

S

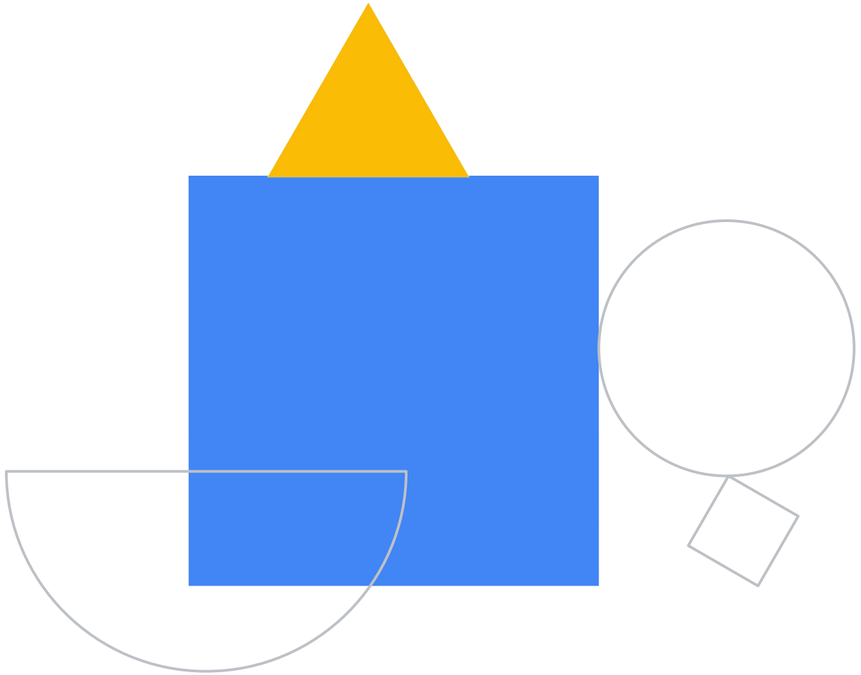
```
module "prod-server" {
 source = "../servers/main.tf"
 name = “prod_server”
 num_vm = 4
}
```

P

```
module "stag-server" {
 source = "../servers/main.tf"
 name = “stag_server”
 num_vm = 6
}
```

Lab

Automating the Deployment of Infrastructure using Terraform



Quiz

Quiz | Question 1

Question

What is the purpose of output values within modules?

A. Pass resource attributes outside a module

B. Parameterize a configuration

C. Ensure the syntax is in canonical format

D. Initialize Terraform to download the plugins.

Quiz | Question 1

Answer

What is the purpose of output values within modules?

A. Pass resource attributes outside a module 

B. Parameterize a configuration

C. Ensure the syntax is in canonical format

D. Initialize Terraform to download the plugins.

Quiz | Question 2

Question

Which code construct of Terraform helps you parameterize a configuration

A. Variables

B. Modules

C. Output values

D. Resources

Quiz | Question 2

Answer

Which code construct of Terraform helps you parameterize a configuration

A. Output values

B. Modules

C. Variables 

D. Resources

Quiz | Question 3

Question

State true or false.
The source of a module can only be remote.

A. True

B. False

Quiz | Question 3

Answer

State true or false.
The source of a module can only be remote.

A. True

B. False 

Quiz | Question 4

Question

What happens when a version argument is specified in a module block?

A. Terraform automatically downgrades the modules to the specific version.

B. Terraform automatically upgrades the modules to the specific version.

C. Terraform automatically upgrades the module to the latest version matching the specified version constraint.

D. Terraform automatically downgrades the module to the oldest version.

Quiz | Question 4

Answer

What happens when a version argument is specified in a module block?

A. Terraform automatically downgrades the modules to the specific version.

B. Terraform automatically upgrades the modules to the specific version.

C. Terraform automatically upgrades the module to the latest version matching the specified version constraint.

D. Terraform automatically downgrades the module to the oldest version.

Module Review

01 Define modules.

02 Use modules to reuse configurations.

03 Use modules from the public registry.

04 Use input variables to parameterize configurations.

05 Use output values to access resource attributes outside the module.