Informed delivery tERRAFORM user guide

Table of Contents

[Install Terraform 2](#_Toc158728095)

[GCP Terraform Registry Documentation 2](#_Toc158728096)

[Naming conventions 3](#_Toc158728097)

[Variables 3](#_Toc158728098)

[Outputs 5](#_Toc158728099)

[Formatting 6](#_Toc158728100)

[Validation 6](#_Toc158728101)

[Terraform root module 6](#_Toc158728102)

[Helper Scripts 6](#_Toc158728103)

# Install Terraform

To install Terraform, find the appropriate package for your system and download it as a zip archive.

Link to Terraform Downloads: [Install | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/install)

After downloading Terraform, unzip the package. Terraform runs as a single binary named terraform. Any other files in the package can be safely removed and Terraform will still function.

Finally, make sure that the terraform binary is available on your PATH. This process will differ depending on your operating system.

**Windows**

1. Go to Control Panel -> System -> System settings -> Environment Variables.
2. Scroll down in system variables until you find PATH.
3. Click edit and change accordingly.
4. BE SURE to include a semicolon at the end of the previous as that is the delimiter, i.e. c:\path;c:\path2
5. Launch a new console for the settings to take effect.

# GCP Terraform Registry Documentation

Use the link below to obtain information on how to configure GCP Modules using terraform.

[Docs overview | hashicorp/google | Terraform | Terraform Registry](https://registry.terraform.io/providers/hashicorp/google/latest/docs)

# Code Organization

Informed Delivery uses [START/END] comment blocks to organize provider resources and variables. Resources and variable of same type should be grouped together. Any variable used across multiple modules should be placed in General Sections. All Toggles should be placed in toggle grouping.

**Environment tfvars files and root variables.tf files**

# [START cloud\_function Variables]

rmin\_alerts\_e\_timeout = 60

rmin\_alerts\_e\_secret\_version = "1"

# [END cloud\_function Variables]

# [START subnetwork Variables]

rmin\_dashboard\_serverless\_vpc\_name = "rmin-dev-dashboard-serverless"

rmin\_peripheral\_vpc\_name = "rmin-dev-peripheral"

rmin\_email\_serverless\_vpc\_name = "rmin-dev-email-serverless"

# [END subnetwork Variables]

# [START Toggle Variables]

variable "toggle\_4583\_1111\_subscriber\_services" {

type = string

}

# [END Toggle Variables]

# [START General Variables]

variable "project\_id" {

type = string

}

# [END General Variables]

# [START vpc Variables]

variable "rmin\_e\_vpc\_name" {

type = string

}

# [END vpc Variables]

**Module main.tf files**

# [START google\_pubsub\_schema]

resource "google\_pubsub\_schema" "dashboard\_api\_logs" {

name = "dashboard-api-logs"

type = "PROTOCOL\_BUFFER"

definition = file("${path.module}/schemas/dashboard-api-logging-schema.proto")

}

# [END google\_pubsub\_schema]

# Naming conventions

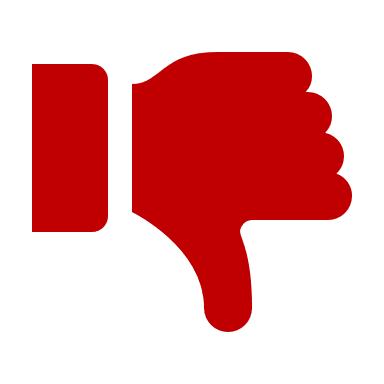
Name all configuration objects should use underscores to delimit multiple words. This practice ensures consistency with the naming convention for resource types, data source types, and other predefined values.

**Recommended:**

resource "google\_compute\_instance" "**web\_server**" {

name = "web-server"

}

**Not recommended:**

resource "google\_compute\_instance" "**web-server**" {

name = "web-server"

}

* Resources names should strive to be the name of the GCP infrastructure object being created (minus an environment specific values)

resource "google\_pubsub\_topic" "**rmin\_campaign\_topic\_e**" {

name = "rmin-campaign-topic-e-${var.name\_suffix}"

project = var.project\_id

}

# Variables

* Declare all variables in variables.tf.
* Give variables descriptive names that are relevant to their usage or purpose:
  + Inputs, local variables, and outputs representing numeric values—such as disk sizes or RAM size—*must* be named with units (such as ram\_size\_gb). Google Cloud APIs don't have standard units, so naming variables with units makes the expected input unit clear for configuration maintainers.
* Variables must have descriptions. Descriptions are automatically included in a published module's auto-generated documentation. Descriptions add additional context for new developers that descriptive names cannot provide.
* A useful naming pattern to follow is “{name}\_{arugment}”, where “name” represents a resource or data source name, and “argument” is the name of field being configured.

resource "google\_compute\_subnetwork" "rmin\_email\_serverless" {

description = "rmin-email-serverless"

ip\_cidr\_range = **var.rmin\_email\_serverless\_ip\_cidr\_range**

log\_config {

aggregation\_interval = "INTERVAL\_5\_SEC"

flow\_sampling = "0.5"

metadata = "INCLUDE\_ALL\_METADATA"

}

…

* Give variables defined types.
* When appropriate, provide default values:
  + For variables that have environment-independent values (such as disk size), provide default values.
  + For variables that have environment-specific values (such as project\_id), don't provide default values. This way, the calling module must provide meaningful values.
* Use empty defaults for variables (like empty strings or lists) only when leaving the variable empty is a valid preference that the underlying APIs don't reject.
* Be judicious in your use of variables. Only parameterize values that must vary for each instance or environment. When deciding whether to expose a variable, ensure that you have a concrete use case for changing that variable. If there's only a small chance that a variable might be needed, don't expose it.
  + Adding a variable with a default value is backwards-compatible.
  + Removing a variable is backwards-incompatible.
  + In cases where a literal is reused in multiple places, you can use a [local value](https://www.terraform.io/docs/configuration/locals.html) without exposing it as a variable.

# Outputs

* Organize all outputs in an outputs.tf file.
* Provide meaningful descriptions for all outputs.
* Output all useful values that root modules might need to refer to or share. Especially for open source or heavily used modules, expose all outputs that have potential for consumption.
* A useful naming pattern to follow is “{name}\_{attribute}”, where “name” represents a resource or data source name, and “attribute” is the attribute returned by the output.

output "**rmin\_email\_topic\_e\_topic\_id**" {

value = google\_pubsub\_topic.rmin\_email\_topic\_e.id

description = "Name of Topic with format projects/{{project}}/topics/{{name}"

}

* Don't pass outputs directly through input variables, because doing so prevents them from being properly added to the dependency graph.

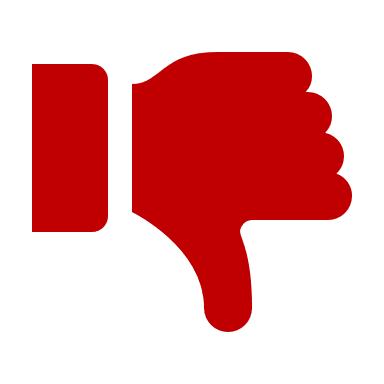
**Recommended:**

output "name" {

description = "Name of instance"

value = **google\_compute\_instance.main.name**

}

**Not recommended:**

output "name" {

description = "Name of instance"

value = **var.name**

}

# Toggles

## **Release Toggle Documentation**

All Toggles will be documented in the terraform\_toggles.md file. File will be stored in the rmin-cloud-terraform-scripts Github Repo under the *docs* folder. Toggles will be documented as followed.

Release Name: R23.8.0 ID - Mail Campaign Reporting to Cloud

Release Date: 1/10/2024

TSLC-ID: 4583-5628

Toggle Name: toggle\_4583\_5628\_mcr

## **Naming convention**

Toggles names will begin with “toggle\_” prefix, followed by the release TSLC ID. Underscores will be the only special characters used in toggle names.

e.g. toggle\_4583\_5628

# Formatting

All Terraform files must conform to the standards of terraform fmt. Run the following commands to on your local before committing to terraform built in formatting.

**Usage**

# Validation

terraform validate can be used as a process dry-run and will check for any glaring flaws before a Terraform run. For example, If you have a variable that is currently unavailable or incorrectly referenced, Terraform validate will catch that error. Run the following commands on your local to check that your code changes will not break the build.

**Usage**

terraform init -backend=false

terraform validate

# Terraform root module

Root configurations (root modules) are the working directories from which you run the Terraform CLI. The root directory of rmin-cloud-terraform-scripts Github Repo is the main working directory for terraform.

## Helper Scripts

The helper directory of the rmin-cloud-terraform-scripts container bash and python scripts to help manage the execution of the terraform command.

#### execute\_plan.sh

This shell script will be used to Preview changes before applying to GCP environment. Script will generate a log file to allow application and service providers to review changes. Additionally, it will save the plan to a file called tfplan, which will be used in execute\_apply.sh

##### execute\_apply.sh

This shell script is used after running execute\_plan.sh script. Using the tfplan file from execute\_plan.sh, it will apply all configuration changes to GCP environment.

##### update\_cloudrun\_partman\_\*.sh

ID Terraform implementation does not update Cloud Run instance for Gen 2 Partman Functions with the Cloud SQL Instance. This script will add Cloud SQL Instance to the Cloud Run. Script is run after running execute\_apply.sh script.

##### Update\_config\_value.sh

Shell script to change environment tfvar file config values.

**Example**

python3 -m pip install -r requirements.txt

python3 update\_config\_values.py -p uspscio-it-idc-9060-01c -o rmin\_flats\_lock\_cache\_c\_replica\_enabled=true --execute