Community Tutorials

COMMUNITY HOME SEARCH TUTORIALS

EDIT ON GITHUB REPORT ISSUE PAGE HISTORY

Getting started with Terraform on Google Cloud

Author(s): @chrisst , Published: 2018-08-17

Google Cloud Community tutorials submitted from the community do not represent official Google Cloud product documentation.

One of the things I find most time consuming when starting on a new stack or technology is moving from reading documentation to a working prototype serving HTTP requests. This can be especially frustrating when trying to tweak configurations and keys, as it can be hard to make incremental progress. However, once I have a shell of a web service stood up, I can add features, connect to other APIs, or add a datastore. I'm able to iterate very quickly with feedback at each step of the process. To help get through those first set up steps I've written this tutorial to cover the following:

- Using Terraform to create a VM in Google Cloud
- Starting a basic Python Flask server

Before you begin

PDFmyURL

You will be starting a single Compute Engine VM instance, which can incur real, although usually minimal, costs. Pay attention to the pricing on the account. If you don't already have a Google Cloud account, you can sign up for a free trial and get $300 of free credit, which is more than you'll need for this tutorial.

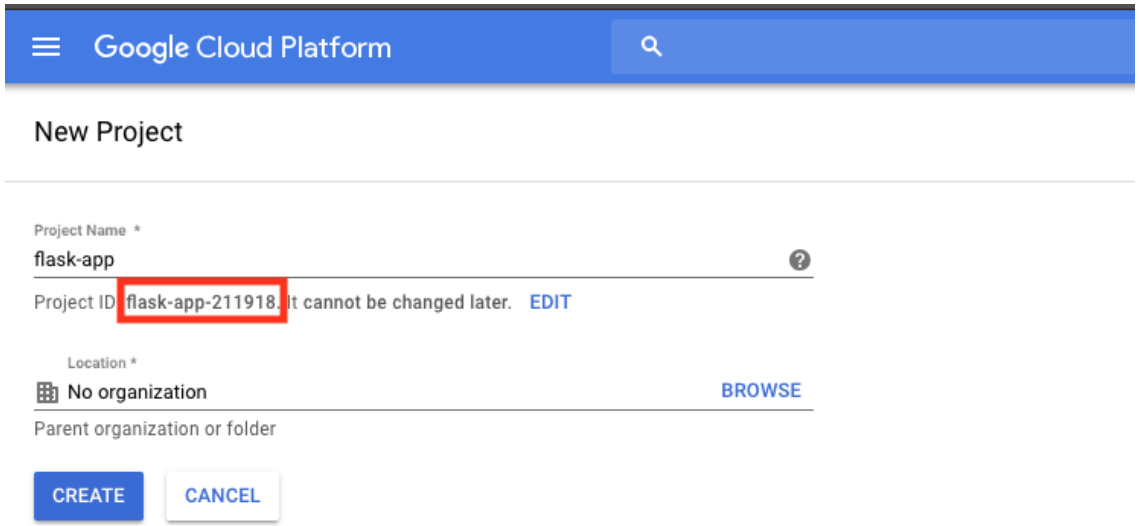
Have the following tools locally:

- An existing SSH key
- Terraform

This tutorial is written using Terraform 0.12 syntax. If you're using a different version of Terraform, some of the syntax will be slightly different.

Create a Google Cloud project

A default project is often set up by default for new accounts, but you will start by creating a new project to keep this separate and easy to tear down later. After creating it, be sure to copy down the project ID as it is usually different then the project name.



Getting project credentials

Next, set up a service account key, which Terraform will use to create and manage resources in your Google Cloud project. Go to the create service account key page. Select the default service account or create a new one, select JSON as the key type, and click **Create**.

This downloads a JSON file with all the credentials that will be needed for Terraform to manage the resources. This file should be located in a secure place for production projects, but for this example move the downloaded JSON file to the project directory.

Setting up Terraform

Create a new directory for the project to live and create a `main.tf` file for the Terraform config. The contents of this file describe all of the Google Cloud resources that will be used in the project.

```
// Configure the Google Cloud provider
provider "google" {
 credentials = file("CREDENTIALS_FILE.json")
 project     = "flask-app-211918"
 region      = "us-west1"
}
```

Set the project ID from the first step to the `project` property and point the credentials section to the file that was downloaded in the last step. The `provider “google”` line indicates that you are using the Google Cloud Terraform provider and at this point you can run `terraform init` to download the latest version of the provider and build the `.terraform` directory.

```
terraform init

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "google" (1.16.2)...

The following providers do not have any version constraints in configuration,
so the latest version was installed.
```

```
To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.google: version = "~> 1.16"

Terraform has been successfully initialized!
```

Configure the Compute Engine resource

Next you will create a single Compute Engine instance running Debian. For this demo you can use the smallest instance possible (check out all machine types here) but you can upgrade to a larger instance later. Add the `google_compute_instance` resource to the `main.tf`:

```
// Terraform plugin for creating random ids
resource "random_id" "instance_id" {
 byte_length = 8
}

// A single Compute Engine instance
resource "google_compute_instance" "default" {
 name         = "flask-vm-${random_id.instance_id.hex}"
 machine_type = "f1-micro"
 zone         = "us-west1-a"

 boot_disk {
   initialize_params {
     image = "debian-cloud/debian-9"
   }
 }

// Make sure flask is installed on all new instances for later steps
 metadata_startup_script = "sudo apt-get update; sudo apt-get install -yq build-essential python-pip r

 network_interface {
   network = "default"

   access_config {
```

```
      // Include this section to give the VM an external ip address
    }
  }
}
```

The `random_id` Terraform plugin allows you to create a somewhat random instance name that still complies with the Google Cloud instance naming requirements but requires an additional plugin. To download and install the extra plugin, run `terraform init` again.

Validate the new Compute Engine instance

You can now validate the work that has been done so far. Run `terraform plan` which will:

- Verify the syntax of `main.tf` is correct
- Ensure the credentials file exists (contents will not be verified until `terraform apply`)
- Show a preview of what will be created

Output:

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + google_compute_instance.default
      id:                                                   [computed]
...

  + random_id.instance_id
      id:                                                   [computed]
...


Plan: 2 to add, 0 to change, 0 to destroy.
```

Now it's time to run `terraform apply` and Terraform will call Google Cloud APIs to set up the new instance. Check the VM Instances page, and the new instance will be there.

Running a server on Google Cloud

There is now a new instance running in Google Cloud, so your next steps are getting a web application created, deploying it to the instance, and exposing an endpoint for consumption.

Add SSH access to the Compute Engine instance

You will need to add a public SSH key to the Compute Engine instance to access and manage it. Add the local location of your public key to the `google_compute_instance` metadata in `main.tf` to add your SSH key to the instance. More information on managing ssh keys is available here.

```
resource "google_compute_instance" "default" {
 ...
metadata = {
   ssh-keys = "INSERT_USERNAME:${file("~/.ssh/id_rsa.pub")}"
 }
}
```

Be sure to replace `INSERT_USERNAME` with your username and then run `terraform plan` and verify the output looks correct. If it does, run `terraform apply` to apply the changes.

The output shows that it will modify the existing compute instance:

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  ~ google_compute_instance.default
      metadata.%:       "0" => "1"
```

```
…

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

Use output variables for the IP address

Use a Terraform output variable to act as a helper to expose the instance's ip address. Add the following to the Terraform config:

```
// A variable for extracting the external IP address of the instance
output "ip" {
 value = google_compute_instance.default.network_interface.0.access_config.0.nat_ip
}
```

Run `terraform apply` followed by `terraform output ip` to return the instance's external IP address. Validate that everything is set up correctly at this point by connecting to that IP address with SSH.

This tutorial needs the `default` network's `default-allow-ssh` firewall rule to be in place before you can use SSH to connect to the instance. If you are starting with a new project, this can take a few minutes. You can check the firewall rules list to make sure that the firewall rule has been created.

```
ssh `terraform output ip`
```

Building the Flask app

You will be building a Python Flask app for this tutorial so that you can have a single file describing your web server and test endpoints. Inside the VM instance, add the following to a new file called `app.py`:

```
from flask import Flask
app = Flask(__name__)
```

```
@app.route('/')
def hello_cloud():
   return 'Hello Cloud!'

app.run(host='0.0.0.0')
```

Then run this command:

```
python app.py
```

Flask serves traffic on `localhost:5000` by default. Run `curl` in a separate SSH instance to confirm that your greeting is being returned. To connect to this from your local computer, you must expose port 5000.

Run this command to validate the server:

```
curl http://0.0.0.0:5000
```

The output from this command is `Hello Cloud`.

Open port 5000 on the instance

Google Cloud allows for opening ports to traffic via firewall policies, which can also be managed in your Terraform configuration. Add the following to the config and proceed to run plan/apply to create the firewall rule.

```
resource "google_compute_firewall" "default" {
 name    = "flask-app-firewall"
 network = "default"

 allow {
   protocol = "tcp"
   ports    = ["5000"]
 }
}
```

Congratulations! You can now point your browser to the instance's IP address and port 5000 and see your server running.

Cleaning up

Now that you are finished with the tutorial, you will likely want to delete everything that was created so that you don't incur any further costs. Thankfully, Terraform will let you remove all the resources defined in the configuration file with `terraform destroy`:

```
terraform destroy
random_id.instance_id: Refreshing state... (ID: ZNS6E3_1miU)
google_compute_firewall.default: Refreshing state... (ID: flask-app-firewall)
google_compute_instance.default: Refreshing state... (ID: flask-vm-64d4ba137ff59a25)

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - google_compute_firewall.default

  - google_compute_instance.default

  - random_id.instance_id
...

google_compute_firewall.default: Destroying... (ID: flask-app-firewall)
google_compute_instance.default: Destroying... (ID: flask-vm-64d4ba137ff59a25)
google_compute_instance.default: Still destroying... (ID: flask-vm-64d4ba137ff59a25, 10s elapsed)
google_compute_firewall.default: Still destroying... (ID: flask-app-firewall, 10s elapsed)
google_compute_firewall.default: Destruction complete after 11s
google_compute_instance.default: Destruction complete after 18s
random_id.instance_id: Destroying... (ID: ZNS6E3_1miU)
random_id.instance_id: Destruction complete after 0s
```

Submit a Tutorial

Share step-by-step guides

SUBMIT A TUTORIAL

Request a Tutorial

Ask for community help

SUBMIT A REQUEST

GCP Tutorials

Tutorials published by GCP

VIEW TUTORIALS

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License, and code samples are licensed under the Apache 2.0 License. For details, see our Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

DevOps solutions
Small business solutions
See all solutions

Google Developers
Google Cloud for Startups
System status
Release Notes

Google Cloud on YouTube
GCP on YouTube
G Suite on YouTube
Follow on Twitter
Join User Research
We're hiring. Join Google Cloud!

About Google | Privacy | Site terms | Google Cloud terms

Sign up for the Google Cloud newsletter Subscribe Language

PDFmyURL.com - convert URLs, web pages or even full websites to PDF online. Easy API for developers!