Terraform Fundamentals in GCP

Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

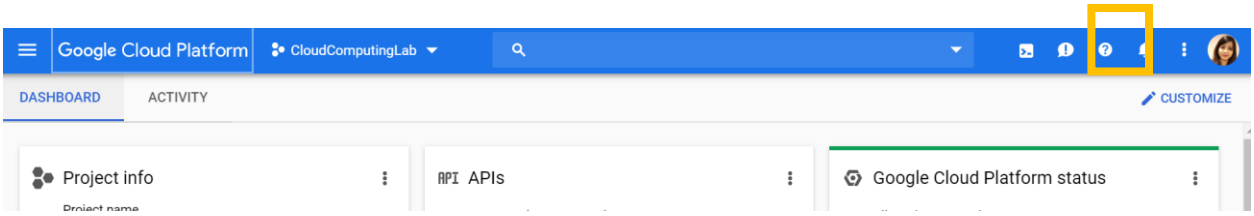
Objectives:

- Getting Started with Terraform in Google Cloud.
- Install Terraform from Installation Binaries.
- Create a VM instance infrastructure using Terraform.

Setup and Requirements:

- Create free account at https://console.cloud.google.com/freetrial
- The GCP Console opens up as shown below on activation

- Activate Google Cloud Shell by clicking on the highlighted icon
  Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Google Cloud Shell provides command-line access to your GCP resources.



- First time, when you activate the cloud shell, it has no Project ID set.
  The gcloud projects group lets you create and manage IAM policies for projects on the Google Cloud Platform. Resources are organized hierarchically and assigned to a particular project. A Project resource is required to use Google Cloud Platform, and forms the basis for creating, enabling and using all Cloud Platform services, managing APIs, enabling billing, adding and removing collaborators, and managing permissions.
  Create a new project in cloud shell

```
swati_jadon0107@cloudshell:~$ gcloud projects create is6641 --name=is6641 --enable-cloud-apis --set-as-default
```

```
swati_jadon0107@cloudshell:~$ gcloud projects create is6641 --name=is6641 --enable-cloud-apis --set-as-default
Create in progress for [https://cloudresourcemanager.googleapis.com/v1/projects/is6641].
Waiting for [operations/cp.8274212247219618776] to finish...done.
Updated property [core/project] to [is6641].


To take a quick anonymous survey, run:
  $ gcloud alpha survey

swati_jadon0107@cloudshell:~ (is6641)$
```

Project is now created and resources can now be assigned to the this project.

- Enable Cloud Compute API . This API Creates and runs virtual machines on Google Cloud Platform. https://console.developers.google.com/apis/library/compute.googleapis.com?supportedpurview=project&project=is6641

You can **list the active account name** with this command:

```
swati_jadon0107@cloudshell:~ (is6641)$ gcloud auth list
```

Output:

```
swati_jadon0107@cloudshell:~ (is6641)$ gcloud auth list
      Credentialed Accounts
ACTIVE  ACCOUNT
*       swati.jadon0107@gmail.com

To set the active account, run:
    $ gcloud config set account `ACCOUNT`
```

You can **list the project ID** with this command:

```
swati_jadon0107@cloudshell:~ (is6641)$ gcloud config list project
```

Output:

```
[core]
project = is6641
```

What is Terraform?

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing, popular service providers as well as custom in-house solutions.

Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.

The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

Key Features

Infrastructure as Code

Infrastructure is described using a high-level configuration syntax. This allows a blueprint of your datacenter to be versioned and treated as you would any other code. Additionally, infrastructure can be shared and re-used.

Execution Plans

Terraform has a "planning" step where it generates an execution plan. The execution plan shows what Terraform will do when you call apply. This lets you avoid any surprises when Terraform manipulates infrastructure.

Resource Graph

Terraform builds a graph of all your resources, and parallelizes the creation and modification of any non-dependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure.

Change Automation

Complex changesets can be applied to your infrastructure with minimal human interaction. With the previously mentioned execution plan and resource graph, you know exactly what Terraform will change and in what order, avoiding many possible human errors.

Install Terraform

Configure your Cloud Shell environment to use the Terraform by installing it with the appropriate package:

```
Your active configuration is: [cloudshell-19607]
swati_jadon0107@cloudshell:~ (is6641)$ wget https://releases.hashicorp.com/terraform/0.11.9/terraform_0.11.9_linux_amd64.zip
--2019-05-14 22:13:37--  https://releases.hashicorp.com/terraform/0.11.9/terraform_0.11.9_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 151.101.1.183, 151.101.65.183, 151.101.129.183, ...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|151.101.1.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20941303 (20M) [application/zip]
Saving to: 'terraform_0.11.9_linux_amd64.zip.2'

terraform_0.11.9_linux_amd64.zip.2      100%[=======================================================================>]  19.97M  35.9MB/s    in 0.6s

2019-05-14 22:13:38 (35.9 MB/s) - 'terraform_0.11.9_linux_amd64.zip.2' saved [20941303/20941303]
```

Unzip the downloaded package:

```
swati_jadon0107@cloudshell:~ (is6641)$ unzip terraform_0.11.9_linux_amd64.zip
Archive:  terraform_0.11.9_linux_amd64.zip
replace terraform? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: terraform
```

Set PATH environmental variable to Terraform binaries:

```
swati_jadon0107@cloudshell:~ (is6641)$ export PATH="$PATH:$HOME/terraform"
swati_jadon0107@cloudshell:~ (is6641)$ cd /usr/bin
swati_jadon0107@cloudshell:/usr/bin (is6641)$ sudo ln -s $HOME/terraform
ln: failed to create symbolic link './terraform': File exists
swati_jadon0107@cloudshell:/usr/bin (is6641)$ cd $HOME
swati_jadon0107@cloudshell:~ (is6641)$ source ~/.bashrc
```

Note: Terraform is distributed as a binary package for all supported platforms and architectures.

Verifying the Installation

After installing Terraform, verify the installation by checking that Terraform is available:

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ terraform
```

Output:

```
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the configuration
    graph              Create a visual graph of Terraform resources
    import             Import existing infrastructure into Terraform
    init               Initialize a Terraform working directory
    output             Read an output from a state file
    plan               Generate and show an execution plan
    providers          Prints a tree of the providers used in the configuration
    push               Upload this Terraform module to Atlas to run
    refresh            Update local state file against real resources
    show               Inspect Terraform state or plan
    taint              Manually mark a resource for recreation
    untaint            Manually unmark a resource as tainted
    validate           Validates the Terraform files
    version            Prints the Terraform version
    workspace          Workspace management

All other commands:
    debug              Debug output management (experimental)
    force-unlock       Manually unlock the terraform state
    state              Advanced state management
```

Build Infrastructure

With Terraform installed, you can dive right in and start creating some infrastructure.

Configuration

The set of files used to describe infrastructure in Terraform is simply known as a Terraform configuration. We're going to write our first configuration now to launch a single VM instance.

The format of the configuration files is documented here. We recommend using JSON for creating configuration files.

Create a configuration an instance.tf file with your favorite editor like vim, nano etc.:

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ nano instance.tf
```

Add the following content in file, Make sure to replace <PROJECT_ID>with the GCP project ID:

```
  GNU nano 2.7.4

resource "google_compute_instance" "default" {
  project      = "is6641"
  name         = "terraform"
  machine_type = "n1-standard-1"
  zone         = "us-central1-a"

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

  network_interface {
    network = "default"
    access_config {
    }
  }
}
```

Ctrl+O ->Enter->Ctrl+X to save the file.

This is a complete configuration that Terraform is ready to apply. The general structure should be intuitive and straightforward.

The "resource" block in the instance.tf file defines a resource that exists within the infrastructure. A resource might be a physical component such as an VM instance.

The resource block has two strings before opening the block: the **resource type** and the **resource name**. For this lab the resource type is google_compute_instance and the name is terraform. The prefix of the type maps to the provider: google_compute_instanceautomatically tells Terraform that it is managed by the Googleprovider.

Within the resource block itself is the configuration needed for the resource.

Verify your new file has been added and that there are no other *.tffiles in your directory, since Terraform loads all of them:

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ ls
instance.tf  README-cloudshell.txt  terraform  terraform_0.11.9_linux_amd64.zip
```

Initialization

The first command to run for a new configuration -- or after checking out an existing configuration from version control -- is terraform init. This will initialize various local settings and data that will be used by subsequent commands.

Terraform uses a plugin-based architecture to support the numerous infrastructure and service providers available. Each "Provider" is its own encapsulated binary distributed separately from Terraform itself. The terraform init command will automatically download and install any Provider binary for the providers to use within the configuration, which in this case is just the Google provider.

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ terraform init

Initializing provider plugins...

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 = "~> 2.6"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

The Google provider plugin is downloaded and installed in a subdirectory of the current working directory, along with various other bookkeeping files. You will see an "Initializing provider plugins" message. Terraform knows that you're running from a Google project and is getting Google resources.

The output specifies which version of the plugin is being installed, and suggests specifying this version in future configuration files to ensure that terraform init will install a compatible version.

The terraform plan command is used to create an execution plan. Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files.

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

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>
      boot_disk.#:                                      "1"
      boot_disk.0.auto_delete:                          "true"
      boot_disk.0.device_name:                          <computed>
      boot_disk.0.disk_encryption_key_sha256:           <computed>
      boot_disk.0.initialize_params.#:                  "1"
      boot_disk.0.initialize_params.0.image:            "debian-cloud/debian-9"
      boot_disk.0.initialize_params.0.size:             <computed>
      boot_disk.0.initialize_params.0.type:             <computed>
      can_ip_forward:                                   "false"
      cpu_platform:                                     <computed>
      deletion_protection:                              "false"
      guest_accelerator.#:                              <computed>
      instance_id:                                      <computed>
      label_fingerprint:                                <computed>
      machine_type:                                     "n1-standard-1"
      metadata_fingerprint:                             <computed>
      name:                                             "terraform"
      network_interface.#:                              "1"
      network_interface.0.access_config.#:              "1"
      network_interface.0.access_config.0.assigned_nat_ip: <computed>
      network_interface.0.access_config.0.nat_ip:       <computed>
      network_interface.0.access_config.0.network_tier: <computed>
      network_interface.0.address:                      <computed>
      network_interface.0.name:                         <computed>
      network_interface.0.network:                      "default"
      network_interface.0.network_ip:                   <computed>
      network_interface.0.subnetwork_project:           <computed>
      project:                                          "is6641"
      scheduling.#:                                     <computed>
      self_link:                                        <computed>
      tags_fingerprint:                                 <computed>
      zone:                                             "us-central1-a"


Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state. For example, terraform plan

might be run before committing a change to version control, to create confidence that it will behave as expected.

Note: The optional -out argument can be used to save the generated plan to a file for later execution with terraform apply.

Apply Changes

In the same directory as the instance.tf file you created, run terraform apply.

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ terraform apply

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>
      boot_disk.#:                                       "1"
      boot_disk.0.auto_delete:                           "true"
      boot_disk.0.device_name:                           <computed>
      boot_disk.0.disk_encryption_key_sha256:            <computed>
      boot_disk.0.initialize_params.#:                   "1"
      boot_disk.0.initialize_params.0.image:             "debian-cloud/debian-9"
      boot_disk.0.initialize_params.0.size:              <computed>
      boot_disk.0.initialize_params.0.type:              <computed>
      can_ip_forward:                                    "false"
      cpu_platform:                                      <computed>
      deletion_protection:                               "false"
      guest_accelerator.#:                               <computed>
      instance_id:                                       <computed>
      label_fingerprint:                                 <computed>
      machine_type:                                      "n1-standard-1"
      metadata_fingerprint:                              <computed>
      name:                                              "terraform"
      network_interface.#:                               "1"
      network_interface.0.access_config.#:               "1"
      network_interface.0.access_config.0.assigned_nat_ip: <computed>
      network_interface.0.access_config.0.nat_ip:        <computed>
      network_interface.0.access_config.0.network_tier:  <computed>
      network_interface.0.address:                       <computed>
      network_interface.0.name:                          <computed>
      network_interface.0.network:                       "default"
      network_interface.0.network_ip:                    <computed>
      network_interface.0.subnetwork_project:            <computed>
      project:                                           "is6641"
      scheduling.#:                                      <computed>
      self_link:                                         <computed>
```

```
      tags_fingerprint:                                   <computed>
      zone:                                               "us-central1-a"


Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

google_compute_instance.default: Creating...
  boot_disk.#:                                        "" => "1"
  boot_disk.0.auto_delete:                            "" => "true"
  boot_disk.0.device_name:                            "" => "<computed>"
  boot_disk.0.disk_encryption_key_sha256:             "" => "<computed>"
  boot_disk.0.initialize_params.#:                    "" => "1"
  boot_disk.0.initialize_params.0.image:              "" => "debian-cloud/debian-9"
  boot_disk.0.initialize_params.0.size:               "" => "<computed>"
  boot_disk.0.initialize_params.0.type:               "" => "<computed>"
  can_ip_forward:                                     "" => "false"
  cpu_platform:                                       "" => "<computed>"
  deletion_protection:                                "" => "false"
  guest_accelerator.#:                                "" => "<computed>"
  instance_id:                                        "" => "<computed>"
  label_fingerprint:                                  "" => "<computed>"
  machine_type:                                       "" => "n1-standard-1"
  metadata_fingerprint:                               "" => "<computed>"
  name:                                               "" => "terraform"
  network_interface.#:                                "" => "1"
  network_interface.0.access_config.#:                "" => "1"
  network_interface.0.access_config.0.assigned_nat_ip: "" => "<computed>"
  network_interface.0.access_config.0.nat_ip:         "" => "<computed>"
  network_interface.0.access_config.0.network_tier:   "" => "<computed>"
  network_interface.0.address:                        "" => "<computed>"
  network_interface.0.name:                           "" => "<computed>"
  network_interface.0.network:                        "" => "default"
  network_interface.0.network_ip:                     "" => "<computed>"
  network_interface.0.subnetwork_project:             "" => "<computed>"
  project:                                            "" => "is6641"
  scheduling.#:                                       "" => "<computed>"
  self_link:                                          "" => "<computed>"
  tags_fingerprint:                                   "" => "<computed>"
  zone:                                               "" => "us-central1-a"
google_compute_instance.default: Still creating... (10s elapsed)
google_compute_instance.default: Still creating... (20s elapsed)
google_compute_instance.default: Still creating... (30s elapsed)
google_compute_instance.default: Still creating... (40s elapsed)
google_compute_instance.default: Creation complete after 49s (ID: terraform)

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

If the plan was created successfully, Terraform will now pause and wait for approval before proceeding. In a production environment, if anything in the Execution Plan seems incorrect or dangerous, it's safe to abort here. No changes have been made to your infrastructure.

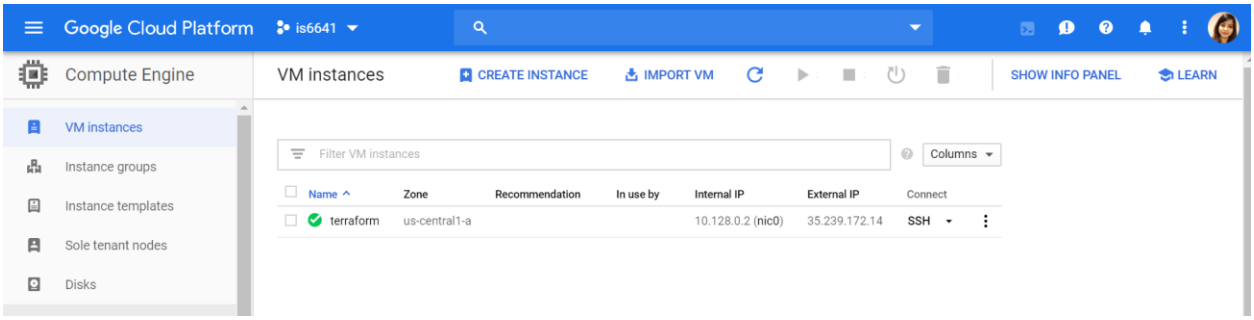
For this case the plan looks acceptable, so type yes at the confirmation prompt to proceed.

Executing the plan will take a few minutes since Terraform waits for the VM instance to become available

After this, Terraform is all done!

Test Setup

In the Console, go to **Compute Engine** > **VM instances** to see the created VM instance.



Terraform has written some data into the terraform.tfstate file. This state file is extremely important; it keeps track of the IDs of created resources so that Terraform knows what it is managing.

You can inspect the current state using terraform show:

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ terraform show
google_compute_instance.default:
  id = terraform
  attached_disk.# = 0
  boot_disk.# = 1
  boot_disk.0.auto_delete = true
  boot_disk.0.device_name = persistent-disk-0
  boot_disk.0.disk_encryption_key_raw =
  boot_disk.0.disk_encryption_key_sha256 =
  boot_disk.0.initialize_params.# = 1
  boot_disk.0.initialize_params.0.image = https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/debian-9-stretch-v20190514
  boot_disk.0.initialize_params.0.size = 10
  boot_disk.0.initialize_params.0.type = pd-standard
  boot_disk.0.source = https://www.googleapis.com/compute/v1/projects/is6641/zones/us-central1-a/disks/terraform
  can_ip_forward = false
  cpu_platform = Intel Haswell
  deletion_protection = false
  guest_accelerator.# = 0
  hostname =
  instance_id = 7312468680199510112
  label_fingerprint = 42WmSpB8rSM=
  labels.% = 0
  machine_type = n1-standard-1
  metadata.% = 0
  metadata_fingerprint = VZjbuej0H_g=
  metadata_startup_script =
  min_cpu_platform =
  name = terraform
  network_interface.# = 1
  network_interface.0.access_config.# = 1
  network_interface.0.access_config.0.assigned_nat_ip =
  network_interface.0.access_config.0.nat_ip = 35.239.172.14
  network_interface.0.access_config.0.network_tier = PREMIUM
  network_interface.0.access_config.0.public_ptr_domain_name =
  network_interface.0.address =
  network_interface.0.alias_ip_range.# = 0
  network_interface.0.name = nic0
  network_interface.0.network = https://www.googleapis.com/compute/v1/projects/is6641/global/networks/default
  network_interface.0.network_ip = 10.128.0.2
  network_interface.0.subnetwork = https://www.googleapis.com/compute/v1/projects/is6641/regions/us-central1/subnetworks/default
  network_interface.0.subnetwork_project = is6641
  project = is6641
  scheduling.# = 1
```

If you want to go review the execution plan after it's been applied, you can use terraform plan command:

```
swati_jadon0107@cs-6000-devshell-vm-6b389325-e88e-4d98-94aa-507b2b2d537e:~$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

google_compute_instance.default: Refreshing state... (ID: terraform)

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Multiple Choice Questions:

Terraform enables you to safely and predictably create, change, and improve infrastructure.

- True

With Terraform we can create our own custom provider plugins.

- True