# Setting Up a Splunk Testing Environment Using Terraform GCP

**By: Darren Fuller**
*Senior Splunk Consultant*
*Discovered Intelligence Inc.*

***https://discoveredintelligence.com***

# Table of Contents

# Part 1: The Splunk Core Hosts

Have you ever wished you had a fresh ephemeral Splunk instance that you could quickly spin up, run some tests and then kill it, with maximum speed and minimum cloud costs?

I was working on a customer engagement recently that required me to test a custom developed Splunk app on a large list of Splunk Universal Forwarder versions and OS distributions (75 distinct host types in total) that if spun up manually on a cloud provider would take a long time to set up and tear down, would be subject to potentially costly "fat-fingering" user errors and would be costly if left up longer than necessary.

Enter Hashi Terraform to the rescue. The industry-leading infrastructure-as-code tool makes the standup, setup and teardown of cloud compute nodes simple, speedy and repeatable so that an environment can be built, a complete set of tests can be run, results received and the test nodes destroyed in minutes rather than hours.

In this whitepaper, I show how I set up my computer and built the Search Head and Deployment server, as well as how I set up the many Splunk Universal Forwarders to satisfy the test plan.

## Setting up your computer

To set up your local host to be able to complete the steps outlined in these instructions you need to install the gcloud and terraform command line tools. These are relatively straightforward and well documented

**Installing GCloud**

The GCloud command line tools are available for Windows, MacOS and Linux and include all the commands needed to create, update and destroy Google Cloud instances from your local machine.

Instructions on how to setup the gcloud cli on your preferred host OS can be found here:
https://cloud.google.com/sdk/docs/install

*Note: To use the gcloud CLI tools you need to have a Google Cloud Platform account. A free trial with $300 credit can be utilized by new GCP customers by visiting: https://cloud.google.com/free*

### Installing Terraform

Similar to the GCloud tools, Terraform is available for all major platforms. To install Terraform, follow the installation instructions on the Hashicorp Terraform website at : https://developer.hashicorp.com/terraform/downloads

The non-cloud version of terraform is free to download and use.

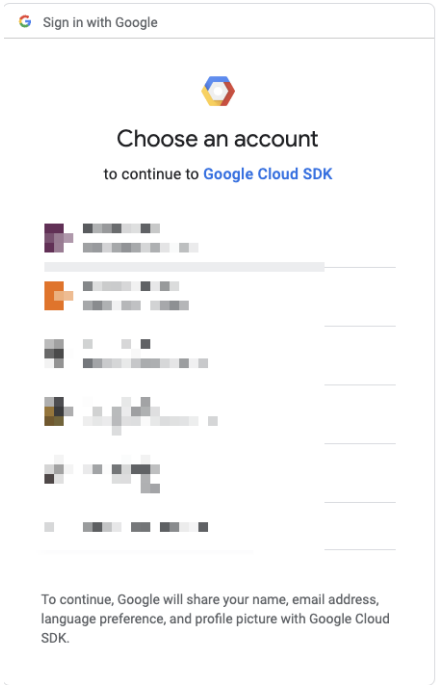
## Logging into GCP using gcloud CLI

Now that we have all the tools setup, let's log in to GCP on your host and test the GCP command line tools.

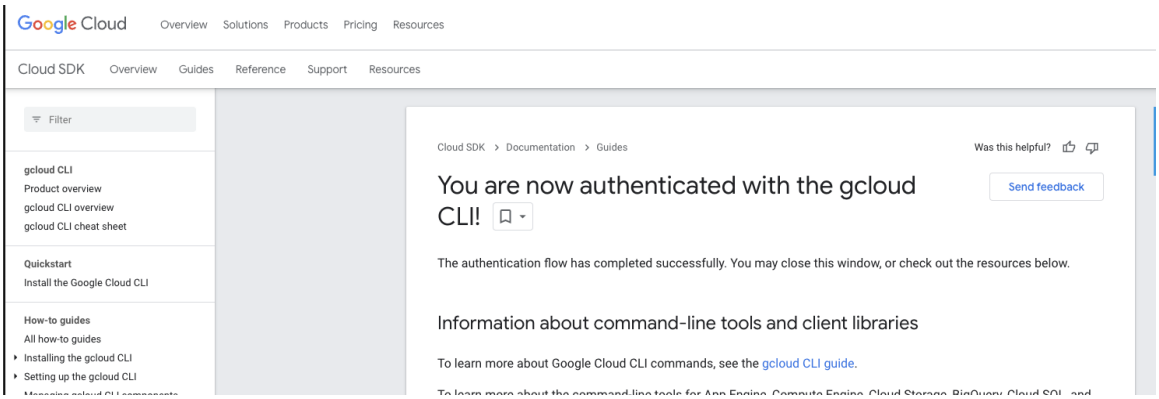
Open up the terminal and run the following command:

```
gcloud auth login
```

This will open your browser to a google authentication window. Select the Google user that you set up (or had set up for you) for GCP



Then you will be shown a permissions request saying you allow the gcloud SDK to manage your instances on the platform. Click 'Allow' to continue. This should open a "You are now authenticated" page in the browser




And should show a similar logged in message on the terminal


## Setting up a GCP Project

Let's talk about projects in GCP for a second. In the context of Google Cloud Platform (GCP), "projects" are the fundamental organizational unit used to manage and control resources. A project is essentially a container for services and resources within GCP. It acts as an isolation boundary that separates and organizes cloud resources, making it easier to manage, monitor, and control access to these resources.

If you are part of an organization, you may already be assigned a project which would have been selected when you logged in. You'd have seen something like this:

If this is true, you may not need to create a project at all. Check with your company policy if testing under this default project is allowed / suggested or should a new project be created for Splunk testing.

To create a new project you would use the gcloud projects create command.

The syntax for this command is:

```
gcloud projects create [PROJECT_ID] [--enable-cloud-apis] [--folder=FOLDER_ID] [--
labels=[KEY=VALUE,…]] [--name=NAME] [--organization=ORGANIZATION_ID] [--set-as-
default]
```

`[PROJECT_ID]:`

This is the unique identifier for the project you want to create. It should be a string of lowercase letters, numbers, and hyphens. This option is required.

`--enable-cloud-apis:`

When specified, this flag enables the Cloud APIs for the newly created project. Cloud APIs are required to use various GCP services, so enabling them ensures that the project can start using GCP resources right away. This is default and doesn't need to be set unless the organization default is to have these disabled

`--folder=FOLDER_ID:`

This option is used to place the newly created project under a specific folder in the GCP resource hierarchy. Folders are used for organizational purposes and help group projects together based on your organizational structure. The folder_id is numeric (ie: 57174735903)

`--labels=[KEY=VALUE,...]:`

Labels are key-value pairs that you can use to add metadata to your projects. They provide a way to categorize and organize projects based on your own custom criteria. Separate labels with commas.

`--name=NAME:`

With this option, you can specify a friendly name for the project. The name does not need to be unique and is used as a human-readable identifier for the project. If you do not supply a name, the name is set as the PROJECT_ID

`--organization=ORGANIZATION_ID:`

If your GCP account is associated with an organization, you can use this option to create the project under a specific organization. An organization helps manage access, billing, and policies across all projects under it. The organization_id is a string of numbers (ie: 655473194031)

`--set-as-default:`

When this flag is used, the newly created project becomes the default project for the gcloud command-line tool. This means that if you run other gcloud commands without specifying the project, it will use the project created here.

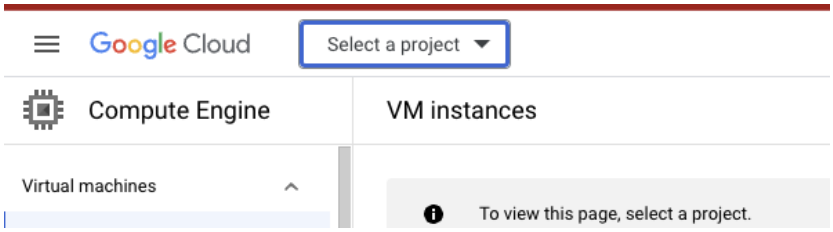
The most basic command to create a project with id terraformblogpost would be:

```
gcloud projects create terraformblogpost
```

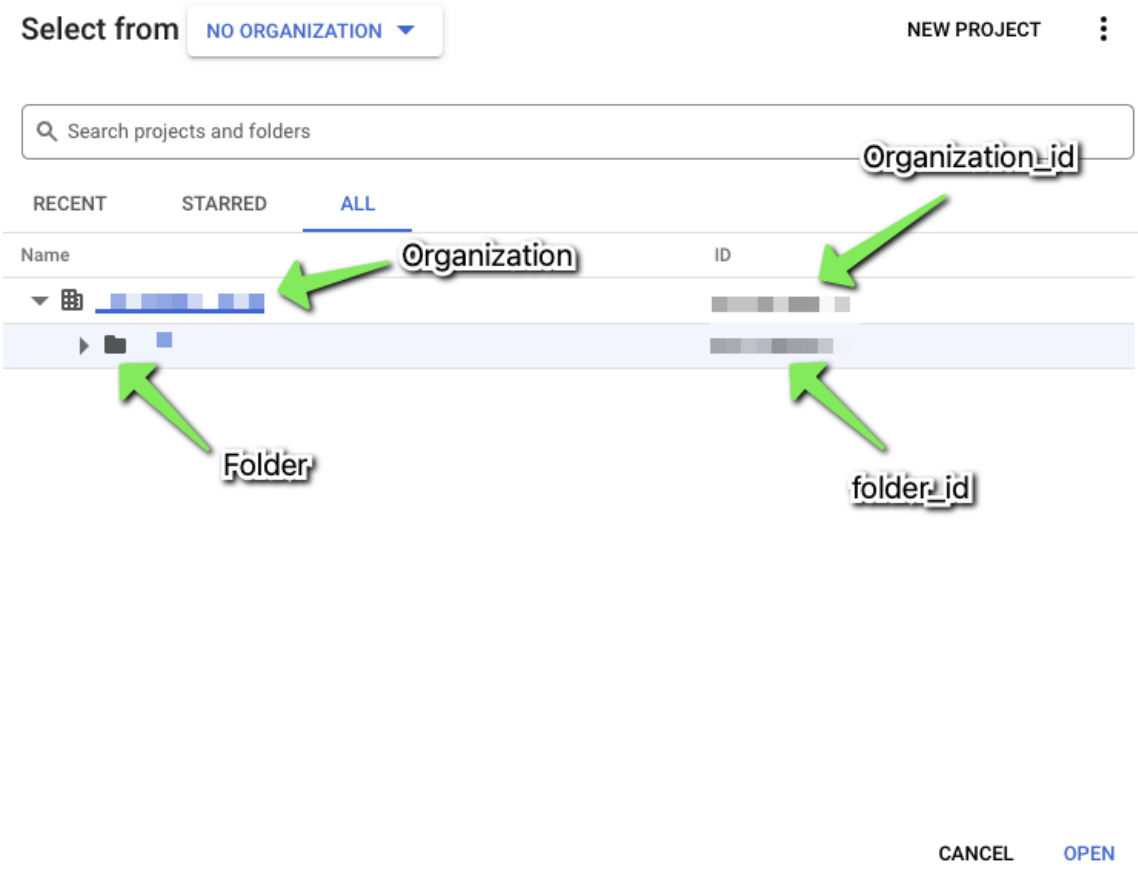
If you get a message like this:


You will need to check with your organization to see if there is a folder or specific location that you can create projects in. For instance… if in the console you go to https://console.cloud.google.com/compute

Click “Select a Project” at the top of the screen




In there, you find everything you need to create a new project including organization_id and (if applicable) folder_id.




(you can also create a new project from within the GCP console)

## Terraform Basics

Now that your computer is set up and you have a GCP project, let's talk about terraform files.

A terraform project is a set of configuration files in a folder on your machine, known as Terraform files, which define the desired state of your infrastructure and the necessary resources to achieve that state. Most terraform files have a .tf extension.

Looking at the anatomy of a basic terraform project, the tree would look like this:

```
my-terraform-project/
├── main.tf
├── variables.tf
```

```
├── outputs.tf
├── provider.tf
├── terraform.tfvars
```

Files:

**my-terraform-project/**: This is the root directory of your Terraform project.

**main.tf**: The primary configuration file for your infrastructure. It defines resources, variables, and other core elements.

**variables.tf**: This file is used to declare input variables that can be used to parameterize your infrastructure configuration.

**outputs.tf**: You define output values in this file that provide information about the resources created in your configuration.

**provider.tf**: Contains provider configurations, specifying which cloud or infrastructure provider(s) you're using.

**terraform.tfvars**: Input variable values are often stored in this file. It's where you set values for variables declared in variables.tf.

## Creating Our First VM...

Now in the simplest Terraform project, all that is required is the main.tf file. Let's create a main.tf file and create our first VM.

```
# main.tf

provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}

resource "google_compute_instance" "example_vm" {
    name = "myfirstvm"
    machine_type = "e2-micro"
    zone = "us-central1-a"
    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-10"
```

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

Looking at that file section by section:

| | |
|---|---|
| `provider "google" {`<br>`  project = "terraform-demo-2024"`<br>`  region = "us-central1"`<br>`}` | The "provider" block tells terraform what cloud provider or infrastructure platform that Terraform will interact with. It specifies the necessary configuration details required to connect to the provider and manage the resources.<br><br>In this case we are using the google provider, connecting to the "terraformdemo" project and creating objects in the us-central1 region. |
| `resource "google_compute_instance" "example_vm"`<br>`{` | Start of the resource block that will create our VM. A resource block section in a Terraform file defines a specific infrastructure resource to be managed.<br><br>We are creating a "google_compute_instance" whose ID is "example_vm". This id will be used to refer to this resource using terraform commands later on. |
| `name = "myfirstvm"` | The rest of the resource block defines configuration details for this resource. Each provider and resource type has it's own set of configurations that can be set as well as what configs are mandatory and what are optional .<br><br>The name parameter sets the hostname for |

| | |
|---|---|
| | the machine |
| `machine_type = "e2-micro"` | The machine_type determines the machine type for the host. For a list of valid machine types in GCP compute instances, go here |
| `zone = "us-central1-a"` | Zone represents the Zone within the region set in the provider block. |
| `boot_disk { initialize_params { image = "debian-cloud/debian-10" } }` | The image setting tells GCP which operating system to install on the new VM. For a full list of GCP images available, you can run : gcloud compute images list |
| `network_interface { network = "default" } }` | This tells GCP that the instance should use the "default" network for the project. |

Now that we have the main.tf file created, to create the VM you would need to run the following commands:

terraform init

*Initializes a new or existing Terraform working directory. It downloads the necessary provider plugins and sets up the backend configuration.*

terraform plan

*Creates an execution plan that shows what actions Terraform will take to reach the desired infrastructure state. It compares the current state with the desired state defined in your Terraform configuration files.*

terraform apply

*Applies the changes required to reach the desired state defined in your Terraform configuration files. It creates, modifies, or deletes resources as necessary. It prompts for confirmation before making any changes.*

Running those in succession, here is what we see:

```
> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/google...
- Installing hashicorp/google v5.16.0...
- Installed hashicorp/google v5.16.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

As you can see, the init command downloaded and installed the hashicorp/google plugin and created a lock file. The lock file is used by Terraform to lock the versions of the provider plugins being used in a particular Terraform configuration. It ensures that the same versions of the provider plugins are used consistently across different environments and by different users, preventing any unexpected changes in behaviour due to version differences. This file is automatically generated and managed by Terraform when you run `terraform init` or `terraform get` commands.

```
> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_instance.example_vm will be created
  + resource "google_compute_instance" "example_vm" {
      + can_ip_forward       = false
      + cpu_platform         = (known after apply)
      + current_status       = (known after apply)
      + deletion_protection  = false
      + effective_labels     = (known after apply)
      + guest_accelerator    = (known after apply)
      + id                   = (known after apply)
      + instance_id          = (known after apply)
```

```
      + label_fingerprint    = (known after apply)
      + machine_type         = "e2-micro"
      + metadata_fingerprint = (known after apply)
      + min_cpu_platform     = (known after apply)
      + name                 = "myfirstvm"
      + project              = "terraform-demo-2024"
      + self_link            = (known after apply)
      + tags_fingerprint     = (known after apply)
      + terraform_labels     = (known after apply)
      + zone                 = "us-central1-a"

      + boot_disk {
          + auto_delete                = true
          + device_name                = (known after apply)
          + disk_encryption_key_sha256 = (known after apply)
          + kms_key_self_link          = (known after apply)
          + mode                       = "READ_WRITE"
          + source                     = (known after apply)

          + initialize_params {
              + image                  = "debian-cloud/debian-10"
              + labels                 = (known after apply)
              + provisioned_iops       = (known after apply)
              + provisioned_throughput = (known after apply)
              + size                   = (known after apply)
              + type                   = (known after apply)
            }
        }

      + network_interface {
          + internal_ipv6_prefix_length = (known after apply)
          + ipv6_access_type            = (known after apply)
          + ipv6_address                = (known after apply)
          + name                        = (known after apply)
          + network                     = "default"
          + network_ip                  = (known after apply)
          + stack_type                  = (known after apply)
          + subnetwork                  = (known after apply)
          + subnetwork_project          = (known after apply)
        }
    }

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

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to
take exactly these actions if
you run "terraform apply" now.
```

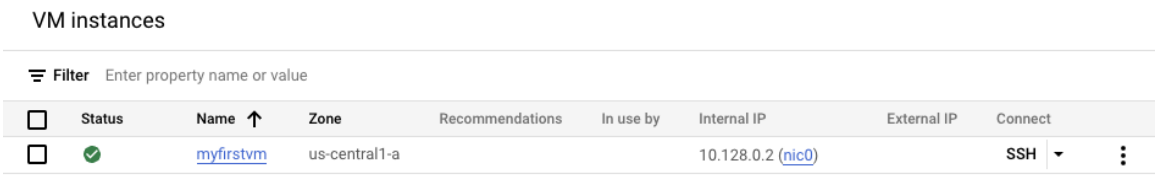
Running the terraform plan, we get the execution plan for what will happen when this terraform project is applied. As you can see there is a single google_compute_instance being created with iD example_vm and host name myfirstvm.

```
> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_instance.example_vm will be created
  + resource "google_compute_instance" "example_vm" {
      + can_ip_forward       = false
      + cpu_platform         = (known after apply)
      + current_status       = (known after apply)
      + deletion_protection  = false
      + effective_labels     = (known after apply)
      + guest_accelerator    = (known after apply)
      + id                   = (known after apply)
      + instance_id          = (known after apply)
      + label_fingerprint    = (known after apply)
      + machine_type         = "e2-micro"
      + metadata_fingerprint = (known after apply)
      + min_cpu_platform     = (known after apply)
      + name                 = "myfirstvm"
      + project              = "terraform-demo-2024"
      + self_link            = (known after apply)
      + tags_fingerprint     = (known after apply)
      + terraform_labels     = (known after apply)
      + zone                 = "us-central1-a"

      + boot_disk {
          + auto_delete                = true
          + device_name                = (known after apply)
          + disk_encryption_key_sha256 = (known after apply)
          + kms_key_self_link          = (known after apply)
          + mode                       = "READ_WRITE"
          + source                     = (known after apply)

          + initialize_params {
              + image                  = "debian-cloud/debian-10"
              + labels                 = (known after apply)
              + provisioned_iops       = (known after apply)
              + provisioned_throughput = (known after apply)
              + size                   = (known after apply)
              + type                   = (known after apply)
            }
        }

      + network_interface {
          + internal_ipv6_prefix_length = (known after apply)
          + ipv6_access_type            = (known after apply)
          + ipv6_address                = (known after apply)
          + name                        = (known after apply)
          + network                     = "default"
          + network_ip                  = (known after apply)
          + stack_type                  = (known after apply)
          + subnetwork                  = (known after apply)
          + subnetwork_project          = (known after apply)
        }
    }
```

```
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.example_vm: Creating...
google_compute_instance.example_vm: Still creating... [10s elapsed]
google_compute_instance.example_vm: Still creating... [20s elapsed]
google_compute_instance.example_vm: Creation complete after 24s [id=projects/terraform-
demo-2024/zones/us-central1-a/instances/myfirstvm]

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

Running terraform apply reran the plan, asked for confirmation that I would like to proceed and then created the VM in 24 seconds.

Looking in my GCP Console Compute Engine page for my project, i can see the new VM in the list




To SSH to this machine, i can use the gcloud binaries

```
gcloud compute ssh "myfirstvm" --project "terraform-demo-2024"
```

```
gcloud compute ssh "myfirstvm" --project "terraform-demo-2024"
External IP address was not found; defaulting to using IAP tunneling.
WARNING:

To increase the performance of the tunnel, consider installing NumPy. For instructions,
please see https://cloud.google.com/iap/docs/using-tcp-
forwarding#increasing_the_tcp_upload_bandwidth

Linux myfirstvm 4.19.0-26-cloud-amd64 #1 SMP Debian 4.19.304-1 (2024-01-09) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
```

```
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb 16 20:13:43 2024 from 35.235.244.33
darren@myfirstvm:~$
```

Now that I am finished with that first VM, I can run the following command from the same directory as the main.tf file: terraform destroy. And all resources in the main.tf file are deleted.

```
> terraform destroy
google_compute_instance.example_vm: Refreshing state... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the
following symbols:
  - destroy

Terraform will perform the following actions:

  # google_compute_instance.example_vm will be destroyed
  - resource "google_compute_instance" "example_vm" {
      - can_ip_forward       = false -> null
      - cpu_platform         = "Intel Broadwell" -> null
      - current_status       = "RUNNING" -> null
      - deletion_protection  = false -> null
      - effective_labels     = {} -> null
      - enable_display       = false -> null
      - guest_accelerator    = [] -> null
      - id                   = "projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm" -> null
      - instance_id          = "6433867197304668466" -> null
      - label_fingerprint    = "42WmSpB8rSM=" -> null
      - labels               = {} -> null
      - machine_type         = "e2-micro" -> null
      - metadata             = {} -> null
      - metadata_fingerprint = "aXmOUjcs5kU=" -> null
      - name                 = "myfirstvm" -> null
      - project              = "terraform-demo-2024" -> null
      - resource_policies    = [] -> null
      - self_link            = "https://www.googleapis.com/compute/v1/projects/terraform-
demo-2024/zones/us-central1-a/instances/myfirstvm" -> null
      - tags                 = [] -> null
      - tags_fingerprint     = "42WmSpB8rSM=" -> null
      - terraform_labels     = {} -> null
      - zone                 = "us-central1-a" -> null

      - boot_disk {
          - auto_delete = true -> null
          - device_name = "persistent-disk-0" -> null
          - mode        = "READ_WRITE" -> null
          - source      = "https://www.googleapis.com/compute/v1/projects/terraform-demo-
```

```
2024/zones/us-central1-a/disks/myfirstvm" -> null

          - initialize_params {
              - enable_confidential_compute = false -> null
              - image                       =
"https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/debian-10-
buster-v20240213" -> null
              - labels                      = {} -> null
              - provisioned_iops            = 0 -> null
              - provisioned_throughput      = 0 -> null
              - resource_manager_tags       = {} -> null
              - size                        = 10 -> null
              - type                        = "pd-standard" -> null
            }
        }

      - network_interface {
          - internal_ipv6_prefix_length = 0 -> null
          - name                        = "nic0" -> null
          - network                     =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-
2024/global/networks/default" -> null
          - network_ip                  = "10.128.0.2" -> null
          - queue_count                 = 0 -> null
          - stack_type                  = "IPV4_ONLY" -> null
          - subnetwork                  =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/regions/us-
central1/subnetworks/default" -> null
          - subnetwork_project          = "terraform-demo-2024" -> null
        }

      - scheduling {
          - automatic_restart   = true -> null
          - min_node_cpus       = 0 -> null
          - on_host_maintenance = "MIGRATE" -> null
          - preemptible         = false -> null
          - provisioning_model  = "STANDARD" -> null
        }

      - shielded_instance_config {
          - enable_integrity_monitoring = true -> null
          - enable_secure_boot          = false -> null
          - enable_vtpm                 = true -> null
        }
    }

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

google_compute_instance.example_vm: Destroying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]
google_compute_instance.example_vm: Still destroying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 10s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 20s elapsed]
```

```
google_compute_instance.example_vm: Destruction complete after 21s

Destroy complete! Resources: 1 destroyed.
```

The destroy command shows an execution plan, in this case destroying 1 host. I confirmed that this was my intention by typing "yes" and the host and all of its resources, drives, ips etc were completely destroyed in 21s.

## Making Infrastructure Changes

Taking the same main.tf file,I am going to show you how easy it is to make changes to the infrastructure being managed by terraform configuration files.

Let's remember what the main.tf file looks like so far:

```
# main.tf

provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}

resource "google_compute_instance" "example_vm" {
    name = "myfirstvm"
    machine_type = "e2-micro"
    zone = "us-central1-a"
    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-10"
        }
    }
    network_interface {
        network = "default"
    }
}
```

I created the myfirstvm host using "`terraform apply`"

Now... what to do if a change is required? What if I want to change the hostname, or machine type..etc.

Changes come in two flavours: destructive and non-destructive. A non-destructive change can be applied on the host in place. For google_compute_instance resources examples of non-destructive changes would include changes to the machine type, updating metadata or

labels, or modifying network settings. Destructive change examples would include changing the host image or changing the zone.

Let's try a change out. I have updated the main.tf file to change the machine_type from e2-micro to e2-small.

When I run terraform plan, I get the following output:

```
> terraform plan
google_compute_instance.example_vm: Refreshing state... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # google_compute_instance.example_vm will be updated in-place
  ~ resource "google_compute_instance" "example_vm" {
        id                   = "projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm"
      ~ machine_type         = "e2-micro" -> "e2-small"
        name                 = "myfirstvm"
        tags                 = []
        # (18 unchanged attributes hidden)

        # (4 unchanged blocks hidden)
    }

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

The plan shows the proposed change on the line marked with a tilde (~) and the plan line at the bottom shows that we are going to change 1 resource (as expected) .

Running terraform apply shows an issue however...

```
> terraform apply
google_compute_instance.example_vm: Refreshing state... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # google_compute_instance.example_vm will be updated in-place
```

```
  ~ resource "google_compute_instance" "example_vm" {
        id                   = "projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm"
      ~ machine_type         = "e2-micro" -> "e2-small"
        name                 = "myfirstvm"
        tags                 = []
        # (18 unchanged attributes hidden)

        # (4 unchanged blocks hidden)
    }

Plan: 0 to add, 1 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.example_vm: Modifying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

╷
│ Error: Changing the machine_type, min_cpu_platform, service_account, enable_display,
shielded_instance_config, scheduling.node_affinities or
network_interface.[#d].(network/subnetwork/subnetwork_project) or
advanced_machine_features on a started instance requires stopping it. To acknowledge this,
please set allow_stopping_for_update = true in your config. You can also stop it by
setting desired_status = "TERMINATED", but the instance will not be restarted after the
update.
│
│   with google_compute_instance.example_vm,
│   on main.tf line 8, in resource "google_compute_instance" "example_vm":
│    8: resource "google_compute_instance" "example_vm" {
│
```

Changing the machine type requires a new configuration added to our resource: allow_stopping_for_update = true. Let's add that into our main.tf file:

```
# main.tf

provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}

resource "google_compute_instance" "example_vm" {
    name = "myfirstvm"
    machine_type = "e2-small"
    zone = "us-central1-a"
    boot_disk {
        initialize_params {
            image = "debian-cloud/debian-10"
        }
    }
```

```
    network_interface {
        network = "default"
    }
    allow_stopping_for_update = true
}
```

Running terraform apply again and the change is completed.

```
> terraform apply
google_compute_instance.example_vm: Refreshing state... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # google_compute_instance.example_vm will be updated in-place
  ~ resource "google_compute_instance" "example_vm" {
      + allow_stopping_for_update = true
        id                        = "projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm"
      ~ machine_type              = "e2-micro" -> "e2-small"
        name                      = "myfirstvm"
        tags                      = []
        # (18 unchanged attributes hidden)

        # (4 unchanged blocks hidden)
    }

Plan: 0 to add, 1 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.example_vm: Modifying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]
google_compute_instance.example_vm: Still modifying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 10s elapsed]
google_compute_instance.example_vm: Still modifying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 20s elapsed]
google_compute_instance.example_vm: Still modifying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 30s elapsed]
google_compute_instance.example_vm: Still modifying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 40s elapsed]
google_compute_instance.example_vm: Modifications complete after 44s
[id=projects/terraform-demo-2024/zones/us-central1-a/instances/myfirstvm]

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

And we can see on the host details page that the change was completed successfully:

## Basic information

| | |
|---|---|
| Name | myfirstvm |
| Instance Id | 1634316693715184915 |
| Description | None |
| Type | Instance |
| Status | Running |
| Creation time | Feb 16, 2024, 4:30:36 PM UTC-05:00 |
| Zone | us-central1-a |
| Instance template | None |
| In use by | None |
| Reservations | Automatically choose (default) |
| Labels | None |
| Tags | — |
| Deletion protection | Disabled |
| Confidential VM service | Disabled |
| Preserved state size | 0 GB |

## Machine configuration

| | |
|---|---|
| Machine type | e2-small |
| CPU platform | Intel Broadwell |
| Minimum CPU platform | None |
| Architecture | x86/64 |
| vCPUs to core ratio | — |
| Custom visible cores | — |
| Display device | Disabled<br>Enable to use screen capturing and recording tools |
| GPUs | None |
| Resource policies | |

If we had opted instead to complete a destructive change the workflow would look like this:

Let's change the host image for the myfirstvm box. Currently we are using Ubuntu as the host image, but you really want RedHat Enterprise Linux for the host instead. To accomplish this, first you need to get the name of the image for RHEL. You can do that using: gcloud compute images list

```
$ gcloud compute images list | grep rhel
rhel-7-v20240213                                rhel-cloud          rhel-7
READY
rhel-8-v20240213                                rhel-cloud          rhel-8
READY
rhel-9-arm64-v20240213                          rhel-cloud          rhel-9-arm64
READY
rhel-9-v20240213                                rhel-cloud          rhel-9
READY
rhel-7-9-sap-v20240213                          rhel-sap-cloud      rhel-7-9-sap-ha
READY
rhel-8-2-sap-v20240213                          rhel-sap-cloud      rhel-8-2-sap-ha
READY
rhel-8-4-sap-v20240214                          rhel-sap-cloud      rhel-8-4-sap-ha
READY
rhel-8-6-sap-v20240213                          rhel-sap-cloud      rhel-8-6-sap-ha
READY
rhel-8-8-sap-v20240213                          rhel-sap-cloud      rhel-8-8-sap-ha
READY
rhel-9-0-sap-v20240213                          rhel-sap-cloud      rhel-9-0-sap-ha
READY
rhel-9-2-sap-v20240213                          rhel-sap-cloud      rhel-9-2-sap-ha
READY
```

We will use the rhel-9-v20240213 image which has project rhel-cloud and family rhel-9. Now edit your main.tf file to replace the image with those new values:

```
$ cat ./main.tf
provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}

resource "google_compute_instance" "example_vm" {
    name = "myfirstvm"
    machine_type = "e2-micro"
    zone = "us-central1-a"
    boot_disk {
        initialize_params {
            image = "rhel-cloud/rhel-9"
        }
    }
    network_interface {
        network = "default"
    }
}
```

When I run terraform plan this time, instead of seeing 1 to change, we get 1 to destroy and 1 to add:

```
$ terraform plan
google_compute_instance.example_vm: Refreshing state... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # google_compute_instance.example_vm must be replaced
-/+ resource "google_compute_instance" "example_vm" {
      ~ cpu_platform         = "Intel Broadwell" -> (known after apply)
      ~ current_status       = "RUNNING" -> (known after apply)
      ~ effective_labels     = {} -> (known after apply)
      - enable_display       = false -> null
      ~ guest_accelerator    = [] -> (known after apply)
      ~ id                   = "projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm" -> (known after apply)
      ~ instance_id          = "4441852230022631571" -> (known after apply)
      ~ label_fingerprint    = "42WmSpB8rSM=" -> (known after apply)
      - labels               = {} -> null
      - metadata             = {} -> null
      ~ metadata_fingerprint = "aXmOUjcs5kU=" -> (known after apply)
      + min_cpu_platform     = (known after apply)
        name                 = "myfirstvm"
      - resource_policies    = [] -> null
      ~ self_link            =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm" -> (known after apply)
      - tags                 = [] -> null
      ~ tags_fingerprint     = "42WmSpB8rSM=" -> (known after apply)
      ~ terraform_labels     = {} -> (known after apply)
        # (5 unchanged attributes hidden)

      ~ boot_disk {
          ~ device_name                = "persistent-disk-0" -> (known after apply)
          + disk_encryption_key_sha256 = (known after apply)
          + kms_key_self_link          = (known after apply)
          ~ source                     =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/zones/us-central1-
a/disks/myfirstvm" -> (known after apply)
            # (2 unchanged attributes hidden)

          ~ initialize_params {
              - enable_confidential_compute = false -> null
              ~ image                       =
"https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/debian-10-
buster-v20240213" -> "rhel-cloud/rhel-9" # forces replacement
              ~ labels                      = {} -> (known after apply)
              ~ provisioned_iops            = 0 -> (known after apply)
              ~ provisioned_throughput      = 0 -> (known after apply)
```

```
            - resource_manager_tags       = {} -> null
            ~ size                        = 10 -> (known after apply)
            ~ type                        = "pd-standard" -> (known after apply)
          }
      }

    ~ network_interface {
        ~ internal_ipv6_prefix_length = 0 -> (known after apply)
        + ipv6_access_type            = (known after apply)
        + ipv6_address                = (known after apply)
        ~ name                        = "nic0" -> (known after apply)
        ~ network                     =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-
2024/global/networks/default" -> "default"
        ~ network_ip                  = "10.128.0.5" -> (known after apply)
        - queue_count                 = 0 -> null
        ~ stack_type                  = "IPV4_ONLY" -> (known after apply)
        ~ subnetwork                  =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/regions/us-
central1/subnetworks/default" -> (known after apply)
        ~ subnetwork_project          = "terraform-demo-2024" -> (known after apply)
      }

    - scheduling {
        - automatic_restart   = true -> null
        - min_node_cpus       = 0 -> null
        - on_host_maintenance = "MIGRATE" -> null
        - preemptible         = false -> null
        - provisioning_model  = "STANDARD" -> null
      }

    - shielded_instance_config {
        - enable_integrity_monitoring = true -> null
        - enable_secure_boot          = false -> null
        - enable_vtpm                 = true -> null
      }
  }

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

Running terraform apply now will destroy the old host and create a new one running rhel 9

```
$ terraform apply
google_compute_instance.example_vm: Refreshing state... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:
```

```
  # google_compute_instance.example_vm must be replaced
-/+ resource "google_compute_instance" "example_vm" {
      ~ cpu_platform         = "Intel Broadwell" -> (known after apply)
      ~ current_status       = "RUNNING" -> (known after apply)
      ~ effective_labels     = {} -> (known after apply)
      - enable_display       = false -> null
      ~ guest_accelerator    = [] -> (known after apply)
      ~ id                   = "projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm" -> (known after apply)
      ~ instance_id          = "4441852230022631571" -> (known after apply)
      ~ label_fingerprint    = "42WmSpB8rSM=" -> (known after apply)
      - labels               = {} -> null
      - metadata             = {} -> null
      ~ metadata_fingerprint = "aXmOUjcs5kU=" -> (known after apply)
      + min_cpu_platform     = (known after apply)
        name                 = "myfirstvm"
      - resource_policies    = [] -> null
      ~ self_link            =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/zones/us-central1-
a/instances/myfirstvm" -> (known after apply)
      - tags                 = [] -> null
      ~ tags_fingerprint     = "42WmSpB8rSM=" -> (known after apply)
      ~ terraform_labels     = {} -> (known after apply)
        # (5 unchanged attributes hidden)

      ~ boot_disk {
          ~ device_name                = "persistent-disk-0" -> (known after apply)
          + disk_encryption_key_sha256 = (known after apply)
          + kms_key_self_link          = (known after apply)
          ~ source                     =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/zones/us-central1-
a/disks/myfirstvm" -> (known after apply)
            # (2 unchanged attributes hidden)

          ~ initialize_params {
              - enable_confidential_compute = false -> null
              ~ image                       =
"https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/debian-10-
buster-v20240213" -> "rhel-cloud/rhel-9" # forces replacement
              ~ labels                      = {} -> (known after apply)
              ~ provisioned_iops            = 0 -> (known after apply)
              ~ provisioned_throughput      = 0 -> (known after apply)
              - resource_manager_tags       = {} -> null
              ~ size                        = 10 -> (known after apply)
              ~ type                        = "pd-standard" -> (known after apply)
            }
        }

      ~ network_interface {
          ~ internal_ipv6_prefix_length = 0 -> (known after apply)
          + ipv6_access_type            = (known after apply)
          + ipv6_address                = (known after apply)
          ~ name                        = "nic0" -> (known after apply)
```

```
          ~ network                     =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-
2024/global/networks/default" -> "default"
          ~ network_ip                  = "10.128.0.5" -> (known after apply)
          - queue_count                 = 0 -> null
          ~ stack_type                  = "IPV4_ONLY" -> (known after apply)
          ~ subnetwork                  =
"https://www.googleapis.com/compute/v1/projects/terraform-demo-2024/regions/us-
central1/subnetworks/default" -> (known after apply)
          ~ subnetwork_project          = "terraform-demo-2024" -> (known after apply)
        }

      - scheduling {
          - automatic_restart   = true -> null
          - min_node_cpus       = 0 -> null
          - on_host_maintenance = "MIGRATE" -> null
          - preemptible         = false -> null
          - provisioning_model  = "STANDARD" -> null
        }

      - shielded_instance_config {
          - enable_integrity_monitoring = true -> null
          - enable_secure_boot          = false -> null
          - enable_vtpm                 = true -> null
        }
    }

Plan: 1 to add, 0 to change, 1 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.example_vm: Destroying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm]
google_compute_instance.example_vm: Still destroying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 10s elapsed]
google_compute_instance.example_vm: Still destroying... [id=projects/terraform-demo-
2024/zones/us-central1-a/instances/myfirstvm, 20s elapsed]
google_compute_instance.example_vm: Destruction complete after 21s
google_compute_instance.example_vm: Creating...
google_compute_instance.example_vm: Still creating... [10s elapsed]
google_compute_instance.example_vm: Creation complete after 13s
[id=projects/terraform-demo-2024/zones/us-central1-a/instances/myfirstvm]

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

Now the boot disk is shown as RHEL 9 for the myfirstvm host

Boot disk

| Name | Image | Interface type | Size (GB) | Device name | Type | Architecture | Encryption | Mode | Wh |
|---|---|---|---|---|---|---|---|---|---|
| myfirstvm | rhel-9-v20240213 | SCSI | 20 | persistent-disk-0 | Standard persistent disk | x86/64 | Google-managed | Boot, read/write | Del |

In the next part of this whitepaper, we will create an All-In-One (AIO) Splunk instance.

- See Part 2 for some more fun where we will demonstrate more terraform basics, create our first Splunk instance and script the installation of Splunk on the host.
- See Part 3 for installing and configuring the deployment server host for our testing environment.
- See Part 4 for installing a series of Universal Forwarders to finish off.

# Part 2: The Splunk All-In-One Host

In Part one, we learned some basics about Terraform, and we created, changed and then subsequently destroyed a VM using terraform. In part two (now that we have some basic terraform skills), we are going to learn some more about how to use terraform to prepare the host and install Splunk.

To accomplish this , I am going to create a bash script that downloads the Splunk installation file, prepares the OS for Splunk installation and installs and configures the splunk instance appropriately.

## Creating a Splunk Enterprise Install Script

Here is the bash script that I used to install Splunk Enterprise on my Ubuntu OS host:  Since this is not a bash tutorial, I am not going to dissect this script too heavily.

```
#!/bin/bash

# --- Logging Configuration ---
# Define a log file for script output
log_file="/tmp/build_splunk_core.log"

# Function to log messages with timestamp
function log_message() {
  local message="$1"
  echo "$(date +'%Y-%m-%d %H:%M:%S') - $message" >> "$log_file" 2>&1
}

# --- Script Execution Check ---
# Check if a flag file exists to indicate previous execution
if [[ -f /tmp/installscriptrun ]]; then
  log_message "Script has already been run. Exiting."
  exit 0
fi

# Create the flag file to mark script execution
touch /tmp/installscriptrun

# --- Variable Definitions ---
# Define Splunk download details
splunk_filename="splunk-9.0.5-e9494146ae5c-Linux-x86_64.tgz"
splunk_download_url="https://download.splunk.com/products/splunk/releases/9.0.5
/linux/splunk-9.0.5-e9494146ae5c-Linux-x86_64.tgz"
```

```
# --- Transparent Huge Pages (THP) Management ---
# Disable Transparent Huge Pages
log_message "Disabling Transparent Huge Pages..."
echo never > /sys/kernel/mm/transparent_hugepage/enabled

# Check if THP is already disabled in GRUB
if ! grep -q "transparent_hugepage=never" /etc/default/grub; then
  log_message "THP not disabled in GRUB. Modifying..."

  # Modify GRUB config to disable THP persistently
  sed -i '/^GRUB_CMDLINE_LINUX=/s/"</ transparent_hugepage=never"/'
/etc/default/grub

  # Update GRUB configuration
  sudo update-grub
else
  log_message "Transparent Huge Pages already disabled in GRUB."
fi

# --- Increase ulimits for Splunk User ---
# log_message "Setting ulimits for Splunk user..."
echo "splunk hard nofile 8192" | sudo tee -a /etc/security/limits.conf
echo "splunk soft nofile 8192" | sudo tee -a /etc/security/limits.conf
echo "splunk hard nproc 4096" | sudo tee -a /etc/security/limits.conf
echo "splunk soft nproc 4096" | sudo tee -a /etc/security/limits.conf
echo "splunk hard stack 8388608" | sudo tee -a /etc/security/limits.conf
echo "splunk soft stack 8388608" | sudo tee -a /etc/security/limits.conf

# Apply ulimit changes
log_message "Applying ulimit changes..."
sudo sysctl -p >> "$log_file" 2>&1

# --- Package Installation ---
# Install required packages (wget and firewalld)
log_message "Installing required packages..."
sudo apt update >> "$log_file" 2>&1;
if ! sudo apt install -y wget firewalld >> "$log_file" 2>&1; then
  log_message "Failed to install packages. Check network/repositories."
  exit 1
fi

# --- Firewall Configuration ---
# Start and enable firewalld
log_message "Starting and enabling firewalld..."
sudo systemctl start firewalld >> "$log_file" 2>&1
sudo systemctl enable firewalld >> "$log_file" 2>&1

# Add firewall rules for Splunk ports
log_message "Adding firewall rules for Splunk communication..."
ports=(8000 8088 8089 9997)
sudo firewall-cmd --permanent --add-port="8000/tcp"
sudo firewall-cmd --permanent --add-port="8088/tcp"
sudo firewall-cmd --permanent --add-port="8089/tcp"
```

```
sudo firewall-cmd --permanent --add-port="9997/tcp"

# Reload firewall rules
log_message "Reloading firewall rules..."
sudo firewall-cmd --reload >> "$log_file" 2>&1

# --- Splunk Installation ---
# Download Splunk Enterprise tarball
log_message "Downloading Splunk Enterprise..."
cd /tmp
if ! sudo wget -O "/tmp/$splunk_filename" "$splunk_download_url" >> "$log_file"
2>&1; then
  log_message "Failed to download Splunk. Check network/URL."
  exit 1
fi

# Untar the downloaded file
log_message "Extracting Splunk tarball..."
if ! sudo tar -xzf "/tmp/$splunk_filename" -C /opt >> "$log_file" 2>&1; then
  log_message "Failed to extract Splunk tarball. Check archive integrity."
  exit 1
fi


# --- Splunk User and Directory Setup ---
# Create Splunk user and group
log_message "Creating Splunk user and group..."
sudo useradd -r -m -d /opt/splunk -s /bin/bash -U splunk >> "$log_file" 2>&1


# --- Ownership and Permissions ---
# Change ownership of Splunk directory to splunk user
log_message "Setting ownership of Splunk directory..."
sudo chown -R splunk:splunk /opt/splunk >> "$log_file" 2>&1


# --- Splunk Configuration ---
# Create necessary folders for Splunk configuration
log_message "Creating Splunk configuration folders..."
su - splunk -c 'mkdir -p /opt/splunk/etc/apps/df_testenvironment_aio/local' >>
"$log_file" 2>&1
su - splunk -c 'mkdir -p /opt/splunk/etc/apps/df_testenvironment_aio/metadata'
>> "$log_file" 2>&1

# Create configuration files (using heredoc syntax for clarity)
log_message "Creating Splunk configuration files..."
cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_aio/metadata/local.meta
[]
access = read : [ * ], write : [ admin ]
export = system
EOT

cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_aio/local/indexes.conf
```

```
[demo]
disabled = false
homePath = \$SPLUNK_DB/demo/db
coldPath = \$SPLUNK_DB/demo/colddb
thawedPath = \$SPLUNK_DB/demo/thaweddb
EOT

cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_aio/local/inputs.conf
[splunktcp://9997]
connection_host = ip
EOT

cat <<EOT >> /opt/splunk/etc/system/local/user-seed.conf
[user_info]
USERNAME = admin
PASSWORD = 1SuperSecretadmiNpassworD
EOT

# Set ownership of configuration files to splunk user
log_message "Setting ownership of Splunk configuration files..."
sudo chown -R splunk:splunk /opt/splunk/etc >> "$log_file" 2>&1


# --- Splunk Startup and Service Management ---
# Start Splunk as the splunk user
log_message "Starting Splunk..."
su - splunk -c '/opt/splunk/bin/splunk start --answer-yes --accept-license' >>
"$log_file" 2>&1

# Enable Splunk to start automatically at boot
log_message "Enabling Splunk boot-start..."
su - splunk -c '/opt/splunk/bin/splunk stop'
sudo /opt/splunk/bin/splunk enable boot-start -systemd-managed 1 -user splunk
>> "$log_file" 2>&1
sudo chown -R splunk:splunk /opt/splunk/etc >> "$log_file" 2>&1
sudo systemctl start Splunkd

# Indicate successful completion
log_message "Splunk AIO is installed and started."
```

I am going to make a new Terraform project called build_testing_environment (which remember a terraform project is just a folder on your disk), create a subfolder called resources, and put the install script into that folder as `./build_splunk_core.sh`

## Building the main.tf

Now that we have the Splunk Enterprise install script in place, let's build our new main.tf for the Testing environment Splunk AIO. We are going to introduce some new concepts with Terraform syntax to achieve our desired outcome.

First, we need to add the provider block, telling Terraform that we are going to be interfacing with Google Cloud.

```
provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}
```

Next, we need a resource block to create the all-in-one Splunk Enterprise instance that will act as a combined Search Head and Indexer for our testing environment:

```
resource "google_compute_instance" "splunk_core_instance_1" {
    project      = "terraform-demo-2024"
    name         = "splunk-aio"
    machine_type = "e2-standard-8"
    zone         = "us-central1-a"
```

In this we are creating a "google_compute_instance" that has a terraform ID "splunk_core_instance_1". The server's hostname is "splunk-aio" and its machine_type is e2-standard-8 (which has 8 CPU cores and 32 GB of RAM). We are creating a 100GB boot disk with Ubuntu 2204 LTS as the operating system.

```
  boot_disk {
       initialize_params {
           size  = 100
           type  = "pd-balanced"
           image = "ubuntu-os-cloud/ubuntu-2204-lts"
       }
    }
```

We want Google Cloud to execute the script we put together above upon startup.

```
    metadata_startup_script = file("${path.module}/resources/build_splunk_core.sh")
```

This will add the public KEY from my local user profile's key pair to the authorized_keys on the new instance under username darren, so i can SSH to the host once it is up.

```
    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }
```

This section tells terraform which network to use.  This configuration is specifying that the resource should use the default network and should have an external IP address with the premium network tier.

```
    network_interface {
        network = "default"

        access_config {
            network_tier = "PREMIUM"
        }
    }
```

Finally, we add some tags to the host which identify the host's purpose and will also be used later for setting up firewall rules for the hosts

```
  tags = [ "splunk", "splunk-core", "splunk-core-aio", "env-test" ]
}
```

With that resource complete, our full main.tf now looks like this:

```
provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}

resource "google_compute_instance" "splunk_core_instance_1" {
    project      = "terraform-demo-2024"
    name         = "splunk-aio"
    machine_type = "e2-standard-8"
    zone         = "us-central1-a"

    boot_disk {
        initialize_params {
            size  = 100
            type  = "pd-balanced"
            image = "ubuntu-os-cloud/ubuntu-2204-lts"
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_core.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }
```

```
    network_interface {
        network = "default"

        access_config {
            network_tier = "PREMIUM"
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-core-aio", "env-test" ]
}
```

Now that we have that complete we need to assign an external IP address for our host so that it can be reached from the internet. To do that we add another resource block, but this time the resource type is google_compute_address:

```
resource "google_compute_address" "splunk_core_address_1" {
    name         = "splunk-core-address-1"
    project      = "terraform-demo-2024"
    region       = "us-central1"
    address_type = "EXTERNAL"
}
```

And to assign this new IP to the host, we will update the access_config section of the host resource block, adding the address to the nat_ip :

```
network_interface {
    network = "default"

    access_config {
        network_tier = "PREMIUM"
         nat_ip = google_compute_address.splunk_core_address_1.address
    }
}
```

So now our full main.tf file looks like this:

```
provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}

resource "google_compute_instance" "splunk_core_instance_1" {
    project      = "terraform-demo-2024"
    name         = "splunk-aio"
    machine_type = "e2-standard-8"
    zone         = "us-central1-a"

    boot_disk {
        initialize_params {
```

```
            size  = 100
            type  = "pd-balanced"
            image = "ubuntu-os-cloud/ubuntu-2204-lts"
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_core.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    network_interface {
        network = "default"

        access_config {
            network_tier = "PREMIUM"
             nat_ip = google_compute_address.splunk_core_address_1.address
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-core-aio", "env-test" ]
}

resource "google_compute_address" "splunk_core_address_1" {
    name         = "splunk-core-address-1"
    project      = "terraform-demo-2024"
    region       = "us-central1"
    address_type = "EXTERNAL"
}
```

Finally, we need to create some firewall rules to allow communication from the internet to our new host. To do that we will introduce yet another resource block for google compute: google_compute_firewall. I am going to create two firewall rules, one which opens ports 22 (ssh), 8000 (Splunk UI) and 8089 (Splunk management port) which would be open on all Splunk instances and a second which opens data ingestion ports 9997 (S2S) and 8088 (HEC) which would only be open on instances that are receiving data (Indexers or All-In-One hosts).

```
resource "google_compute_firewall" "splunk_core_all" {
    project = "terraform-demo-2024"
    name    = "splunk-core-all"
    network = "default"

    allow {
        protocol = "tcp"
        ports    = ["22", "8000", "8089"]
    }

    source_ranges = ["0.0.0.0/0"]
```

```
    target_tags = ["splunk-core"]
}


resource "google_compute_firewall" "splunk_core_aio" {
    project = "terraform-demo-2024"
    name    = "splunk-core-aio"
    network = "default"

    allow {
      protocol = "tcp"
      ports    = ["9997", "8088"]
    }

    source_ranges = ["0.0.0.0/0"]

    target_tags = ["splunk-core-aio"]
}
```

The main things to note on the google_compute_firewall blocks are

- A zone is not required for these
- Protocol can either be "tcp" or "udp"
- Source ranges can be one or more of single IPs or CIDR ranges and accepts a comma separated list of ipv4 and ipv6 style addresses. ["0.0.0.0/0"] opens communication from all source IPs. IPv4 and IPv6 rules should be placed into separate resource blocks and cannot be combined.
- The target_tags identify what tags need to be applied to a host for this firewall rule to be applied. In this example, the splunk_core_all rules will be applied to any host with "splunk-core" tag and the splunk_core_aio rule will be applied to any host with the splunk-core-aio tag. Since our single resource has both of these tags, both of the rules will be applied to that host.

# Building the outputs.tf

With our full main.tf completed for the Splunk AIO host, we are going to create one more file : outputs.tf. This file tells Terraform to print to the screen specific data once the terraform apply is complete.

```
$> cat outputs.tf

output "splunk_core_aio_ip" {
  description = "Public IP address of Splunk AiO instance"
  value                                                                    =
google_compute_instance.splunk_core_instance_1.network_interface[0].access_conf
ig[0].nat_ip
}
```

This will output to screen the nat_ip for our google_compute_instance with id splunk_core_instance_1

Our tree view looks like this at this point

```
build_testing_environment
├── main.tf
├── outputs.tf
└── resources
    └── build_splunk_core.sh
```

# Putting it all together

To create the host, first we run terraform init:

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/google...
- Installing hashicorp/google v5.20.0...
- Installed hashicorp/google v5.20.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

Then we run terraform plan to check that the code is syntactically correct and review the output to ensure it is as expected:

```
$ terraform plan

Terraform used the selected providers to generate the following execution plan.
Resource actions are
indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_address.splunk_core_address_1 will be created
  + resource "google_compute_address" "splunk_core_address_1" {
      + address            = (known after apply)
      + address_type       = "EXTERNAL"
      + creation_timestamp = (known after apply)
      + effective_labels   = (known after apply)
      + id                 = (known after apply)
      + label_fingerprint  = (known after apply)
      + name               = "splunk-core-address-1"
      + network_tier       = (known after apply)
      + prefix_length      = (known after apply)
      + project            = "terraform-demo-2024"
      + purpose            = (known after apply)
      + region             = "us-central1"
      + self_link          = (known after apply)
      + subnetwork         = (known after apply)
      + terraform_labels   = (known after apply)
      + users              = (known after apply)
    }

  # google_compute_firewall.splunk_core_aio will be created
  + resource "google_compute_firewall" "splunk_core_aio" {
      + creation_timestamp = (known after apply)
      + destination_ranges = (known after apply)
      + direction          = (known after apply)
      + enable_logging     = (known after apply)
      + id                 = (known after apply)
      + name               = "splunk-core-aio"
      + network            = "default"
      + priority           = 1000
      + project            = "terraform-demo-2024"
      + self_link          = (known after apply)
      + source_ranges      = [
```

```
        + "0.0.0.0/0",
      ]
    + target_tags        = [
        + "splunk-core-aio",
      ]

    + allow {
        + ports    = [
            + "9997",
            + "8088",
          ]
        + protocol = "tcp"
      }
  }

# google_compute_firewall.splunk_core_all will be created
+ resource "google_compute_firewall" "splunk_core_all" {
    + creation_timestamp = (known after apply)
    + destination_ranges = (known after apply)
    + direction          = (known after apply)
    + enable_logging     = (known after apply)
    + id                 = (known after apply)
    + name               = "splunk-core-all"
    + network            = "default"
    + priority           = 1000
    + project            = "terraform-demo-2024"
    + self_link          = (known after apply)
    + source_ranges      = [
        + "0.0.0.0/0",
      ]
    + target_tags        = [
        + "splunk-core",
      ]

    + allow {
        + ports    = [
            + "22",
            + "8000",
            + "8089",
          ]
        + protocol = "tcp"
      }
  }

# google_compute_instance.splunk_core_instance_1 will be created
+ resource "google_compute_instance" "splunk_core_instance_1" {
    + can_ip_forward          = false
    + cpu_platform            = (known after apply)
    + current_status          = (known after apply)
    + deletion_protection     = false
    + effective_labels        = (known after apply)
    + guest_accelerator       = (known after apply)
    + id                      = (known after apply)
```

```
      + instance_id             = (known after apply)
      + label_fingerprint       = (known after apply)
      + machine_type            = "e2-standard-8"
      + metadata                = {
          + "ssh-keys" = <<-EOT
              darren:ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQC9gOg5Kq9yrqHoWwIpSpTT/BdFpB1SGlwvTE5CTHzjpntq+aQ04WJdQz
LW3EEjPXY6yBtOfIIx5VG40/OiFsQU1HOk1rnTRMgl5hA8S27qD0dpVbxfZGS6YjK3gTuJMiBjVTOkCX/caGdc
1KDiN7PW4bwiatkvg6C4Q7xMhRm/Rg93izYjs3X9pUjk210P2Vxd0EqhyhRAiughpvTsPou4NpSRFMBRktPUuD
GtBuMcBdxhHeDVJg/fxZUH0RxPDQwFEuOrNm1DsPBjkKJnsRGP4kHR0se7GkiDi1FfBz+s4VQe4py1qrK1ein2
UH6VpsvSGcm8mlISWS/uDFcOHzCb4p5eOSVpPmn+qZzc4SxMotHw/hb3eg5EAWkS8uo3pec8RHR5X9uDNJOjc8
p7m4yuWNd0OT8QTjnnUOuIuIXw8mLiWGFYL0cvHCU4RF66HUFwdZomGb5vYcmWEF72/iBEQxZsIK8EIfXPq2TV
3CLn6vRGVGL3P7F3LhV9Nu+v8Pc= darren
              EOT
        }
      + metadata_fingerprint    = (known after apply)
      + metadata_startup_script = <<-EOT
            #!/bin/bash

            # --- Logging Configuration ---
            # Define a log file for script output
            log_file="/tmp/build_splunk_core.log"

            # Function to log messages with timestamp
            function log_message() {
              local message="$1"
              echo "$(date +'%Y-%m-%d %H:%M:%S') - $message" >> "$log_file" 2>&1
            }

            # --- Script Execution Check ---
            # Check if a flag file exists to indicate previous execution
            if [[ -f /tmp/installscriptrun ]]; then
              log_message "Script has already been run. Exiting."
              exit 0
            fi

            # Create the flag file to mark script execution
            touch /tmp/installscriptrun

            # --- Variable Definitions ---
            # Define Splunk download details
            splunk_filename="splunk-9.0.5-e9494146ae5c-Linux-x86_64.tgz"

splunk_download_url="https://download.splunk.com/products/splunk/releases/9.0.5/linux/
splunk-9.0.5-e9494146ae5c-Linux-x86_64.tgz"

            # --- Transparent Huge Pages (THP) Management ---
            # Disable Transparent Huge Pages
            log_message "Disabling Transparent Huge Pages..."
            echo never > /sys/kernel/mm/transparent_hugepage/enabled

            # Check if THP is already disabled in GRUB
            if ! grep -q "transparent_hugepage=never" /etc/default/grub; then
```

```
    log_message "THP not disabled in GRUB. Modifying..."

    # Modify GRUB config to disable THP persistently
    sed -i '/^GRUB_CMDLINE_LINUX=/s/"</ transparent_hugepage=never"/'
/etc/default/grub

    # Update GRUB configuration
    sudo update-grub
  else
    log_message "Transparent Huge Pages already disabled in GRUB."
  fi

  # --- Increase ulimits for Splunk User ---
  # log_message "Setting ulimits for Splunk user..."
  echo "splunk hard nofile 8192" | sudo tee -a /etc/security/limits.conf
  echo "splunk soft nofile 8192" | sudo tee -a /etc/security/limits.conf
  echo "splunk hard nproc 4096" | sudo tee -a /etc/security/limits.conf
  echo "splunk soft nproc 4096" | sudo tee -a /etc/security/limits.conf
  echo "splunk hard stack 8388608" | sudo tee -a /etc/security/limits.conf
  echo "splunk soft stack 8388608" | sudo tee -a /etc/security/limits.conf

  # Apply ulimit changes
  log_message "Applying ulimit changes..."
  sudo sysctl -p >> "$log_file" 2>&1

  # --- Package Installation ---
  # Install required packages (wget and firewalld)
  log_message "Installing required packages..."
  sudo apt update >> "$log_file" 2>&1;
  if ! sudo apt install -y wget firewalld >> "$log_file" 2>&1; then
    log_message "Failed to install packages. Check network/repositories."
    exit 1
  fi

  # --- Firewall Configuration ---
  # Start and enable firewalld
  log_message "Starting and enabling firewalld..."
  sudo systemctl start firewalld >> "$log_file" 2>&1
  sudo systemctl enable firewalld >> "$log_file" 2>&1

  # Add firewall rules for Splunk ports
  log_message "Adding firewall rules for Splunk communication..."
  ports=(8000 8088 8089 9997)
  sudo firewall-cmd --permanent --add-port="8000/tcp"
  sudo firewall-cmd --permanent --add-port="8088/tcp"
  sudo firewall-cmd --permanent --add-port="8089/tcp"
  sudo firewall-cmd --permanent --add-port="9997/tcp"

  # Reload firewall rules
  log_message "Reloading firewall rules..."
  sudo firewall-cmd --reload >> "$log_file" 2>&1

  # --- Splunk Installation ---
```

```
            # Download Splunk Enterprise tarball
            log_message "Downloading Splunk Enterprise..."
            if ! sudo wget -O "/tmp/$splunk_filename" "$splunk_download_url" >>
"$log_file" 2>&1; then
              log_message "Failed to download Splunk. Check network/URL."
              exit 1
            fi

            # Untar the downloaded file
            log_message "Extracting Splunk tarball..."
            if ! sudo tar -xzf "/tmp/$splunk_filename" -C /opt >> "$log_file" 2>&1;
then
              log_message "Failed to extract Splunk tarball. Check archive integrity."
              exit 1
            fi


            # --- Splunk User and Directory Setup ---
            # Create Splunk user and group
            log_message "Creating Splunk user and group..."
            sudo useradd -r -m -d /opt/splunk -s /bin/bash -U splunk >> "$log_file"
2>&1


            # --- Ownership and Permissions ---
            # Change ownership of Splunk directory to splunk user
            log_message "Setting ownership of Splunk directory..."
            sudo chown -R splunk:splunk /opt/splunk >> "$log_file" 2>&1


            # --- Splunk Configuration ---
            # Create necessary folders for Splunk configuration
            log_message "Creating Splunk configuration folders..."
            su - splunk -c 'mkdir -p
/opt/splunk/etc/apps/df_testenvironment_aio/local' >> "$log_file" 2>&1
            su - splunk -c 'mkdir -p
/opt/splunk/etc/apps/df_testenvironment_aio/metadata' >> "$log_file" 2>&1

            # Create configuration files (using heredoc syntax for clarity)
            log_message "Creating Splunk configuration files..."
            cat <<EOT >>
/opt/splunk/etc/apps/df_testenvironment_aio/metadata/local.meta
            []
            access = read : [ * ], write : [ admin ]
            export = system
            EOT

            cat <<EOT >>
/opt/splunk/etc/apps/df_testenvironment_aio/local/indexes.conf
            [demo]
            disabled = false
            homePath = \$SPLUNK_DB/demo/db
            coldPath = \$SPLUNK_DB/demo/colddb
```

```
            thawedPath = \$SPLUNK_DB/demo/thaweddb
            EOT

            cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_aio/local/inputs.conf
            [splunktcp://9997]
            connection_host = ip
            EOT

            cat <<EOT >> /opt/splunk/etc/system/local/user-seed.conf
            [user_info]
            USERNAME = admin
            PASSWORD = 1SuperSecretadmiNpasswordD
            EOT

            # Set ownership of configuration files to splunk user
            log_message "Setting ownership of Splunk configuration files..."
            sudo chown -R splunk:splunk /opt/splunk/etc >> "$log_file" 2>&1


            # --- Splunk Startup and Service Management ---
            # Start Splunk as the splunk user
            log_message "Starting Splunk..."
            su - splunk -c '/opt/splunk/bin/splunk start --answer-yes --accept-
license' >> "$log_file" 2>&1

            # Enable Splunk to start automatically at boot
            log_message "Enabling Splunk boot-start..."
            su - splunk -c '/opt/splunk/bin/splunk stop'
            sudo /opt/splunk/bin/splunk enable boot-start -systemd-managed 1 -user
splunk >> "$log_file" 2>&1
            sudo chown -R splunk:splunk /opt/splunk/etc >> "$log_file" 2>&1
            sudo systemctl start Splunkd

            # Indicate successful completion
            log_message "Splunk AIO is installed and started."
          EOT
        + min_cpu_platform        = (known after apply)
        + name                    = "splunk-aio"
        + project                 = "terraform-demo-2024"
        + self_link               = (known after apply)
        + tags                    = [
            + "env-test",
            + "splunk",
            + "splunk-core",
            + "splunk-core-aio",
          ]
        + tags_fingerprint        = (known after apply)
        + terraform_labels        = (known after apply)
        + zone                    = "us-central1-a"

        + boot_disk {
            + auto_delete                = true
            + device_name                = (known after apply)
```

```
      + disk_encryption_key_sha256 = (known after apply)
      + kms_key_self_link          = (known after apply)
      + mode                       = "READ_WRITE"
      + source                     = (known after apply)

      + initialize_params {
          + image                  = "ubuntu-os-cloud/ubuntu-2204-lts"
          + labels                 = (known after apply)
          + provisioned_iops       = (known after apply)
          + provisioned_throughput = (known after apply)
          + size                   = 100
          + type                   = "pd-balanced"
        }
    }

    + network_interface {
        + internal_ipv6_prefix_length = (known after apply)
        + ipv6_access_type            = (known after apply)
        + ipv6_address                = (known after apply)
        + name                        = (known after apply)
        + network                     = "default"
        + network_ip                  = (known after apply)
        + stack_type                  = (known after apply)
        + subnetwork                  = (known after apply)
        + subnetwork_project          = (known after apply)

        + access_config {
            + nat_ip       = (known after apply)
            + network_tier = "PREMIUM"
          }
      }
  }

Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + splunk_core_aio_ip = (known after apply)

──────────────────────────────────────────────────────────────────────────────
──────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee
to take exactly
these actions if you run "terraform apply" now.
```

This shows 4 objects that are being created. 1 google_compute_instance, 1 google_compute_address and 2 google_compute_firewall and it will output the splunk_core_aio_ip.. Since this is the expected result, we can apply the code with terraform apply.

```
$ terraform apply
```

```
<< snipped plan output as it is seen above >>

...
Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + splunk_core_aio_ip = (known after apply)

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_firewall.splunk_core_aio: Creating...
google_compute_firewall.splunk_core_all: Creating...
google_compute_address.splunk_core_address_1: Creating...
google_compute_firewall.splunk_core_aio: Still creating... [10s elapsed]
google_compute_firewall.splunk_core_all: Still creating... [10s elapsed]
google_compute_address.splunk_core_address_1: Still creating... [10s elapsed]
google_compute_address.splunk_core_address_1: Creation complete after 12s
[id=projects/terraform-demo-2024/regions/us-central1/addresses/splunk-core-
address-1]
google_compute_instance.splunk_core_instance_1: Creating...
google_compute_firewall.splunk_core_aio: Creation complete after 12s
[id=projects/terraform-demo-2024/global/firewalls/splunk-core-aio]
google_compute_firewall.splunk_core_all: Creation complete after 12s
[id=projects/terraform-demo-2024/global/firewalls/splunk-core-all]
google_compute_instance.splunk_core_instance_1: Still creating... [10s elapsed]
google_compute_instance.splunk_core_instance_1: Creation complete after 13s
[id=projects/terraform-demo-2024/zones/us-central1-a/instances/splunk-aio]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

splunk_core_aio_ip = "34.71.1.185"
```

The apply command completes successfully in a few moments and the IP address for the new instance is `34.71.1.185`.

## Log into the newly created AIO host

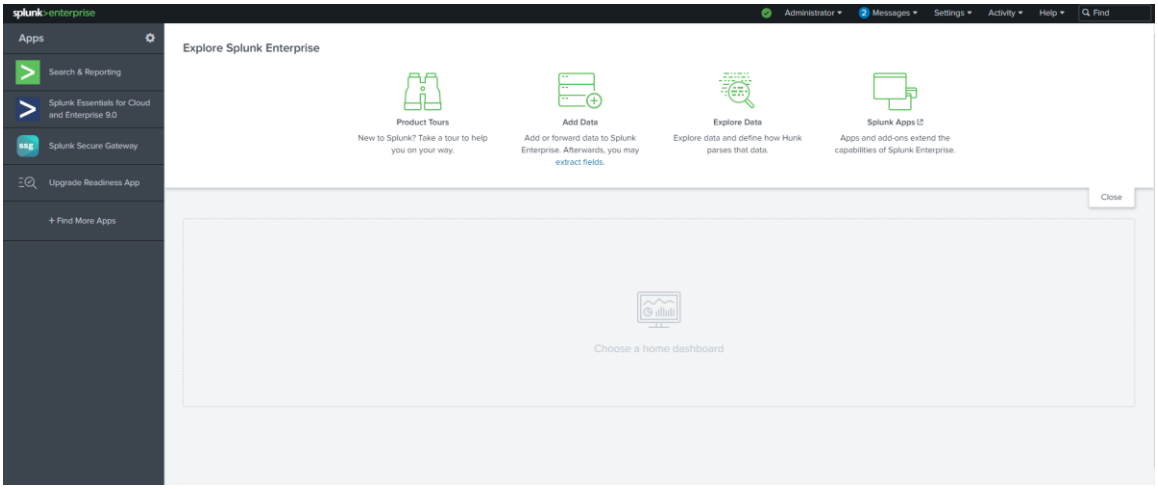
Since i have used the default ssh key for my user profile with username darren, I can now log into the host using

```
ssh darren@35.223.244.161
```

The installation script logs its actions to /tmp/build_splunk_core.log. Review that file for any errors and if none are found, then log into Splunk with the IP address and port 8000. Note that the communication is http since https was not set up in the web.conf when I set up the config files in the startup script.

http://35.223.244.161:8000

In my script, I set the admin login to username: admin password: 1SuperSecretadmiNpassworD. When I open the Splunk UI, I can login with those credentials and I am ready to go with my AIO Splunk instance.



Thanks for reading. In the next part, we discuss deployment and configuration of the Splunk deployment server.

- Part 1 - Introduction to Terraform
- Part 2 - Deploy, install and configure the Splunk All-in-One host
- Part 3 - Deploy, install and configure the Splunk Deployment Server
- Part 4 - Install a series of Universal Forwarders

# Part 3: Splunk Deployment Server Installation & Configuration

In Part one, we learned some basics about Terraform, and we created, changed and then subsequently destroyed a VM using terraform. Then in part two we learned how to set up a startup script on the GCP host and we used terraform to prepare the host and install Splunk on our AIO host. In this the third part, we are going to set up our Deployment Server host, prepare it to deploy our app to deployment clients and set it out to output its internal logs to the main Splunk AIO host.

## Creating a Deployment Server Install Script

First, we are going to create another bash script for installing Splunk. This will be very similar to the one we used in Part 2, but with a few differences. The new script is located in the same ./resources directory as the build_splunk_core.sh and is called build_splunk_ds.sh.

Here are the differences between the splunk core and ds installation scripts:

1) We don't need to open ports 9997 and 8088 on firewalld as the DS does not need to have data ingest turned on,

2) In the config file creation section, I don't create the inputs.conf file and instead create an outputs.conf to point the DS's internal logs to the AIO splunk host. Note, I am using GCP DNS to point to the host using <hostname>.c.<project>.internal so in the case of this demo, the AIO hostname is splunk-aio and the project is called terraformdemo2024, so I point the outputs.conf to splunk-`aio.c.terraformdemo2024.internal:9997`

3) I create an df_all_forwarderoutputs app in `$SPLUNK_HOME/etc/deployment-apps` which sets outputs.conf for our deployment clients

4) I uploaded the app I am testing to google drive (TA-demo_app) and set its sharing settings to "Anyone with the link". In the script I am using wget to download the file

to /tmp and then install it to $SPLUNK_HOME/etc/deployment-apps so that it will be installed to the deployment clients.

5) I create $SPLUNK_HOME/etc/apps/TA-demo_app/local/inputs.conf for the TA-demo_app so it collects data and sends to

6) Finally, still in the config file creation section, I created a serverclass.conf which defines a single serverclass (demoapp), whitelists all hosts that connect to forwarder management (whitelist.0 = *) and installs the app that for testing (TA-demo_app)

```
#!/bin/bash

# --- Logging Configuration ---
# Define a log file for script output
log_file="/tmp/build_splunk_ds.log"

# Function to log messages with timestamp
function log_message() {
  local message="$1"
  echo "$(date +'%Y-%m-%d %H:%M:%S') - $message" >> "$log_file" 2>&1
}

# --- Script Execution Check ---
# Check if a flag file exists to indicate previous execution
if [[ -f /tmp/installscriptrun ]]; then
  log_message "Script has already been run. Exiting."
  exit 0
fi

# Create the flag file to mark script execution
touch /tmp/installscriptrun

# --- Variable Definitions ---
# Define Splunk download details
splunk_filename="splunk-9.0.5-e9494146ae5c-Linux-x86_64.tgz"
splunk_download_url="https://download.splunk.com/products/splunk/releases/9.0.5
/linux/splunk-9.0.5-e9494146ae5c-Linux-x86_64.tgz"

# --- Transparent Huge Pages (THP) Management ---
# Disable Transparent Huge Pages
log_message "Disabling Transparent Huge Pages..."
echo never > /sys/kernel/mm/transparent_hugepage/enabled

# Check if THP is already disabled in GRUB
if ! grep -q "transparent_hugepage=never" /etc/default/grub; then
  log_message "THP not disabled in GRUB. Modifying..."

  # Modify GRUB config to disable THP persistently
```

```
  sed -i '/^GRUB_CMDLINE_LINUX=/s/"</ transparent_hugepage=never"/'
/etc/default/grub

  # Update GRUB configuration
  sudo update-grub
else
  log_message "Transparent Huge Pages already disabled in GRUB."
fi

# --- Increase ulimits for Splunk User ---
# log_message "Setting ulimits for Splunk user..."
echo "splunk hard nofile 8192" | sudo tee -a /etc/security/limits.conf
echo "splunk soft nofile 8192" | sudo tee -a /etc/security/limits.conf
echo "splunk hard nproc 4096" | sudo tee -a /etc/security/limits.conf
echo "splunk soft nproc 4096" | sudo tee -a /etc/security/limits.conf
echo "splunk hard stack 8388608" | sudo tee -a /etc/security/limits.conf
echo "splunk soft stack 8388608" | sudo tee -a /etc/security/limits.conf

# Apply ulimit changes
log_message "Applying ulimit changes..."
sudo sysctl -p >> "$log_file" 2>&1

# --- Package Installation ---
# Install required packages (wget and firewalld)
log_message "Installing required packages..."
sudo apt update >> "$log_file" 2>&1;
if ! sudo apt install -y wget firewalld >> "$log_file" 2>&1; then
  log_message "Failed to install packages. Check network/repositories."
  exit 1
fi

# --- Firewall Configuration ---
# Start and enable firewalld
log_message "Starting and enabling firewalld..."
sudo systemctl start firewalld >> "$log_file" 2>&1
sudo systemctl enable firewalld >> "$log_file" 2>&1

# Add firewall rules for Splunk ports
log_message "Adding firewall rules for Splunk communication..."
sudo firewall-cmd --permanent --add-port="8000/tcp"
sudo firewall-cmd --permanent --add-port="8089/tcp"

# Reload firewall rules
log_message "Reloading firewall rules..."
sudo firewall-cmd --reload >> "$log_file" 2>&1

# --- Splunk Installation ---
# Download Splunk Enterprise tarball
log_message "Downloading Splunk Enterprise..."
if ! sudo wget -O "/tmp/$splunk_filename" "$splunk_download_url" >> "$log_file"
2>&1; then
  log_message "Failed to download Splunk. Check network/URL."
  exit 1
```

```
fi

# Untar the downloaded file
log_message "Extracting Splunk tarball..."
if ! sudo tar -xzf "/tmp/$splunk_filename" -C /opt >> "$log_file" 2>&1; then
  log_message "Failed to extract Splunk tarball. Check archive integrity."
  exit 1
fi


# --- Splunk User and Directory Setup ---
# Create Splunk user and group
log_message "Creating Splunk user and group..."
sudo useradd -r -m -d /opt/splunk -s /bin/bash -U splunk >> "$log_file" 2>&1


# --- Ownership and Permissions ---
# Change ownership of Splunk directory to splunk user
log_message "Setting ownership of Splunk directory..."
sudo chown -R splunk:splunk /opt/splunk >> "$log_file" 2>&1

# --- Download app to be tested to deployment server
# Note: App has been saved and shared in Google Drive with
#       "Anyone with the link can access" permissions
log_message "Downloading TA-demo_app from Google Drive"
if ! sudo wget -O "/tmp/TA-demo_app.tgz" "https://drive.google.com/file/d/1jH3-
07md8rXYzyMT7Jn841C-xoTWB6BB/view?usp=sharing" >> "$log_file" 2>&1; then
  log_message "Failed to download Splunk. Check network/URL."
  exit 1
fi

# --- Splunk Configuration ---
# Create necessary folders for Splunk configuration
log_message "Creating Splunk configuration folders..."
su - splunk -c 'mkdir -p /opt/splunk/etc/apps/df_testenvironment_ds/local' >>
"$log_file" 2>&1
su - splunk -c 'mkdir -p /opt/splunk/etc/apps/df_testenvironment_ds/metadata'
>> "$log_file" 2>&1
su - splunk -c 'mkdir -p /opt/splunk/etc/deployment-
apps/df_all_forwarderoutputs/local' >> "$log_file" 2>&1
su - splunk -c 'mkdir -p /opt/splunk/etc/deployment-
apps/df_all_forwarderoutputs/metadata' >> "$log_file" 2>&1

# Create configuration files (using heredoc syntax for clarity)
log_message "Creating Splunk configuration files..."

cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_ds/metadata/local.meta
[]
access = read : [ * ], write : [ admin ]
export = system
EOT

cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_ds/local/outputs.conf
```

```
[tcpout]
defaultGroup = aio
[tcpout:aio]
server = splunk-aio.c.terraformdemo2024.internal:9997
EOT

cat <<EOT >> /opt/splunk/etc/apps/df_testenvironment_ds/local/serverclass.conf
[serverClass:demoapp]
filterType = whitelist
whitelist.0 = *
[serverClass:demoapp:app:TA-demo_app]
stateOnClient = enabled
restartSplunkd = true
[serverClass:demoapp:app:df_all_forwarderoutputs]
stateOnClient = enabled
restartSplunkd = true
EOT

cat <<EOT >> /opt/splunk/etc/system/local/user-seed.conf
[user_info]
USERNAME = admin
PASSWORD = 1SuperSecretadmiNpassworD
EOT

cat <<EOT >> /opt/splunk/etc/deployment-
apps/df_all_forwarderoutputs/metadata/local.meta
[]
access = read : [ * ], write : [ admin ]
export = system
EOT

cat <<EOT >> /opt/splunk/etc/deployment-
apps/df_all_forwarderoutputs/local/outputs.conf
[tcpout]
defaultGroup = aio
[tcpout:aio]
server = splunk-aio.c.terraformdemo2024.internal:9997
EOT

# Download TA-demo_app from Google drive and install to deployment-apps folder
log_message "Downloading TA-demo_app"
if ! sudo wget -O "/tmp/TA-demo_app.tgz" "https://drive.google.com/file/d/1jH3-
07md8rXYzyMT7Jn841C-xoTWB6BB" >> "$log_file" 2>&1; then
  log_message "Failed to download Demo App. Check network/URL."
  exit 1
fi
if ! tar -xzf /tmp/TA-demo_app.tgz -C /opt/splunk/etc/deployment-apps/ >>
"$log_file" 2>&1; then
  log_message "Failed to untar TA-demo_app.tgz.   Check file integrity."
  exit 1
fi

# Set ownership of configuration files to splunk user
```

```
log_message "Setting ownership of Splunk configuration files..."
sudo chown -R splunk:splunk /opt/splunk/etc >> "$log_file" 2>&1


# --- Splunk Startup and Service Management ---
# Start Splunk as the splunk user
log_message "Starting Splunk..."
su - splunk -c '/opt/splunk/bin/splunk start --answer-yes --accept-license' >>
"$log_file" 2>&1

# Enable Splunk to start automatically at boot
log_message "Enabling Splunk boot-start..."
su - splunk -c '/opt/splunk/bin/splunk stop'
sudo /opt/splunk/bin/splunk enable boot-start -systemd-managed 1 -user splunk
>> "$log_file" 2>&1
sudo chown -R splunk:splunk /opt/splunk/etc >> "$log_file" 2>&1
sudo systemctl start Splunkd

# Indicate successful completion
log_message "Splunk DS is installed and started."
```

## Amending the main.tf

With the new script in place in the resources directory, now I need to alter the main.tf file to add resource definitions for the new infrastructure. This will require adding two more resource blocks, another google_compute_instance and another google_compute_address. Both will look familiar, though the google_compute_instance block will point to the DS setup script instead of the AIO one we used in the previous host definition.

```
resource "google_compute_instance" "splunk_core_instance_2" {
    project      = "terraform-demo-2024"
    name         = "splunk-ds"
    machine_type = "e2-small"
    zone         = "us-central1-a"

    boot_disk {
        initialize_params {
            size  = 20
            type  = "pd-balanced"
            image = "ubuntu-os-cloud/ubuntu-2204-lts"
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_ds.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
```

```
    }

    network_interface {
        network = "default"

        access_config {
            network_tier = "PREMIUM"
           nat_ip = google_compute_address.splunk_core_address_2.address
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-ds", "env-test" ]
}


resource "google_compute_address" "splunk_core_address_2" {
    name         = "splunk-core-address-2"
    project      = "terraform-demo-2024"
    region       = "us-central1"
    address_type = "EXTERNAL"
}
```

Note, additional firewall rules are not required as the original firewall rule "splunk_core_all" seen below opened the required ports for the DS (8000 and 8089) and this is applied to the new host because of its target tags pointing to "splunk-core" which was applied to the host definition.

```
resource "google_compute_firewall" "splunk_core_all" {
    project = "terraform-demo-2024"
    name    = "splunk-core-all"
    network = "default"

    allow {
        protocol = "tcp"
        ports    = ["22", "8000", "8089"]
    }

    source_ranges = ["0.0.0.0/0"]

    target_tags = ["splunk-core"]
}
```

The whole main.tf now looks like this:

```
provider "google" {
    project = "terraform-demo-2024"
    region = "us-central1"
}
```

```
resource "google_compute_instance" "splunk_core_instance_1" {
    project      = "terraform-demo-2024"
    name         = "splunk-aio"
    machine_type = "e2-standard-8"
    zone         = "us-central1-a"

    boot_disk {
        initialize_params {
            size  = 100
            type  = "pd-balanced"
            image = "ubuntu-os-cloud/ubuntu-2204-lts"
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_core.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    network_interface {
        network = "default"

        access_config {
            network_tier = "PREMIUM"
           nat_ip = google_compute_address.splunk_core_address_1.address
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-core-aio", "env-test" ]
}


resource "google_compute_address" "splunk_core_address_1" {
    name         = "splunk-core-address-1"
    project      = "terraform-demo-2024"
    region       = "us-central1"
    address_type = "EXTERNAL"
}


resource "google_compute_instance" "splunk_core_instance_2" {
    project      = "terraform-demo-2024"
    name         = "splunk-ds"
    machine_type = "e2-small"
    zone         = "us-central1-a"

    boot_disk {
        initialize_params {
            size  = 20
            type  = "pd-balanced"
            image = "ubuntu-os-cloud/ubuntu-2204-lts"
```

```
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_ds.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    network_interface {
        network = "default"

        access_config {
            network_tier = "PREMIUM"
           nat_ip = google_compute_address.splunk_core_address_2.address
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-ds", "env-test" ]
}


resource "google_compute_address" "splunk_core_address_2" {
    name         = "splunk-core-address-2"
    project      = "terraform-demo-2024"
    region       = "us-central1"
    address_type = "EXTERNAL"
}


resource "google_compute_firewall" "splunk_core_all" {
    project = "terraform-demo-2024"
    name    = "splunk-core-all"
    network = "default"

    allow {
        protocol = "tcp"
        ports    = ["22", "8000", "8089"]
    }

    source_ranges = ["0.0.0.0/0"]

    target_tags = ["splunk-core"]
}


resource "google_compute_firewall" "splunk_core_aio" {
    project = "terraform-demo-2024"
    name    = "splunk-core-aio"
    network = "default"

    allow {
```

```
      protocol = "tcp"
      ports    = ["9997", "8088"]
    }

    source_ranges = ["0.0.0.0/0"]

    target_tags = ["splunk-core-aio"]
}
```

## Amending the outputs.tf

Lastly before I apply the terraform code, I want to add the output of the DS IP address to the outputs.tf file:

```
output "splunk_core_aio_ip" {
    description = "Public IP address of Splunk AiO instance"
    value                                                                    =
google_compute_instance.splunk_core_instance_1.network_interface[0].access_conf
ig[0].nat_ip
}

output "splunk_core_ds_ip" {
    description = "Public IP address of Splunk DS instance"
    value                                                                    =
google_compute_instance.splunk_core_instance_2.network_interface[0].access_conf
ig[0].nat_ip
}
```

## Putting it all together

Now when I run terraform plan, I get the following output:

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

Changes to Outputs:
  + splunk_core_ds_ip  = (known after apply)
```

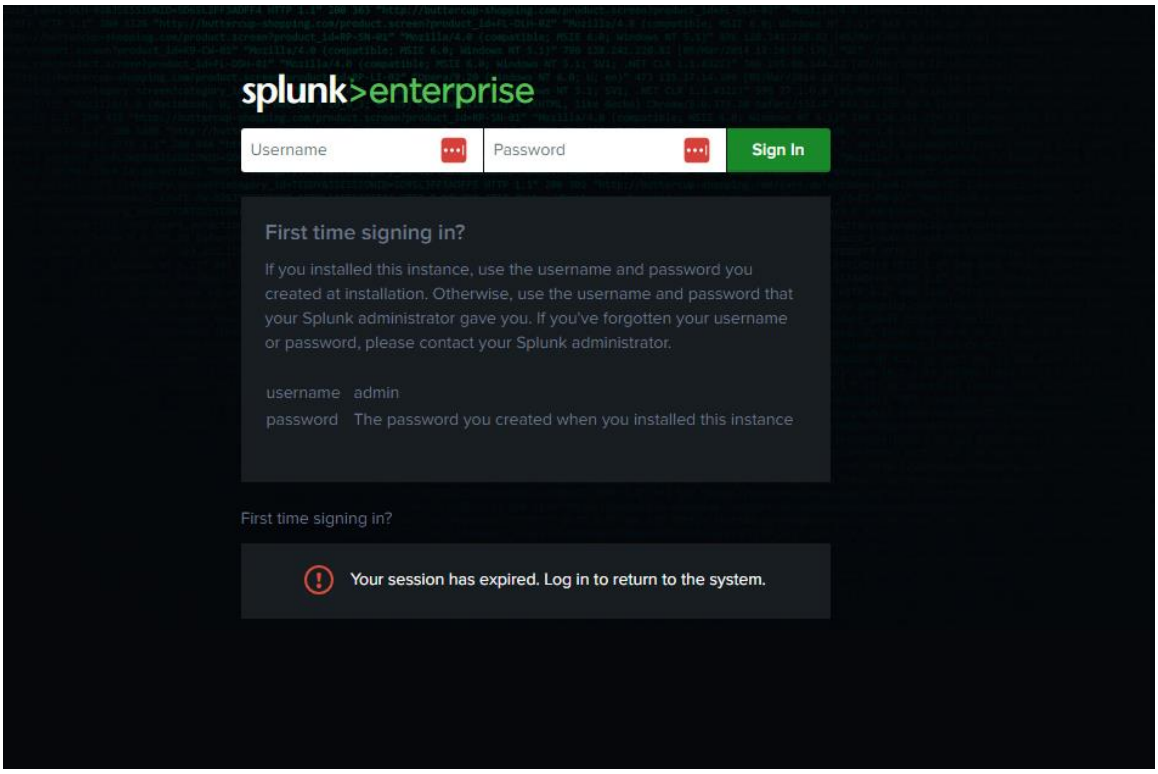
This is exactly what I expected.  Since the AIO host was already created as well as the IP address and firewall rules, two new resources added so I will run terraform apply.

```
> terraform apply
...
Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + splunk_core_ds_ip  = (known after apply)
```

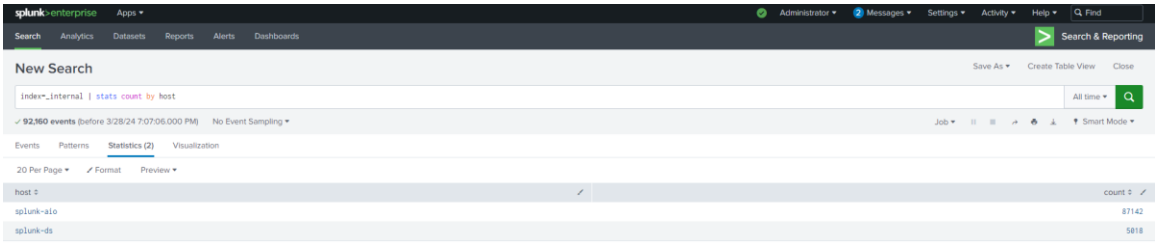
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

google_compute_address.splunk_core_address_2: Creating...
google_compute_address.splunk_core_address_2: Still creating... [10s elapsed]
google_compute_address.splunk_core_address_2: Creation complete after 12s
[id=projects/terraform-demo-2024/regions/us-central1/addresses/splunk-core-
address-2]
google_compute_instance.splunk_core_instance_2: Creating...
google_compute_instance.splunk_core_instance_2: Still creating... [10s elapsed]
google_compute_instance.splunk_core_instance_2: Creation complete after 13s
[id=projects/terraform-demo-2024/zones/us-central1-a/instances/splunk-ds]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

splunk_core_aio_ip = "34.123.210.235"
splunk_core_ds_ip = "34.41.154.142"
```

## Log into the newly created Splunk deployment server

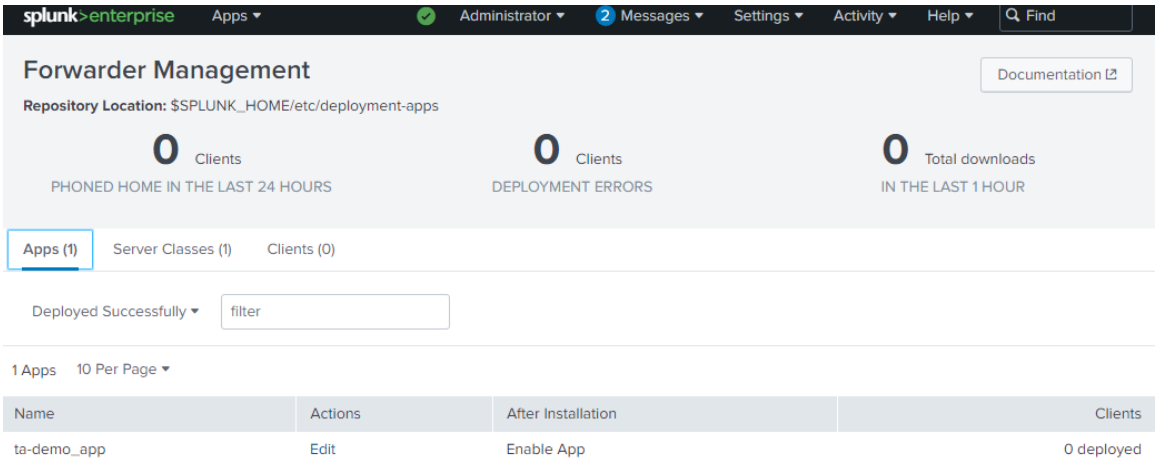
Similar to on the AIO host, to confirm everything worked correctly, I SSH into the host and review the install script log for errors and then log into the host on the Splunk UI to confirm that it is running.



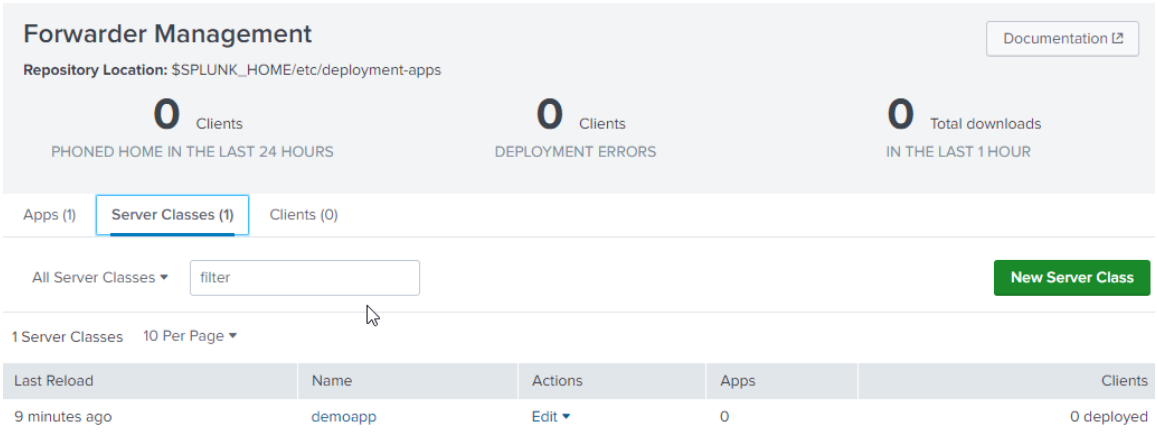

The Splunk UI on the DS loads successfully. When I look at the _internal index on the Splunk-AIO host, I can see that we are receiving logs from both hosts.



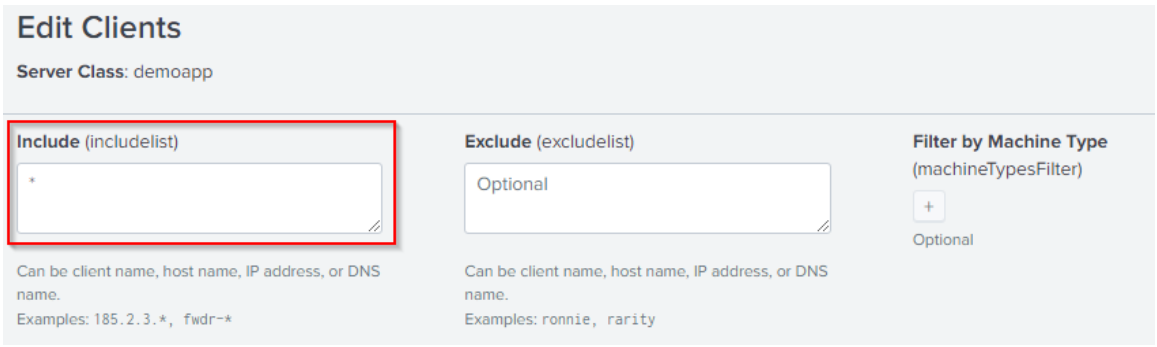

When i open the Forwarder Management page, I see one DS Managed app, TA-demo_app



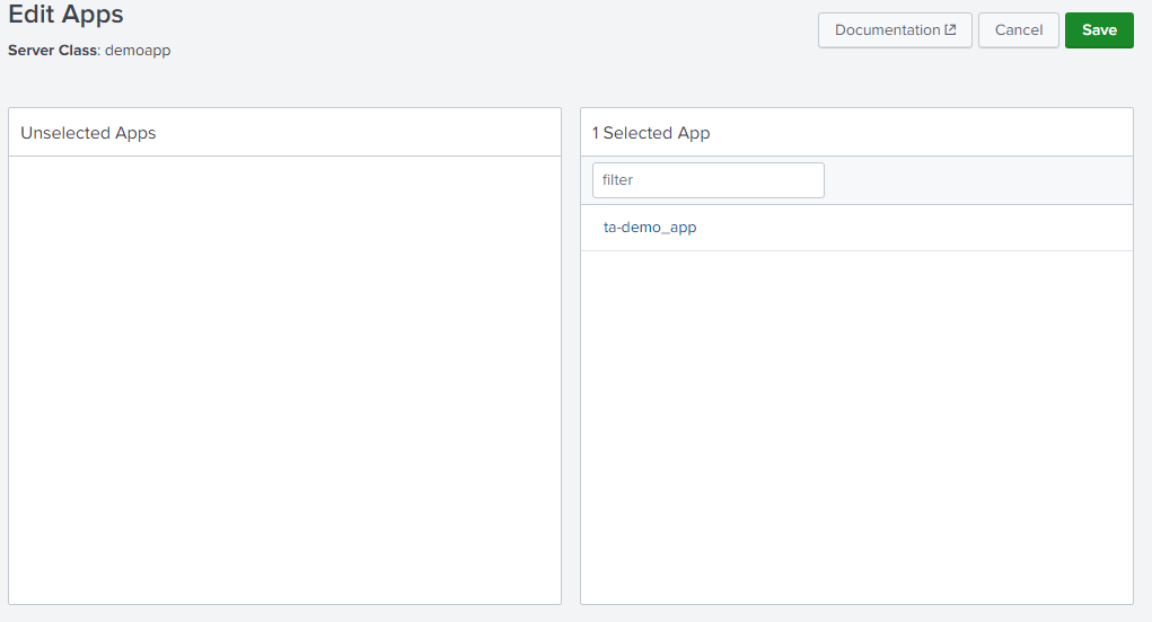

And the single server class defined in serverclass.conf: demo




Which matches on all hosts (*)



And adds the TA-demo_app application to each host.




Thanks for reading.  Check out the final part of this whitepaper, where we will install some universal forwarders, join them to the deployment server and review their output to finish off this testing cycle.

- Part 1 - Introduction to Terraform
- Part 2 - Deploy, install and configure the Splunk All-in-One host
- Part 3 - Deploy, install and configure the Splunk Deployment Server
- Part 4 - Install a series of Universal Forwarders and run through a test cycle

# Part 4: Installing Universal Forwarders

In Part one, we learned some basics about Terraform, and we created, changed and then subsequently destroyed a VM using terraform. In part two, we learned how to add scripting to prepare and install Splunk for the all-in-one host. Then in part three we installed and configured a Deployment Server. In this last part, we are going to install the Splunk Universal Forwarder on 4 different OS's and have them join the deployment server, install the TA-demo_app and df_all_forwarderoutputs apps, and then review the data returned from the TA on the Splunk Enterprise indexes.

## Creating a forwarder install script

Similar to the previous entries, let's start with the script that will prep the hosts and install the Splunk Universal Forwarder on each.

I wanted this version of the script to be able to handle both Debian and Redhat flavours of Linux, and to be able to install any available version of the Splunk Universal Forwarder, so the script has been designed to take command line arguments specifying the linux flavour (-f), the download url for the Splunk Universal Forwarder (-u), and the version of Splunk that is being installed (-v).

The other notable difference between the Splunk Enterprise installation scripts is the deploymentclient.conf, which is pointing each UF to the Deployment Server so it will check into the DS for Forwarder Management.

Below is the script which i have saved into ./resources/build_splunk_uf.sh:

```
#!/bin/bash

# --- Logging Configuration ---
# Define a log file for script output
log_file="/tmp/build_splunk_uf.log"


# Function to log messages with timestamp
function log_message() {
  local message="$1"
  echo "$(date +'%Y-%m-%d %H:%M:%S') - $message" >> "$log_file" 2>&1
```

```
}

# --- Script Execution Check ---
# Check if a flag file exists to indicate previous execution
if [[ -f /tmp/installscriptrun ]]; then
  log_message "Script has already been run. Exiting."
  exit 0
fi

# Create the flag file to mark script execution
touch /tmp/installscriptrun

# Retrieve command line arguments
while getopts ":f:u:v:" opt; do
  case $opt in
    f) FLAVOUR="$OPTARG"
    ;;
    u) URL="$OPTARG"
    ;;
    v) VERSION="$OPTARG"
    ;;
    \?) echo "Invalid option -$OPTARG" >&2
    exit 1
    ;;
  esac
done
log_message "Building version [$VERSION] on [$FLAVOUR] from [$URL]"

# Installing needed apps
if [[ $FLAVOUR == "rh" ]]; then
  if ! sudo yum -y install wget;  then
    log_message "Error installing wget."
    exit 1
  fi
  if ! sudo yum -y install bind-utils; then
    log_message "Error installing bind-utils."
    exit 1
  fi
elif [[ $FLAVOUR == "deb" ]]; then
  if ! sudo apt-get -y install wget; then
    log_message "Error installing wget."
    exit 1
  fi
  if ! sudo apt-get -y install dnsutils; then
    log_message "Error installing dnsutils."
    exit 1
  fi
fi

# --- Increase ulimits for Splunk User ---
# log_message "Setting ulimits for Splunk user..."
echo "splunk hard nofile 8192" | sudo tee -a /etc/security/limits.conf
echo "splunk soft nofile 8192" | sudo tee -a /etc/security/limits.conf
```

```
echo "splunk hard nproc 4096" | sudo tee -a /etc/security/limits.conf
echo "splunk soft nproc 4096" | sudo tee -a /etc/security/limits.conf
echo "splunk hard stack 8388608" | sudo tee -a /etc/security/limits.conf
echo "splunk soft stack 8388608" | sudo tee -a /etc/security/limits.conf

# Download the latest Splunk Enterprise tarball using wget
cd /tmp
FILENAME="${URL##*/}"
if ! sudo wget -O $FILENAME "$URL"; then
  log_message "Error downloading Splunk tarball."
  exit 1
fi

# Untar the tarball to /opt
if ! sudo tar -xzvf /tmp/$FILENAME -C /opt; then
  log_message "Error extracting Splunk tarball."
  exit 1
fi


# --- Splunk User and Directory Setup ---
# Create Splunk user and group
log_message "Creating Splunk user and group..."
sudo useradd -r -m -d /opt/splunkforwarder -s /bin/bash -U splunk


# --- Ownership and Permissions ---
# Change ownership of Splunk directory to splunk user
log_message "Setting ownership of Splunk directory..."
sudo chown -R splunk:splunk /opt/splunkforwarder


# --- Splunk Configuration ---
# Create necessary folders for Splunk configuration
log_message "Creating Splunk configuration folders..."
su - splunk -c 'mkdir -p /opt/splunkforwarder/etc/apps/df_testenvironment_uf/local'
su - splunk -c 'mkdir -p /opt/splunkforwarder/etc/apps/df_testenvironment_uf/metadata'

# Create configuration files (using heredoc syntax for clarity)
log_message "Creating Splunk UF configuration files..."

cat <<EOT >> /opt/splunkforwarder/etc/apps/df_testenvironment_uf/metadata/local.meta
[]
access = read : [ * ], write : [ admin ]
export = system
EOT

cat <<EOT >>
/opt/splunkforwarder/etc/apps/df_testenvironment_uf/local/deploymentclient.conf
[deployment-client]
disabled = false
clientName = TestUF--$FLAVOUR-$VERSION
phoneHomeIntervalInSecs = 60
```

```
[target-broker:deploymentServer]
targetUri = https://splunk-ds.c.terraform-demo-2024.internal:8089
EOT

cat <<EOT >> /opt/splunkforwarder/etc/apps/df_testenvironment_uf/local/outputs.conf
[tcpout]
defaultGroup = aio

[tcpout:aio]
server = splunk-aio.c.terraform-demo-2024.internal:9997
EOT

cat <<EOT >> /opt/splunkforwarder/etc/system/local/user-seed.conf
[user_info]
USERNAME = admin
PASSWORD = 1SuperSecretadmiNpassworD
EOT


# Set ownership of configuration files to splunk user
log_message "Setting ownership of Splunk configuration files..."
sudo chown -R splunk:splunk /opt/splunkforwarder/etc


# --- Splunk Startup and Service Management ---
# Start Splunk as the splunk user
log_message "Starting Splunk..."
su - splunk -c '/opt/splunkforwarder/bin/splunk start --answer-yes --accept-license'

# Enable Splunk to start automatically at boot
log_message "Enabling Splunk boot-start..."
su - splunk -c '/opt/splunkforwarder/bin/splunk stop'
sudo /opt/splunkforwarder/bin/splunk enable boot-start -systemd-managed 1 -user splunk
sudo chown -R splunk:splunk /opt/splunkforwarder
sudo systemctl start SplunkForwarder

# Indicate successful completion
log_message "Splunk UF is installed and started."
```

---

## Using the count meta-argument

There are three new Terraform concepts that we will need to use to be able to define the UFs in such a way that we don't have to create a separate resource block for each of the UFs: count, variables and provisioners.

First let's look at count.

In Terraform, the count meta-argument allows you to create multiple resources using a single resource block. It helps in defining the number of instances of a resource that should be created. By using count, you can efficiently manage the creation and deletion of resources without duplicating the resource definition.

So where you could have a main.tf script that looks like this:

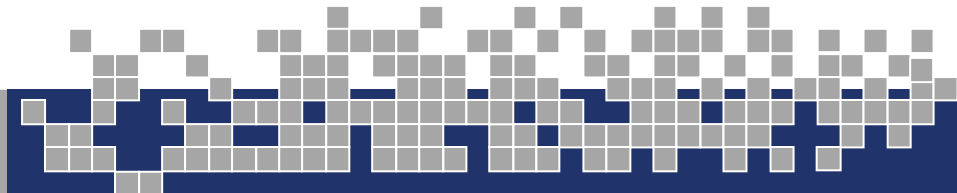
```
resource "google_compute_instance" "instance1" {
  name         = "instance-1"
  machine_type = "n1-standard-1"
  zone         = "us-central1-a"

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

  network_interface {
    network = "default"
  }
}

resource "google_compute_instance" "instance2" {
  name         = "instance-2"
  machine_type = "n1-standard-1"
  zone         = "us-central1-a"

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

  network_interface {
    network = "default"
  }
}

resource "google_compute_instance" "instance3" {
  name         = "instance-3"
  machine_type = "n1-standard-1"
  zone         = "us-central1-a"

  boot_disk {
    initialize_params {
```

```
      image = "debian-cloud/debian-9"
    }
  }

  network_interface {
    network = "default"
  }
}

resource "google_compute_instance" "instance4" {
  name         = "instance-4"
  machine_type = "n1-standard-1"
  zone         = "us-central1-a"

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

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

With count, you can achieve the same output using a single resource block.

```
resource "google_compute_instance" "instances" {
  count        = 4
  name         = "instance-${count.index}"
  machine_type = "n1-standard-1"
  zone         = "us-central1-a"

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

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

Note: in the example above, the name field uses ${count.index} to generate unique names like instance-0, instance-1, etc. count.index is a special variable that is available when using count on a resource. The count.index starts at 0 and increments by 1 for each instance created. If you wanted to start at 1, you could use ${count.index + 1}



So, to start our configuration for the Splunk UFs, we are going to create a resource block of type google_compute_instance with an identifier of splunk_ufs, we are setting the project, zone, machine_type and boot disk parameters, all similar to what we have done before. I am intentionally not putting in the startup script or network blocks at this point, more about those later.

```
resource "google_compute_instance" "instances" {
    count        = 5

    project      = "terraform-demo-2024"
    name         = "uf-${count.index}"
    machine_type = "e2-small"
    zone         = "us-central1-a"
    boot_disk {
        initialize_params {
            size  = 20
            type  = var.type
            image = var.uf_image[count.index]
        }
    }
    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    tags = [ "splunk", "splunk-uf", "env-test" ]
}
```

## Using Variables to parameterize infrastructure

Terraform also has the concept of variables. In Terraform, **variables** are a way to parameterize and configure your infrastructure. They allow you to abstract hard-coded values and make your configurations more flexible, reusable, and manageable. Variables can hold various types of data.

The variable types supported in Terraform are:

**string**: a sequence of characters representing some text, like "hello".

**number**: a numeric value. The number type can represent both whole numbers like 15 and fractional values like 6.283185.

**bool**: a boolean value, either true or false.

**list**: a sequence of values, like ["us-west-1a", "us-west-1c"]. Identify elements in a list with consecutive whole numbers, starting with zero ie zone[0] = "us-west-1a".

**map**: a group of values identified by named labels, like {name = "John Smith", age = 20}.

There are three types of Variables:

**Input Variables**: Defined by the user and passed into Terraform configurations. They are declared using the variable block.

**Local Variables**: Used to assign intermediate values within a configuration. They are declared using a locals block.

**Output Variables**: Used to display information about your infrastructure after an apply operation. They are declared using the output block.

We are going to create a new file: variables.tf, in the same directory as the main.tf file. The variables.tf file will have one or more variable blocks defining the variable, its data type and optionally a default value for that variable and description of the variable's purpose.

```
variable "variable_name" {
    description = "A description of the variable's purpose"
    type        = string
    default     = "default_value"
}
```

Let's start creating variables for the Universal Forwarders.

I am creating a variable to identify which OS image to install on each of the forwarders. This is a list type variable. When assigning values to a list variable a comma separated list of values in square brackets is used (ie: ["item1", "item2", "itemN"] )

```
variable "uf_image" {
    Description = "A list of images to install on each of the Universal
Forwarder hosts"
```

```
    type        = list
    default     = [
        "rhel-cloud/rhel-8-v20240611",
        "rhel-cloud/rhel-9-v20240611",
        "debian-cloud/debian-12-bookworm-v20240617",
        "ubuntu-os-cloud/ubuntu-2204-lts"
    ]
}
```

And a second list to identify what flavour of Linux (Redhat or Debian) each of these represents:

```
variable "uf_flavour" {
    description = "A list of the linux flavours on each of the Universal Forwarder
Hosts"
    type        = list
    default     = [
        "rh",
        "rh",
        "deb",
        "deb"
    ]
}
```

Another to set the Splunk UF Version to install.  This variable will be a string as we are only going to install one version in each round of testing.

```
variable "splunk_version" {
    description = "Version of Splunk universal Forwarder to install"
    type        = string
    default     = "9.2.1"
}
```

Then create a map variable to point to the download URL for each version of the Splunk forwarder, this way if we change the splunk_version variable from 9.2.1 to 9.0.0, the script will pull the url for the updated version.

```
variable "splunk_download_link" {
    type    = map
    default = {
        "9.2.1" =
"https://download.splunk.com/products/universalforwarder/releases/9.2.1/linux/s
plunkforwarder-9.2.1-78803f08aabb-Linux-x86_64.tgz",
        "9.1.4" =
"https://download.splunk.com/products/universalforwarder/releases/9.1.4/linux/s
plunkforwarder-9.1.4-a414fc70250e-Linux-x86_64.tgz",
```

```
        "9.0.9" =
"https://download.splunk.com/products/universalforwarder/releases/9.0.9/linux/s
plunkforwarder-9.0.9-6315942c563f-Linux-x86_64.tgz",
        "8.2.9" =
"https://download.splunk.com/products/universalforwarder/releases/8.2.9/linux/s
plunkforwarder-8.2.9-4a20fb65aa78-Linux-x86_64.tgz",
        "8.1.9" =
"https://download.splunk.com/products/universalforwarder/releases/8.1.9/linux/s
plunkforwarder-8.1.9-a16db3287b56-Linux-x86_64.tgz",
        "8.0.9" =
"https://download.splunk.com/products/universalforwarder/releases/8.0.9/linux/s
plunkforwarder-8.0.9-153839c8b72f-Linux-x86_64.tgz"
    }
}
```

And while we're at it, let's create variables for all the repeated values used throughout the main.tf file.

```
variable "zone" {
    description =
    type        = string
    default     = "us-central1-a"
}

variable "region" {
    description =
    type        = string
    default     = "us-central1"
}

variable "type" {
    description =
    type        = string
    default     = "pd-balanced"
}

variable "core_image" {
    description =
    type        = string
    default     = "ubuntu-os-cloud/ubuntu-2204-lts"
}

variable "network" {
    description =
    type        = string
    default     = "default"
}

variable "project" {
    description =
    type        = string
```

```
    default     = "terraform-demo-2024"
}

variable "uf_machine_type" {
    description =
    type        = string
    default     = "e2-small"
}
```

Now that all the variables are defined, let's start replacing the hard coded values in the main.tf file with the appropriate variables.

The use of variables in the main.tf file depends on the variable type.

| | |
|---|---|
| String variables | var.variablename |
| List Variables | var.variablename[index] |
| Map Variables | var.variablename{label} |

Here is what the new splunk_ufs resource block looks like with the variables in place

```
resource "google_compute_instance" "splunk_ufs" {
    project      = var.project
    name         = "uf-${count.index}"

    machine_type = var.uf_machine_type
    zone         = var.zone
    count        = 4

    boot_disk {
        initialize_params {
            size  = 20
            type  = var.type
            image = var.uf_image[count.index]
        }
    }

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa_zscaler.pub")}"
    }

    tags = [ "splunk", "splunk-ufs", "env-test" ]

}
```

Similar to the Splunk core instance, the network ip is being defined in a google_compute_address block, which will have the same count as the splunk_ufs block..

```
resource "google_compute_address" "splunk_uf_addresses" {
    count        = 4
    name         = "splunk-uf-address-${count.index}"
    project      = var.project
    region       = var.region
    address_type = "EXTERNAL"
}
```

And to refer to that, let's build the network interface block, note that the nat_ip points back to the address block we just created using the same count.index so host uf-0 will be assigned splunk_uf_addresses[0], uf-1 assigned splunk_uf_addresses[1], etc.

```
    network_interface {
        network = var.network

        access_config {
            network_tier = "PREMIUM"
            nat_ip =
google_compute_address.splunk_uf_addresses[count.index].address
        }
    }
```

So here is the full set of the UF configs how they look now.

```
resource "google_compute_instance" "splunk_ufs" {
    project      = var.project
    name         = "uf-${count.index}"

    machine_type = "e2-small"
    zone         = var.zone
    count = 4

    boot_disk {
        initialize_params {
            size  = 20
            type  = var.type
            image = var.uf_image[count.index]
        }
    }

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }
```

```
    network_interface {
        network = var.network

        access_config {
            network_tier = "PREMIUM"
            nat_ip =
google_compute_address.splunk_uf_addresses[count.index].address
        }
    }

    tags = [ "splunk", "splunk-ufs", "env-test" ]
}

resource "google_compute_address" "splunk_uf_addresses" {
    count        = 4
    name         = "splunk-uf-address-${count.index}"
    project      = var.project
    region       = var.region
    address_type = "EXTERNAL"
}
```

## Using provisioners to execute actions

The last concept this post is going to add is provisioners. Provisioners are a mechanism in Terraform to execute actions after a resource has been created or destroyed. They can be used to:

**Transfer files:** The file provisioner copies files from the local machine to the remote resource.

**Execute commands locally**: The local-exec provisioner runs a command on the machine where Terraform is being executed.

**Execute commands remotely**: The remote-exec provisioner runs commands on the remote resource itself (e.g., an EC2 instance) using SSH or WinRM.

The reason we need provisioners in this use case is that the metadata_startup_script that we used to execute the splunk installation scripts on the Splunk Enterprise instances does not have a mechanism for executing the script with command line arguments (which we need for the UF install script).

For the UF installation use case, we will create a connection block, which defines a connection to the host, a file provisioner to copy the script to the host and then a remote-exec provisioner to execute the script with the command line arguments.

The connection block provides essential information about how Terraform should connect to the remote resource. This includes details like:

- **Host**: The IP address or hostname of the remote machine.
- **User**: The username to use for authentication.
- **Authentication method**: SSH key, password, or other methods.

For this example, we point to the NAT IP address of the first access configuration on the first network interface of the current resource, connect to the host using ssh, user darren, and use the private key at ~/.ssh/id_rsa for authentication.

```
connection {
    host        = self.network_interface[0].access_config.0.nat_ip
    type        = "ssh"
    user        = "darren"
    private_key = file("~/.ssh/id_rsa")
}
```

Next, we need to copy the installation script to the remote host. For this the file provisioner is used. All that is required to use a file provisioner is the path on the local machine where the file exists and the path on the remote host where the file is to be copied.

```
provisioner "file" {
    source      = "./resources/build_splunk_uf.sh"
    destination = "/tmp/build_splunk_uf.sh"
}
```

Finally, we need to use a remote-exec provisioner to set permissions on the script that we copied to executable and then to execute the script with the required command line arguments appropriate for this host. We are using the ${var.variablename[count.index]} to get the correct value for each variable.

```
provisioner "remote-exec" {
    inline = [
        "sudo chmod 550 /tmp/build_splunk_uf.sh",
        "sudo bash /tmp/build_splunk_uf.sh -f ${var.uf_flavour[count.index]} -u
${var.splunk_download_link[var.splunk_version]} -v ${var.splunk_version}"
    ]
```

```
}
```

## Putting it all together

Now the full main.tf script is complete. Here is the final main.tf script including all the resource blocks

```
provider "google" {
    project = var.project
    region  = var.region
}

resource "google_compute_instance" "splunk_core_instance_1" {
    project      = var.project
    name         = "splunk-aio"
    machine_type = "e2-standard-8"
    zone         = var.zone

    boot_disk {
        initialize_params {
            size  = 100
            type  = var.type
            image = var.core_image
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_core.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    network_interface {
        network = var.network

        access_config {
            network_tier = "PREMIUM"
            nat_ip = google_compute_address.splunk_core_address_1.address
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-core-aio", "env-test" ]
}


resource "google_compute_address" "splunk_core_address_1" {
    name         = "splunk-core-address-1"
    project      = var.project
    region       = var.region
    address_type = "EXTERNAL"
```

```
}


resource "google_compute_instance" "splunk_core_instance_2" {
    project      = var.project
    name         = "splunk-ds"
    machine_type = "e2-small"
    zone         = var.zone

    boot_disk {
        initialize_params {
            size  = 20
            type  = var.type
            image = var.core_image
        }
    }

    metadata_startup_script =
file("${path.module}/resources/build_splunk_ds.sh")

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    network_interface {
        network = var.network

        access_config {
            network_tier = "PREMIUM"
            nat_ip = google_compute_address.splunk_core_address_2.address
        }
    }

    tags = [ "splunk", "splunk-core", "splunk-ds", "env-test" ]
}


resource "google_compute_address" "splunk_core_address_2" {
    name         = "splunk-core-address-2"
    project      = var.project
    region       = var.region
    address_type = "EXTERNAL"
}


resource "google_compute_firewall" "splunk_core_all" {
    project = var.project
    name    = "splunk-core-all"
    network = var.network

    allow {
        protocol = "tcp"
        ports    = ["22", "8000", "8089"]
```

```
    }

    source_ranges = ["0.0.0.0/0"]

    target_tags = ["splunk-core"]
}


resource "google_compute_firewall" "splunk_core_aio" {
    project = var.project
    name    = "splunk-core-aio"
    network = var.network

    allow {
      protocol = "tcp"
      ports    = ["9997", "8088"]
    }

    source_ranges = ["0.0.0.0/0"]

    target_tags = ["splunk-core-aio"]
}


resource "google_compute_instance" "splunk_ufs" {
    project      = var.project
    name         = "uf-${count.index}"

    machine_type = "e2-small"
    zone         = var.zone
    count = 4

    boot_disk {
        initialize_params {
            size  = 20
            type  = var.type
            image = var.uf_image[count.index]
        }
    }

    metadata = {
        ssh-keys = "darren:${file("~/.ssh/id_rsa.pub")}"
    }

    network_interface {
        network = var.network

        access_config {
            network_tier = "PREMIUM"
            nat_ip =
google_compute_address.splunk_uf_addresses[count.index].address
        }
    }
```

```
    tags = [ "splunk", "splunk-ufs", "env-test" ]

    connection {
        host        = self.network_interface[0].access_config.0.nat_ip
        type        = "ssh"
        user        = "darren"
        private_key = file("~/.ssh/id_rsa")
    }

    provisioner "file" {
        source      = "./resources/build_splunk_uf.sh"
        destination = "/tmp/build_splunk_uf.sh"
    }

    provisioner "remote-exec" {
        inline = [
            "sudo chmod 770 /tmp/build_splunk_uf.sh",
            "sudo bash /tmp/build_splunk_uf.sh -f
${var.uf_flavour[count.index]} -u
${var.splunk_download_link[var.splunk_version]} -v ${var.splunk_version}"
        ]
    }
}

resource "google_compute_address" "splunk_uf_addresses" {
    count        = length(var.uf_flavour)
    name         = "splunk-uf-address-${count.index}"
    project      = var.project
    region       = var.region
    address_type = "EXTERNAL"
}
```

# Creating the full test environment

Now, let's run terraform apply to see the full test environment. For this test to be successful we want to see:

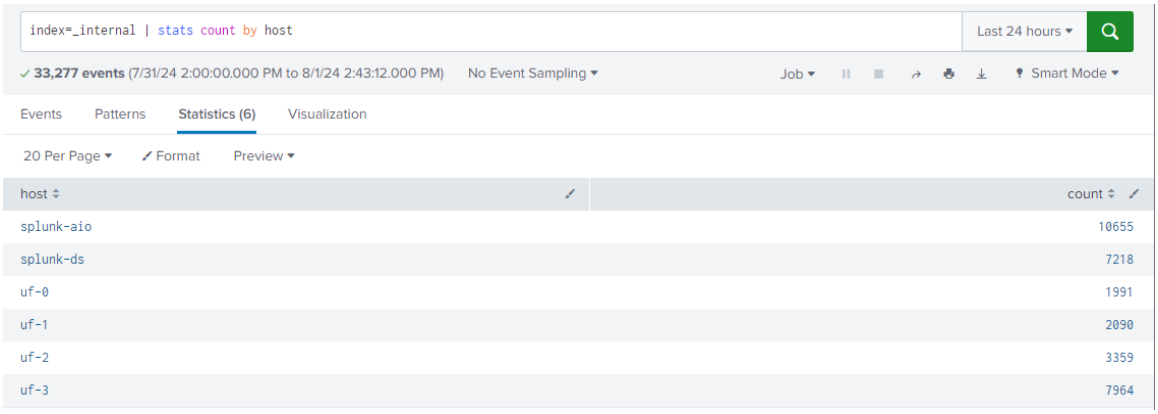
- Splunk AIO host built and ready
- Splunk DS host built and ready
- Splunk DS internal logs forwarding to the Splunk AIO
- 4 Splunk UF hosts built, each on different OSs
- 4 Splunk UF host internal logs forwarding to the Splunk AIO
- 4 Splunk UF hosts communicating with DS
- 4 Splunk UF hosts install the TA-demo-app application from the DS

```
> terraform apply

…

Plan: 14 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + splunk_core_aio_ip = (known after apply)
  + splunk_core_ds_ip  = (known after apply)
  + uf_instance_ips    = (known after apply)

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

  Enter a value:
```

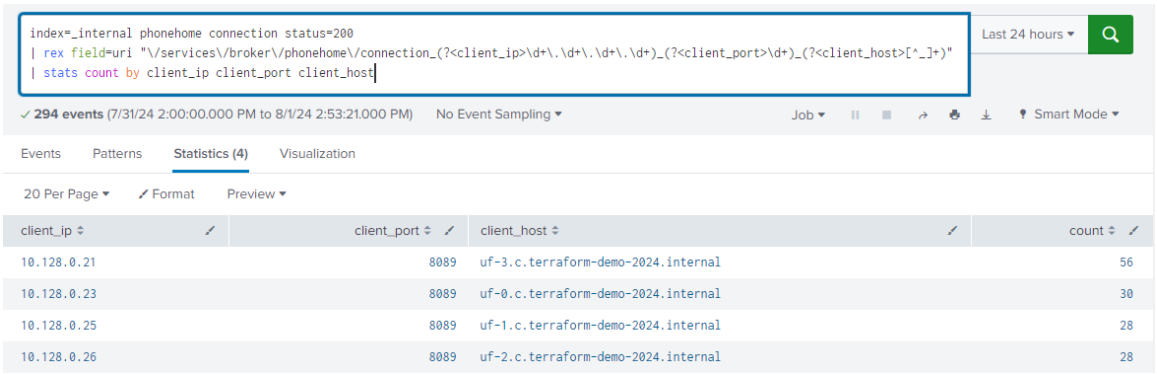
The plan shows 14 to add, this is 6 hosts (1x AIO, 1x DS, 4x UF), 6 addresses, and two firewall objects. This is correct.

```
Apply complete! Resources: 14 added, 0 changed, 0 destroyed.

Outputs:

splunk_core_aio_ip = "35.202.2.168"
splunk_core_ds_ip = "34.28.216.53"
uf_instance_ips = "34.67.185.116,34.134.121.190,34.41.220.64,35.184.192.153"
```

The full build took 8 minutes to run. Let's log into the AIO, run some searches and see if our tests are complete.




All 6 hosts are forwarding internal logs, let's check for deployment server communication:




All four UFs checked into the Deployment server, now let's check for the installation of the TA-demo_app


Success, all tests are passed.

# Wrap Up

In this whitepaper we provided you with an introduction to terraform and have successfully created Terraform code to build a fully automated Splunk test environment. Thanks for reading!

---

# Terraform Professional Services

Discovered Intelligence offers comprehensive professional service offerings to ensure your success with Terraform. Let us help you improve the speed of infrastructure deployment, enable agility and automate complex infrastructure through Infrastructure as Code (IaC) with Terraform.

Our specialized **Terraform Professional Services** can be viewed at:
https://DiscoveredIntelligence.com/terraform-professional-services

Our professional services offerings include:

- Terraform Implementation
- Infrastructure Migration Using Terraform
- Implementing Zero Trust Architectures
- Terraform Operational Assessment