TERRAFORM

Google Cloud

Terraform Using Google Cloud Platform

Course Introduction

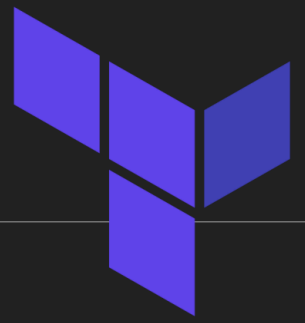
Udemy Tips

IAC, Terraform & Installation

Traditional IT

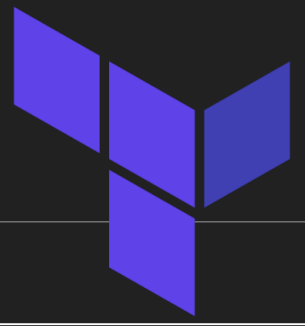
- How Application Dev Lifecycle works
- Business create requirement
- BA – Convert Requirement into Technical detail
- Cloud Architect/ Tech lead : Infrastructure design
- If more hardware require – contact procurement
- Buying new hardware in Datacenter may take weeks to months
- Infrastructure team – Provision hardware
- Dev Team start working on Application
- This flow has very slow App Deployment
- Expensive, Scaling is issue
- Lots of different team involved will lead to error.
- Need to overcome above issue – Public Cloud Provider is the solution : AWS, GCP
- In Cloud, Resource provisioning is very fast. From month to weeks
- Public cloud provider will manage everything for you.

Interaction with Cloud



- Cloud Console/Portal
  - Compared to Traditional Flow this is better
  - With few clicks, can provision VM in Cloud
  - But good enough if managing limited resource
- With Programmatic way – API
  - Python, java, Shell
  - Different Team write different script for resource provisioning
  - There is no unique approach
  - Different Organization try to solve same problem, but different way
  - There comes Some common unifying approach, language, tool for infrastructure creation inside Cloud
  - Docker, Puppet, Ansible, terraform, packer

IAC

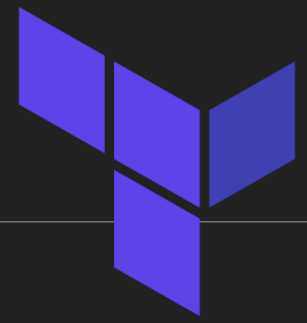


- Resource provisioning using Code
- Create Shell/Python script for creating VM
- But writing/maintaining such code is tedious task

```
Create N/W
Wait for above step to finish
Provision Subnet
Create Firewall rule
Wait for above step to finish
Compute engine instance with all parameter
```

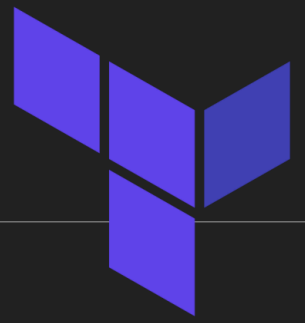
```
resource "google_compute_instance"
"first-instance"{
    name = "hello-1"
    zone = "us-central1-a"
    machine_type = "n1-standard-1"
    boot_disk {
        initialize_params {
            image = "debian-
cloud/debian-9"
        }
    }
    network_interface {
        network = "default"
    }
}
```

Terraform



- Terraform is the one of the most popular tool for Infrastructure provisioning
- Free – Open source
- Developed by HashiCorp
- Quick & easy to get started with single binary file
- Master HCL – terraform in short span of time
- Terraform has multiple provider are available.
- Apart from Public cloud, lots pf different other provider are available for network, DNS, Firewall, database
- Write configuration in HCL/JSON.
  - HCL is preferred.
- Terraform is agentless tool
- It is not configuration tool. Work well with Ansible.

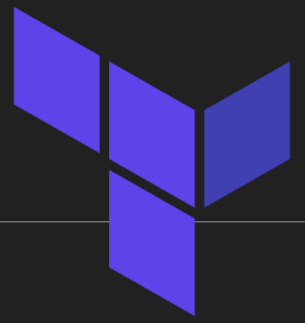
Terraform is idempotent



Python/Shell script	To create VM
	run 3 times
	3 resource will be created
	It will cost 3 times.

Terraform HCL script	To create VM
	run 3 times
	Only 1 resource will be created
	It will not cost 3 times.

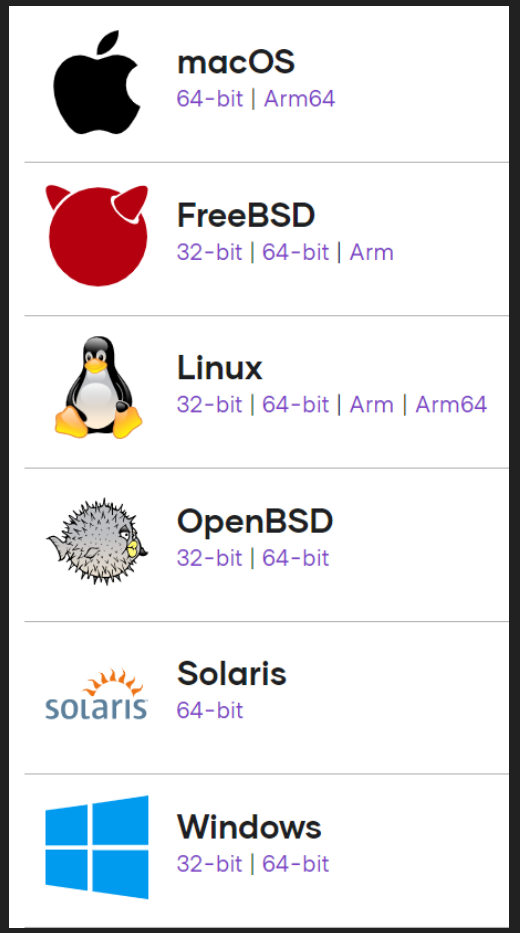
Native tool



- Cloud Native tool available for infrastructure provisioning
- Azure – Template
- Google – Deployment manager
- AWS - Cloud Formation
- JSON/Yaml
- Terraform is cloud agnostic.
- With Multiple provider, resource can be provisioned for multiple cloud.

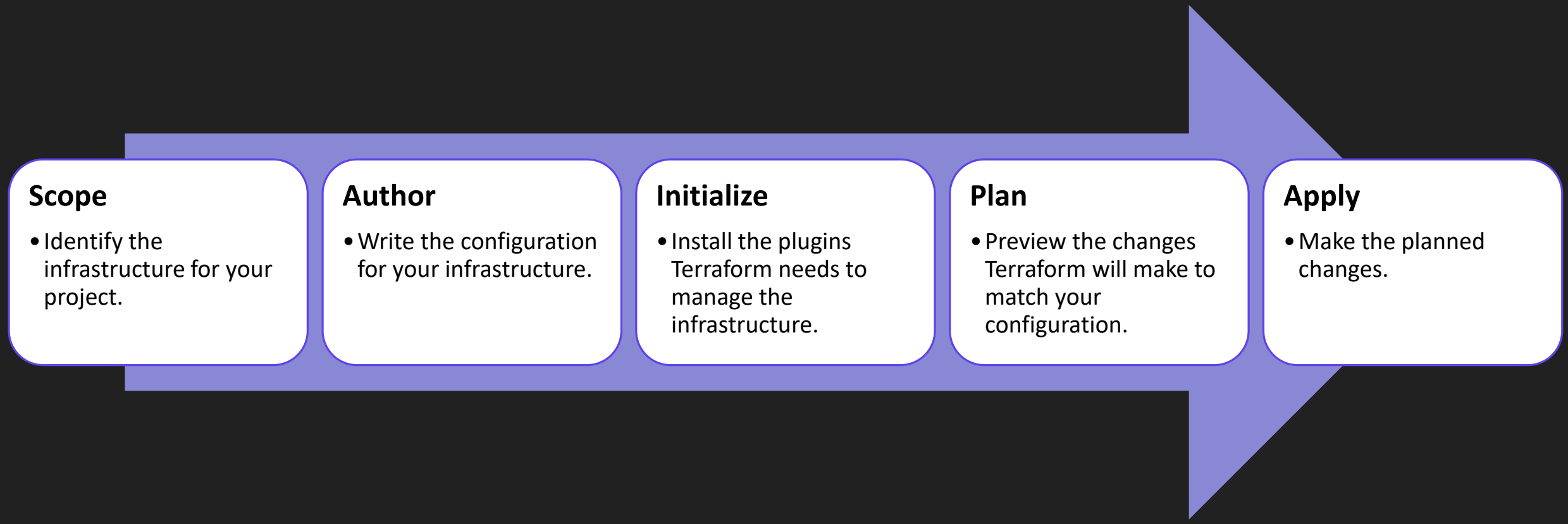
Terraform Installation

- Available for all major OS:
- Visit : https://www.terraform.io/downloads.html
- Download Binary
- Unzip it.
- Export Path variable
  - Windows – will see
  - export PATH=$PATH:path to terraform binary
- verify with terraform version
- Editor – Free to use any of your favorite editor
- Let's see in action

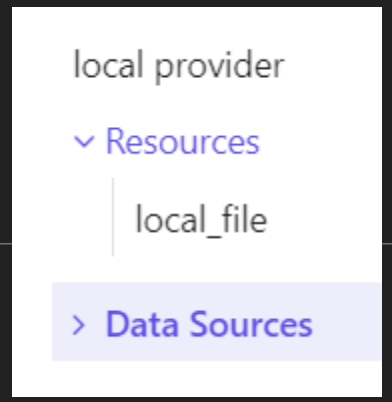


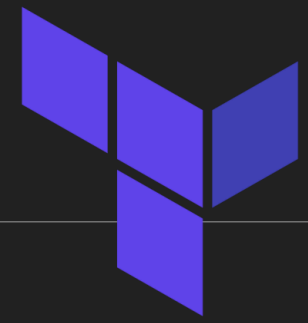
Terraform Basics - I

Terraform Workflow



Scope & Author





- Identify what resource need to provision
- Create Local File – sample.txt with some content
- Write configuration file for it in HCL language

.tf Extension

Block

Resource Type - Local

Resource Name

Arguments

main.tf

```
1  resource local_file sample_res {
2      filename = "sample.txt"
3      content = "I love Terraform"
4  }
```

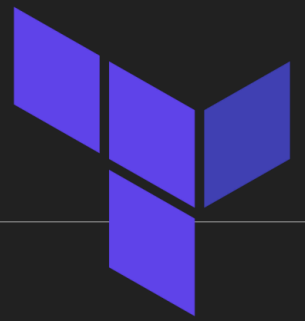
Create first Terraform File

Terraform
init, Plan & apply

init, Plan & apply

- init
  - first command after writing configuration files
  - initialize a working directory
  - Download plugin
    - local_file
    - random
- plan
  - Creates execution plan
  - Doesn’t change any infrastructure
- apply
  - execute all changes & provision resource specified in configuration files

local_file argument



Argument Reference

The following arguments are supported:

- `content` - (Optional) The content of file to create. Conflicts with `sensitive_content` and `content_base64` .
- `sensitive_content` - (Optional) The content of file to create. Will not be displayed in diffs. Conflicts with `content` and `content_base64` .
- `content_base64` - (Optional) The base64 encoded content of the file to create. Use this when dealing with binary data. Conflicts with `content` and `sensitive_content` .
- `filename` - (Required) The path of the file to create.
- `file_permission` - (Optional) The permission to set for the created file. Expects a string. The default value is `"0777"` .
- `directory_permission` - (Optional) The permission to set for any directories created. Expects a string. The default value is `"0777"` .

Multiple Resource

```
resource local_file cat_res {
  filename = "cat.txt"
  content = "I love cat"
}
```

cat.tf

```
resource local_file dog_res {
  filename = “dog.txt"
  content = "I love dogs"
}
```

dog.tf

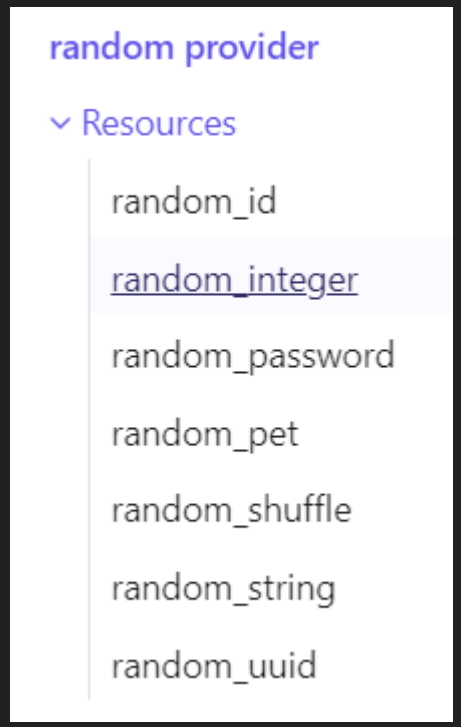
Main.tf

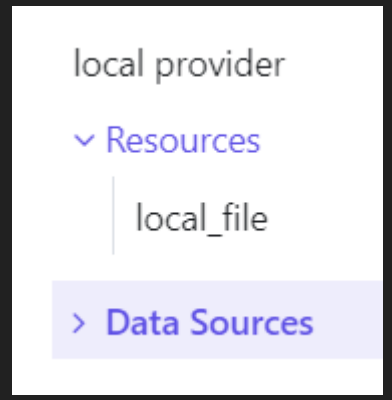
```
resource local_file cat_res {
  filename = "cat.txt"
  content = "I love cat"
}

resource local_file dog_res {
  filename = "dog.txt"
  content = "I love dogs"
}
```

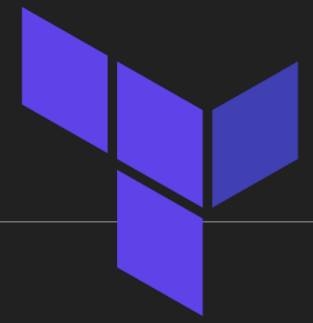
Random Provider

- The "random" provider allows the use of randomness within Terraform configurations.
- This is a *logical provider*, which means that it works entirely within Terraform's logic, and doesn't interact with any other services.
- Let's see in action





Variables



Main.tf

```
resource local_file sample_res {
  filename = “sample.txt"
  content = "I love Terraform"
}
```

```
resource local_file sample_res {
  filename = var.filename
  content = var.content
}
```

variables.tf

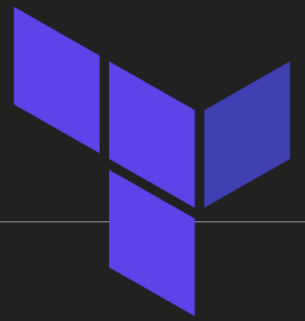
```
variable filename {
  type          = string
  default       = "sample.txt"
}

variable content {
  type          = string
  default       = "I Love Terraform"
}
```

Types of variables

- string – “cat”
- number – 234, 6.5
- bool - true/false
- list – sequence of value
  - list(string) =>[“red”, “green”, “blue”]
- Tuple – group non homogeneous data type
  - tuple([string, number, bool]) => [“dog”, 23, true]
- map – like key value : Dictionary
  - {name = “Ankit", age = 32}
- set – only unique values
- object – complex data type

Use Variables



```
variable filename {
  type        = string
  default     = "sample.txt"
}
```

var.filename

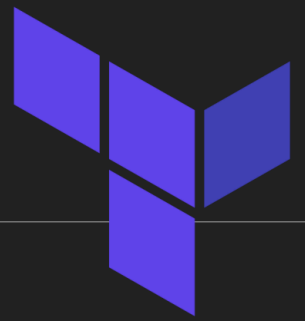
```
variable filename {}
```

Terraform apply will ask

```
terraform apply -var "filename=sample.txt"
```

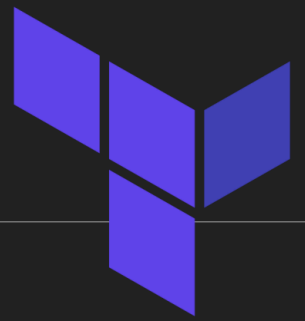
```
export TF_VAR_filename=sample.txt"
```

Variable Definition File



- terraform.tfvars
- terraform.tfvars.json
- *.auto.tfvars
- *.auto.tfvars.json

Which Variable will load first



1. `export TF_VAR_filename=sample.txt“`
2. terraform.tfvars file
3. variable.auto.tfvars file
4. terraform apply -var "filename=sample.txt”

Multiple Provider

main.tf

```
resource "local_file" "rand_res" {
  filename = “sample.txt"
  content  = "I love terraform”
}
```

```
resource "random_string" "rand_name" {
  length = 20
}
```

Implicit Dependency

```
resource "local_file" "rand_res" {
  filename = "implicit.txt"
  content  = "I love terraform "
}
```

```
resource "random_string" "rand_name" {
  length = 20
}
```

```
resource "local_file" "rand_res" {
  filename = "implicit.txt"
  content  = "I love random text ${random_string.rand_name.id}"
}
```

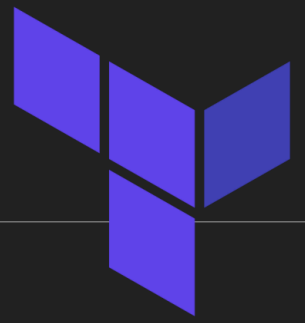
Explicit Dependency

```
resource "local_file" "rand_res" {
  filename = “explicit.txt"
  content  = "I love terraform "
}
```

```
resource "random_string" "rand_name" {
  length = 20
}
```

```
resource "local_file" "rand_res" {
  filename = "implicit.txt"
  content  = "I love random text ${random_string.rand_name.id}“
  depend_on = [random_string.rand_name]
}
```

Output

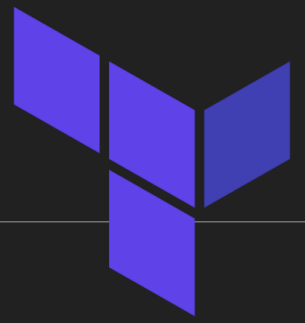


```
resource "random_string" "rand_name" {
  length = 20
}
```

```
output name {
  value = random_string.rand_name.id
}
```

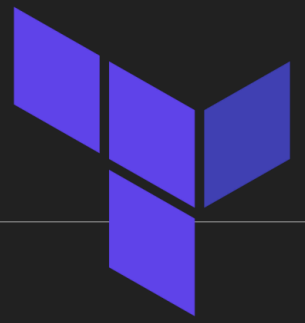
output.tf

Lifecycle Rules



- lifecycle – resource attributes
- create_before_destroy - Create the resource first and then destroy older
- prevent_destroy - Prevents destroy of a resource
- ignore_changes - Ignore Changes to Resource – Specific tag or all

Provider version



```
terraform {
  required_providers {
    local = {
      source = "hashicorp/local"
      version = "2.1.0"
    }
  }
}

provider "local" {
  # Configuration options
}
```

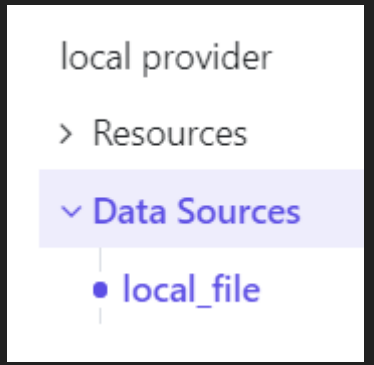
https://www.terraform.io/docs/language/expressions/version-constraints.html

Data source

- local_file reads a file from the local filesystem.

```
data "local_file" "foo" {
    filename = "sample1.txt"
}

output name1 {
  value = data.local_file.foo.content
}
```



Terraform + GCP

Setup GCP Project

SERVICE ACCOUNT

Google Provider

- Terraform has multiple provider to intract with different public cloud
- Infrastructure provision inside GCP from Terraform
- https://registry.terraform.io/providers/hashicorp/google/latest/docs

```
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "3.84.0"
    }
  }
}
provider "google" {
  project, region, zone
}
```

Connect with GCP

- Google Provider Configuration
  - Projectid, zone, region
- Multiple ways to authenticate with GCP
  1. Username/Password - gcloud auth application-default login
  2. Google Cloud VM
  3. Service Account – Keys : Preferred in Production
- Create Google Cloud Storage Bucket.

```
resource "google_storage_bucket" "gcs1"{
    name = "bucketname"
}
```

Approach to Provision resource

1. What this resource do
2. Cloud Console resource provisioning
3. Terraform script with minimum attributes (all required)
4. Add more arguments

Google Cloud Storage

- Object storage solution in GCP
- Unstructured Data storage
  - Image
  - Video
  - Binary File, etc…
- Cloud storage can be used for long term archival storage
- Can be access object over http, Rest API
- Let’s see in action

GCS + Terraform

Google Cloud Network

- No Network – No Cloud
- To create any resource, Network is must
- VPC – Virtual Private Network
- VPC contains subnets – Logical grouping of IP in single region
- 3 Types of VPC
  - Default VPC
  - Auto Mode
  - Custom Mode
- Let’s see in action –
  - How to create VPC
  - Create Subnet
  - Create firewall Policy

Network + Terraform

GCE + Terraform

Cloud Run + Terraform

Cloud Function + Terraform

BigQuery + Terraform

PubSub + Terraform

Spanner + Terraform

Cloud SQL + Terraform

BigTable + Terraform

MemoryStore + Terraform

THANK YOU