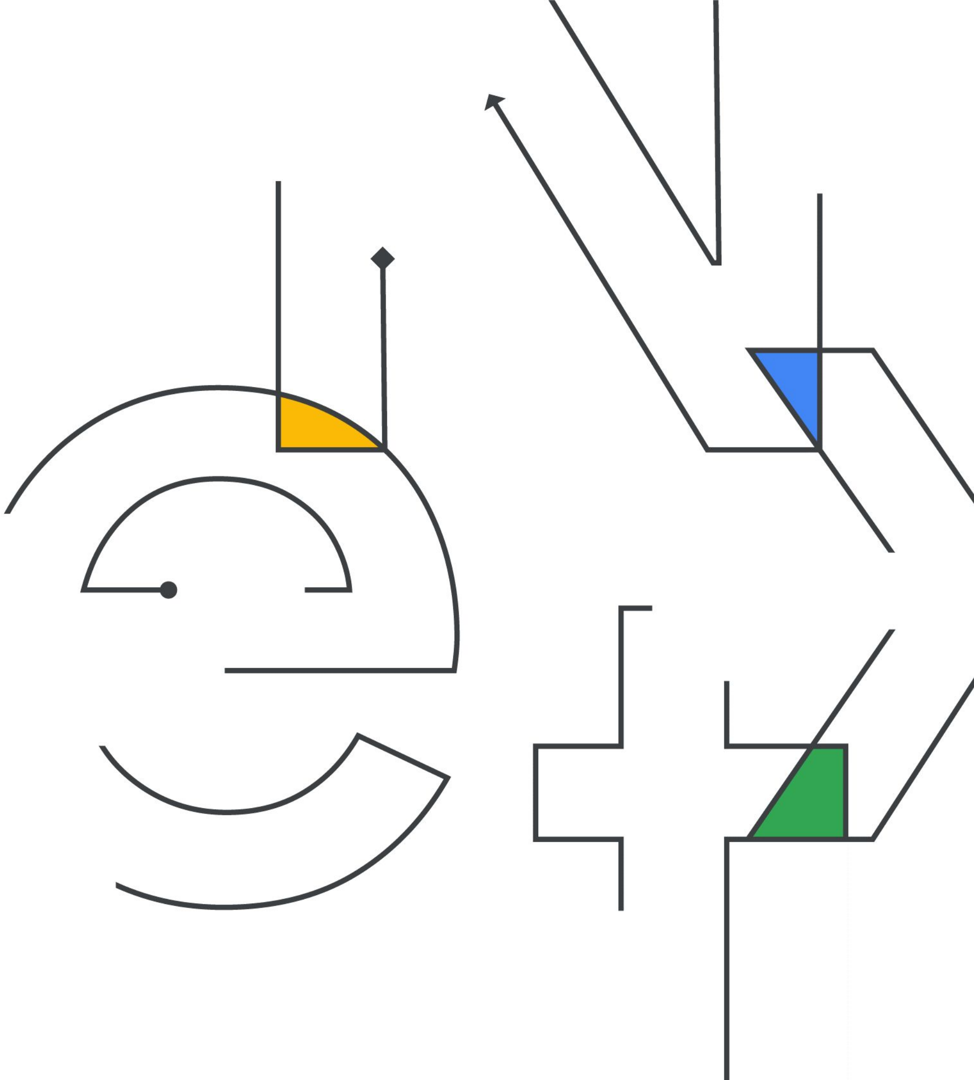
Google Cloud

Next '22

Better Hardware Provisioning for ML Experiments on GCP

Provision hardware for ML reliably.

Oct/
11–13



Sayak Paul

ML Engineer at Carted

October 13, 2022

Google Cloud

I’m presenting work done by my colleagues (Nilabhra, Shane, Sam) and myself.

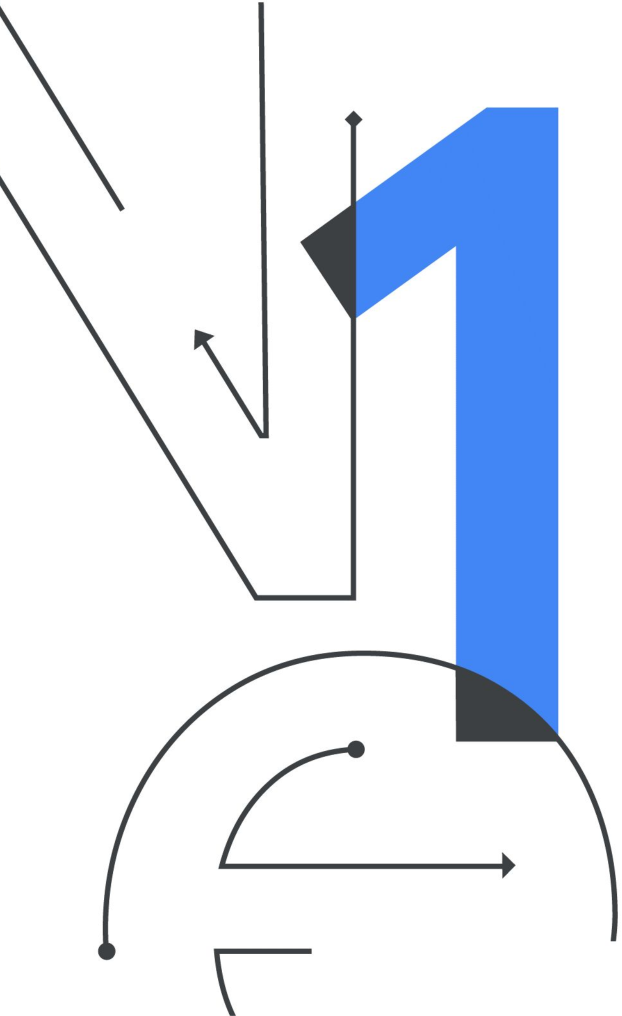
Contents

01 Typical workflow for ML hardware provisioning

02 Problems with the typical workflow

03 Streamlining provisioning workflow with Terraform

04 Demo



The standard workflow

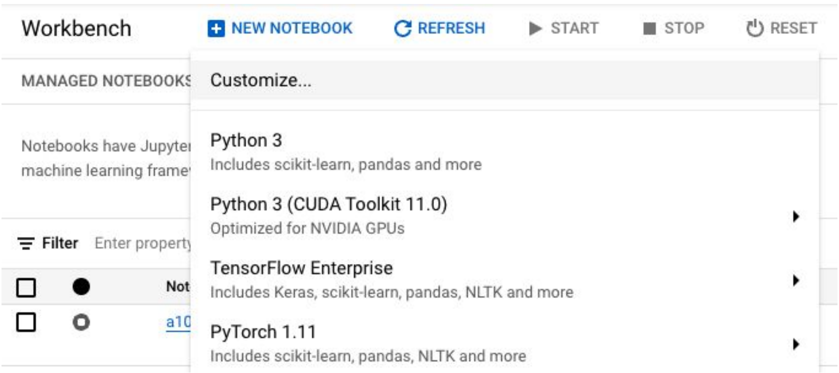
Components of ML experiments

We need:

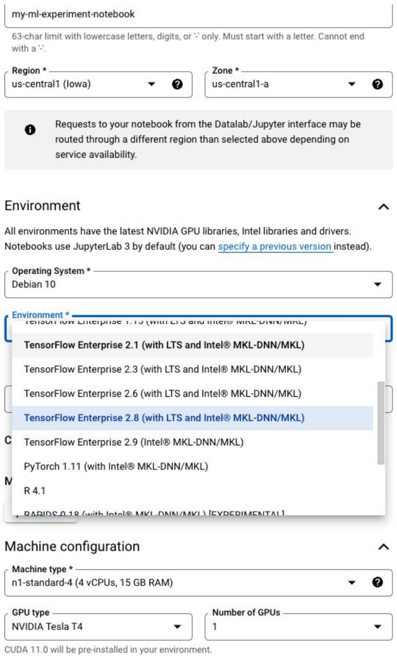
- Pre-configured environment (with libraries and CUDA stuff sorted).
- Notebook instance for better interaction with the modules.
- Access to a GPU or GPUs.
- Optionally a central storage location for storing experiment artifacts.

Vertex AI and GCS to the rescue

Vertex AI Workbench allows to do this with a few clicks!


Workbench
NEW NOTEBOOK
REFRESH
START
STOP
RESET
MANAGED NOTEBOOKS
Customize...
Python 3
Includes scikit-learn, pandas and more
Python 3 (CUDA Toolkit 11.0)
Optimized for NVIDIA GPUs
TensorFlow Enterprise
Includes Keras, scikit-learn, pandas, NLTK and more
PyTorch 1.11
Includes scikit-learn, pandas, NLTK and more
Filter
Enter property
Not
a10

Notebooks have Jupyter
machine learning frame



my-ml-experiment-notebook

63-char limit with lowercase letters, digits, or '-' only. Must start with a letter. Cannot end with a '-'.

Region * us-central1 (Iowa)

Zone * us-central1-a

Requests to your notebook from the Datalab/Jupyter interface may be routed through a different region than selected above depending on service availability.

Environment

All environments have the latest NVIDIA GPU libraries, Intel libraries and drivers. Notebooks use JupyterLab 3 by default (you can specify a previous version instead).

Operating System * Debian 10

Environment *

TensorFlow Enterprise 1.15 (with LTS and Intel® MKL-DNN/MKL)
TensorFlow Enterprise 2.1 (with LTS and Intel® MKL-DNN/MKL)
TensorFlow Enterprise 2.3 (with LTS and Intel® MKL-DNN/MKL)
TensorFlow Enterprise 2.6 (with LTS and Intel® MKL-DNN/MKL)
TensorFlow Enterprise 2.8 (with LTS and Intel® MKL-DNN/MKL)
TensorFlow Enterprise 2.9 (Intel® MKL-DNN/MKL)
PyTorch 1.11 (with Intel® MKL-DNN/MKL)
R 4.1
RAPIDS 0.18 (with Intel® MKL-DNN/MKL) [EXPERIMENTAL]

Machine configuration

Machine type * n1-standard-4 (4 vCPUs, 15 GB RAM)

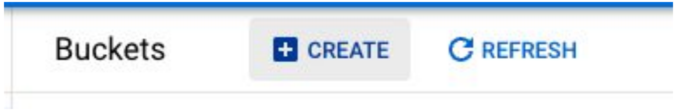
GPU type NVIDIA Tesla T4

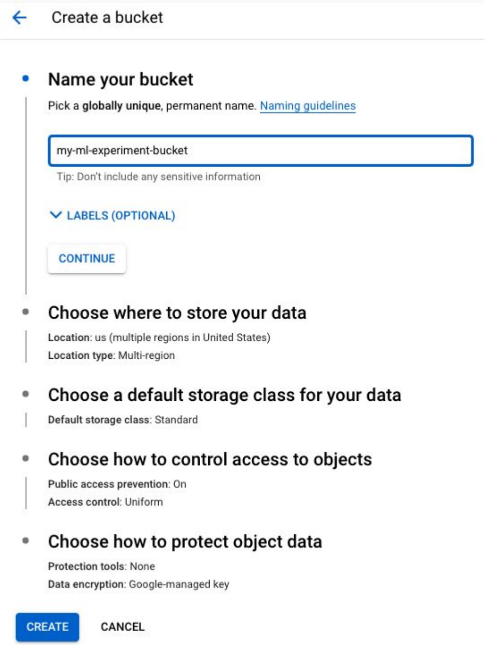
Number of GPUs 1

CUDA 11.0 will be pre-installed in your environment.

Vertex AI and GCS to the rescue

Same applies to GCS too!




Create a bucket

- **Name your bucket**

  Pick a **globally unique**, permanent name. Naming guidelines

  my-ml-experiment-bucket

  Tip: Don't include any sensitive information

  LABELS (OPTIONAL)

  CONTINUE

- **Choose where to store your data**

  Location: us (multiple regions in United States)
  Location type: Multi-region

- **Choose a default storage class for your data**

  Default storage class: Standard

- **Choose how to control access to objects**

  Public access prevention: On
  Access control: Uniform

- **Choose how to protect object data**

  Protection tools: None
  Data encryption: Google-managed key

CREATE CANCEL

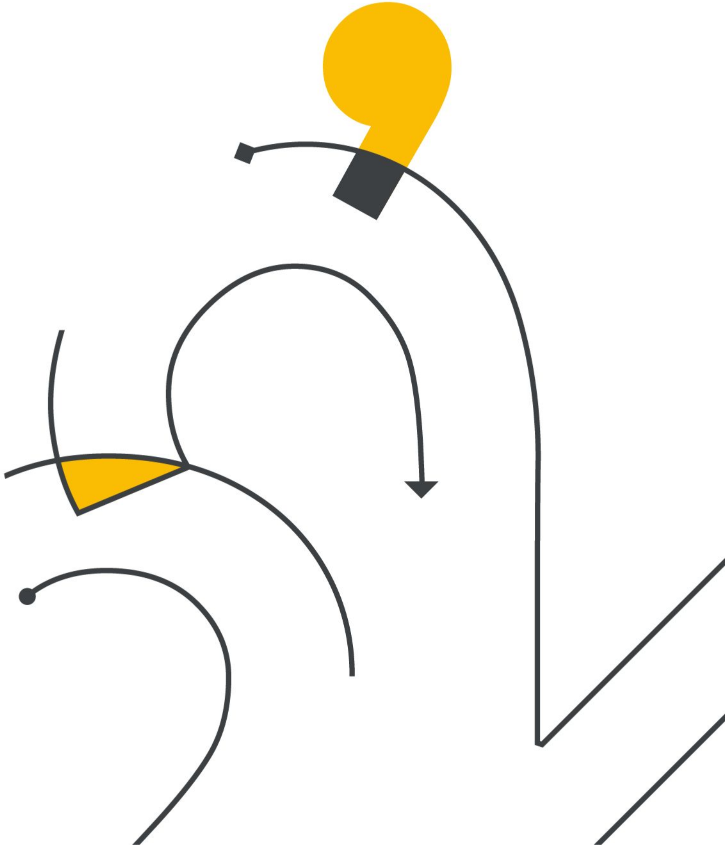
Problems with the typical workflow

Handling hardware components

- Manual efforts
  - Redundancy from repeating the instructions every time a new instance needs to be created.
  - What if a step is missed out?
- Easy to forget about cleaning up the resources after experimentation
  - Multiple moving pieces (notebook instance, GCS bucket, etc.)

> Provisioning infrastructure through point-and-click GUI’s or custom scripts is slow, error-prone, inefficient, and doesn't scale.”

Terraform



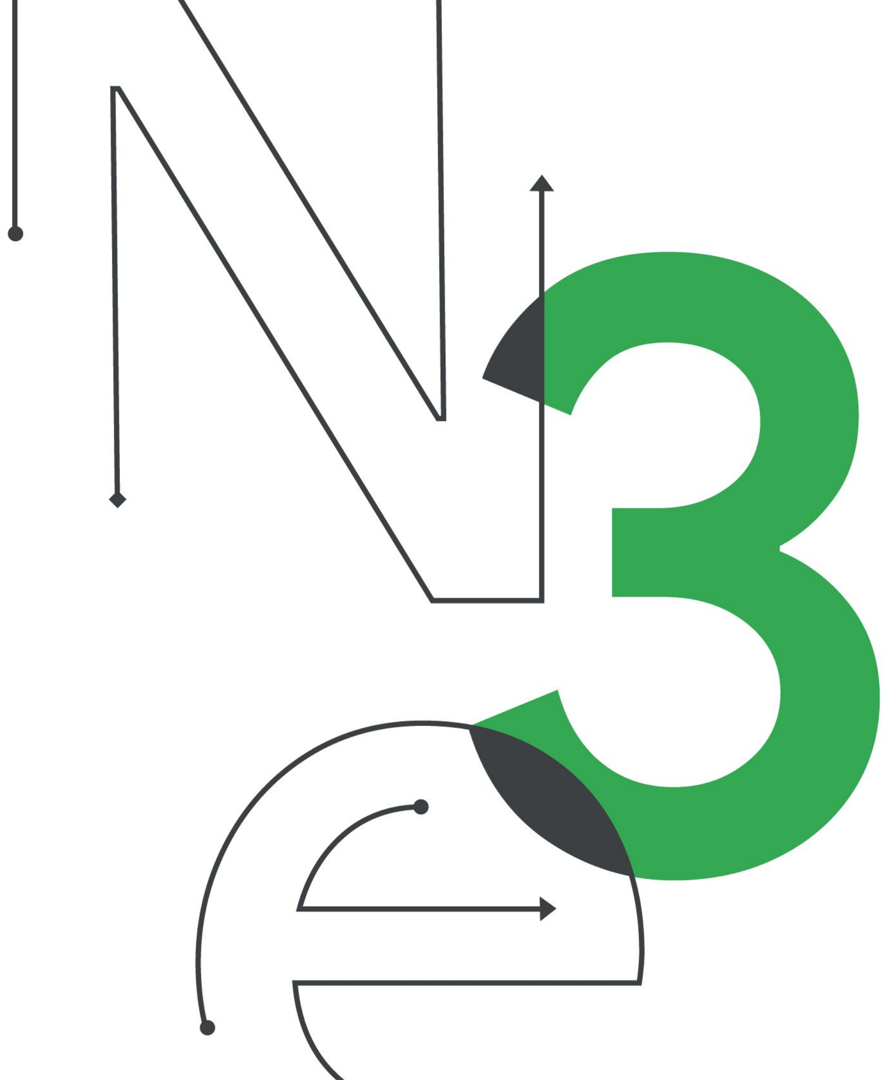
Further extensions

- Supporting custom dependencies
- Supporting Python virtual environments

Observations

- Bad:
  - Multiple interconnected steps that can be easily missed out and wreak havoc
- Good:
  - Single commands to provision and deprovision the resources

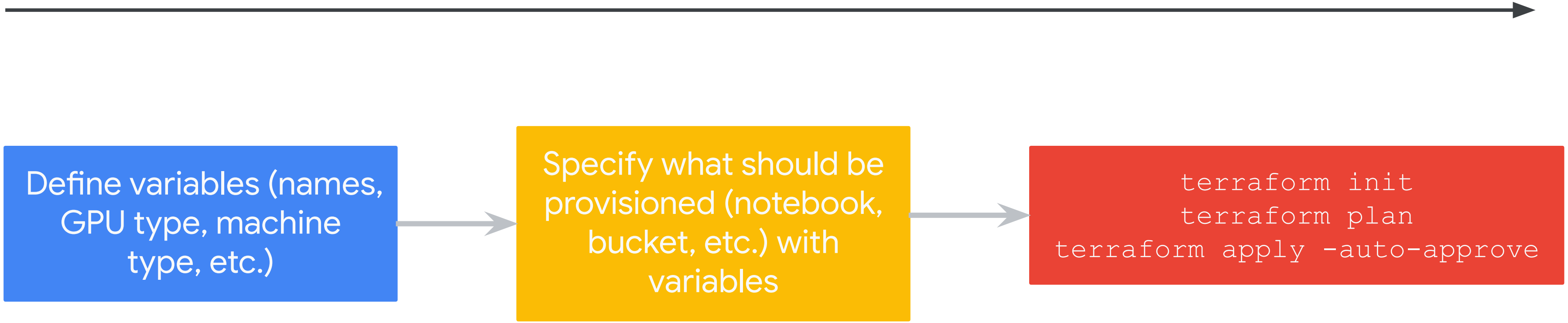
Enter Terraform



With HashiCorp Terraform, provisioning and security can be automated based on infrastructure and policy as code. Infrastructure and policies are codified, shared, managed, and executed within a workflow that is consistent across all infrastructure.

Terraform

Realize the same workflow but with code

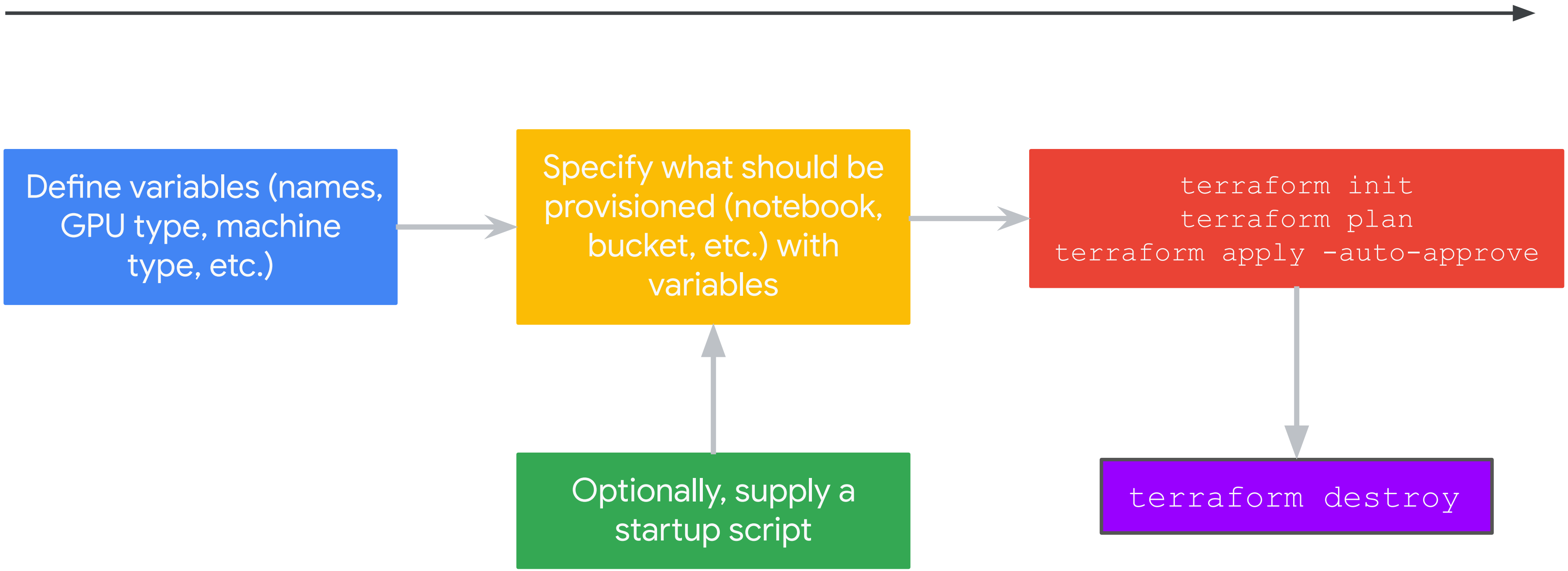


Define variables (names, GPU type, machine type, etc.)

Specify what should be provisioned (notebook, bucket, etc.) with variables

```
terraform init
terraform plan
terraform apply -auto-approve
```

Realize the same workflow but with code



Define variables (names, GPU type, machine type, etc.)

Specify what should be provisioned (notebook, bucket, etc.) with variables

Optionally, supply a startup script

```
terraform init
terraform plan
terraform apply -auto-approve
```

```
terraform destroy
```

Standard anatomy for Terraform code

- `variables.tf` (define variables)
- `main.tf` (specify what should be provisioned with `variables.tf`)
- scripts/
  - `auto_shutdown.sh` (configure automatic shutdown)
  - `dependencies.sh` (configure additional dependencies)

`.tf` is the extension for HCL (Hashicorp Configuration Language)

The `variables.tf` file

- GCP related information
  - Project ID
  - Location
  - Region
- Notebook related information
  - Name
  - Machine type
  - GPU type and count
  - Image family
- GCS related information
  - Name
  - Region

The `variables.tf` file

- GCP related information

```
variable "gcp" {
  type = object(
    {
      project_id = string
      location   = string
      region     = string
    }
  )
  default = {
    project_id = "my-gcp-project"
    location   = "us-central1-a"
    region     = "us-central1"
  }
}
```

The `variables.tf` file

- Notebook related information

```
variable "notebook" {
  type = object(
    {
      notebook_name      = string
      machine_type       = string
      gpu_type           = string
      …
    }
  )
  default = {
    notebook_name      = "my-ml-nb"
    machine_type       = "n1-standard-8"
    gpu_type           = "NVIDIA_TESLA_T4"
    …
  }
}
```

The `variables.tf` file

- GCS related information

```
variable "gcs" {
  type = object(
    {
      name          = string
      location      = string
      force_destroy = bool
    }
  )
  default = {
    name          = "my-ml-bucket"
    location      = "us-central1"
    force_destroy = false
  }
}
```

The `main.tf` file

- Provision notebook (Vertex AI Workbench) resource

```
resource "google_notebooks_instance" "notebook_instance" {
  project      = var.gcp.project_id
  name         = var.notebook.notebook_name
  machine_type = var.notebook.machine_type
  location     = var.gcp.location
  ...
}
```

The `main.tf` file

- Provision notebook (Vertex AI Workbench) resource

```
resource "google_notebooks_instance" "notebook_instance" {
  ...
  network = ...
  subnet  = ...

  vm_image {
    project      = local.image_project
    image_family = var.notebook.image_family
  }

  accelerator_config {
    type       = var.notebook.gpu_type
    core_count = var.notebook.gpu_count
  }
  ...
}
```

The `main.tf` file

- Provision notebook (Vertex AI Workbench) resource

```
resource "google_notebooks_instance" "notebook_instance" {
  ...
  install_gpu_driver = var.notebook.install_gpu_driver
  boot_disk_size_gb  = ...
  data_disk_size_gb  = ...
}
```

The `main.tf` file

- Provision GCS bucket resource

```
resource "google_storage_bucket" "default" {
  project                     = var.gcp.project_id
  name                        = var.gcs.name
  location                    = var.gcs.location
  storage_class               = "REGIONAL"
  force_destroy               = var.gcs.force_destroy
  uniform_bucket_level_access = true
}
```

Optionals

- Pass additional scripts to
  - Install additional dependencies
  - Automatically shutdown the instance w.r.t low CPU utilization
- Pass a URI of a predefined container image (`container_image`)
- Store the Terraform states to a remote location for better concurrency and collaboration

State management: Shell-scripts vs. Terraform

- Terraform comes with state management
  - Keeps track of what's successfully provisioned or failed to provision
  - Control over the entire lifecycle of the infrastructure
  - Easy “planning” of the hardware to be provisioned for previewing changes
  - Enforces version control providing ease of collaboration

bit.ly/next-22-tfm

Demo

Wrapping up

- Manual hardware provisioning is faulty, introducing redundancy and rough edges.
- Streamlining the provisioning workflow establishes DRY and helps teams to have reproducibility when provisioning infrastructure.
- With infrastructure as code, we get:
    - Version control
    - Maintainability
    - Reusability

Thank you

Sayak Paul (@RisingSayak)

bit.ly/next-22-tfm

Google Cloud

Next '22