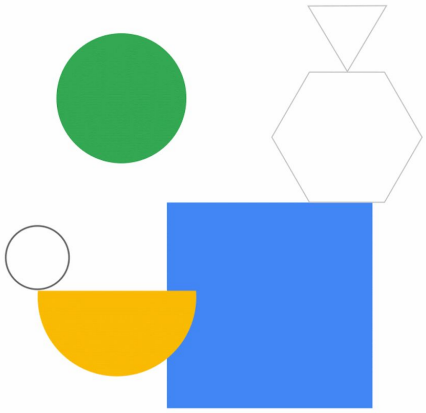
Google Cloud

Introduction to the Terraform State



Objectives

After you complete this module, you will be able to:

01 Define Terraform state.

02 List the benefits of storing state file remotely.

03 Explain how to store the Terraform state in a Google Cloud Storage bucket.

04 Explain Terraform state best practices.

The module starts with an introduction to Terraform state. You'll then learn about the different ways to store Terraform state. Later in the module you'll explore the benefits of storing the state file in a remote location. While there are many remote locations in which you can store the state file, this module describes how to store it in a Google Cloud Storage Bucket. You'll wrap up the module by learning best practices for working with state files.

Let's dive in.

Topics

01 Terraform state overview

02 Different ways to store state files

03 Storing a state file in a remote location

04 Storing a state file in a Cloud Storage bucket

05 Terraform state best practices

Terraform State

- A state is a metadata repository of your infrastructure configuration.
- A state file is stored by default in a local file named terraform.tfstate.
- A Terraform state stores the bindings between objects in a remote system and resource instances.
- The state file records the identity of an instance and updates or deletes in response to configuration changes.

```
-- servers/
   -- main.tf
   -- variables.tf
   -- outputs.tf

-- main.tf

terraform.tfstate
```

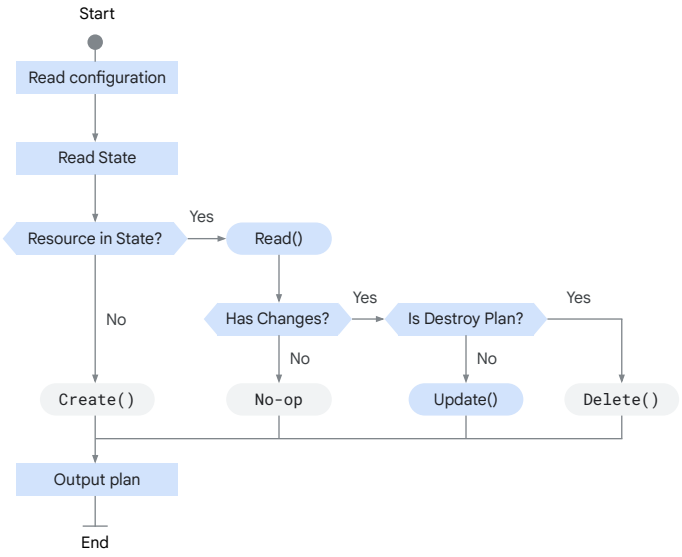

A Terraform state is a metadata repository of your infrastructure configuration. Terraform saves the state of the resources that it manages in a state file.

The state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.

Terraform uses this local state to create plans and change your infrastructure. Before any operation, Terraform does a refresh to update the state with the real infrastructure.

The primary purpose of a Terraform state is to store bindings between objects in a remote system and resource instances declared in your configuration. When Terraform creates a remote object in response to a change of configuration, it will record the identity of that remote object against a particular resource instance, and then potentially update or delete that object in response to future configuration changes.

How information is stored in a Terraform state file



Any change you make to your infrastructure will be stored in the Terraform state file which reflects the current state of your infrastructure resources. Every infrastructure component created within the resource block is identified by its name within the terraform state.

When a terraform configuration is applied for the first time, infrastructure resources are created on cloud and a state file is automatically generated. This enables the resource to be updated or destroyed in response for future changes

If a resource referred in the configuration has an entry already within the Terraform state file, then Terraform compares the configuration with the existing state file and the current live state. Based on the difference in the comparison, a plan is generated which when applied updates the resource to match the configuration defined. Once the configuration is applied, the Terraform state file is updated to reflect the current state of the infrastructure. In some cases, when the arguments cannot be updated in-place due to remote API limitations, then the resource is destroyed and re-created.

If a resource is removed from the current configuration but has an entry in the state file, then terraform compares the configuration and destroys the resources that no longer exist.

Topics

01 Terraform state overview

02 **Different ways to store state files**

03 Storing a state file in a remote location

04 Storing a state file in a Cloud Storage bucket

05 Terraform state best practices

Ways to save a state file

Save locally

```
-- servers/
   -- main.tf
   -- variables.tf
   -- outputs.tf
-- main.tf
terraform.tfstate
```

The `terraform apply` command automatically generates a state file that is saved in a working directory.

Save remotely

The file is stored in a remote location like a Google Storage bucket or Terraform Cloud.

Local state file: Terraform, by default, saves local state files in the current working directory with a .tfstate extension, so they don't require additional maintenance. Local states work well when there's only one developer working on a project. The default configuration can get tricky when multiple developers run Terraform simultaneously and each machine has its own understanding of the current infrastructure.

Remote state file: If you are working in a team where multiple developers are writing the Terraform code to manage the infrastructure, then you should store Terraform files remotely on a central location, so that if your infrastructure is changed, your Terraform state file is updated and is in sync with others.

Topics

01 Terraform state overview

02 Different ways to store state files

03 Storing a state file in a remote location

04 Storing a state file in a Cloud Storage bucket

05 Terraform state best practices

Issues with storing the Terraform state locally

No shared access	For any update to the infrastructure, each member of your team needs access to the same state file.
No locking	When team members run Terraform at the same time, they run into conflict in access, which leads to data corruption and data loss.
No confidentiality	State file exposes all sensitive data such as username and password of the database.

As mentioned in the previous slide, remote state works great for team environment. But, let us explore some of the issues you may face when you use local state for team environments.

- No Shared access: To be able to use Terraform to update your infrastructure, each member of your team needs access to the same Terraform state files. That means you must store those files in a shared location.
- No Locking state files: Without locking, if two team members are running Terraform at the same time, you might run into race conditions because multiple Terraform processes are making updates to the state files, which leads to conflicts, data loss and state file corruption.
- No Confidentiality: The information on Terraform state files is in plain text. This exposes all sensitive data such as username and password of the database.

Benefits of storing a state file in a remote location

Automatic updates	Remote state supports automatic updates to the state file.
Locking	Cloud Storage buckets natively support state locking.
Secure access	Google Cloud Storage buckets support encryption and IAM policies.

Remote backends solve all three of the issues discussed on the previous slide.

- Automatic updates: Once you configure a remote backend, Terraform will automatically load the state file from that backend every time you run `terraform plan` or `terraform apply`. It will also automatically store the state file in that backend after each `terraform apply`, so there's no chance of manual error.
- Locking: Cloud Storage buckets natively support state locking. Whenever `terraform apply` is run simultaneously, at any point in time only one person can secure a lock to ensure that the state file is not corrupted with simultaneous updates.
- Secure access: Cloud Storage buckets natively support encryption in transit and encryption on disk of the state file. Moreover, Google Cloud Storage includes various ways to configure access permissions (for example, using IAM policies with a bucket), so you can control who has access to your state files. Though Google Cloud Storage buckets are encrypted at rest, you can use customer-supplied encryption keys to provide an added layer of protection. Do this by using the GOOGLE_ENCRYPTION_KEY environment variable. Although no secrets should be in the state file, always encrypt the state as an extra measure of defense.

Topics

01 Terraform state overview

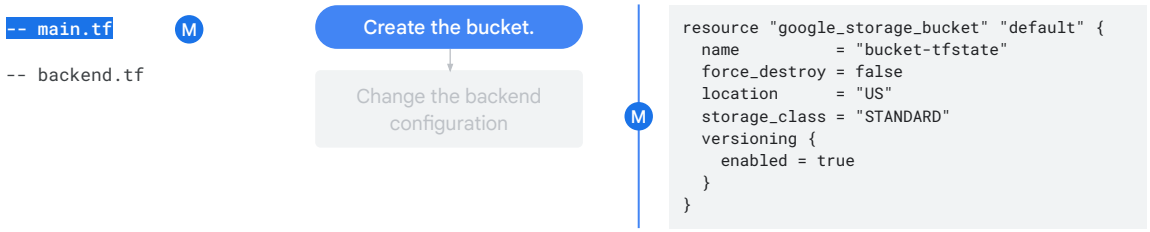
02 Different ways to store state files

03 Storing a state file in a remote location

04 **Storing a state file in a Cloud Storage bucket**

05 Terraform state best practices

Store Terraform state remotely in a Cloud Storage bucket

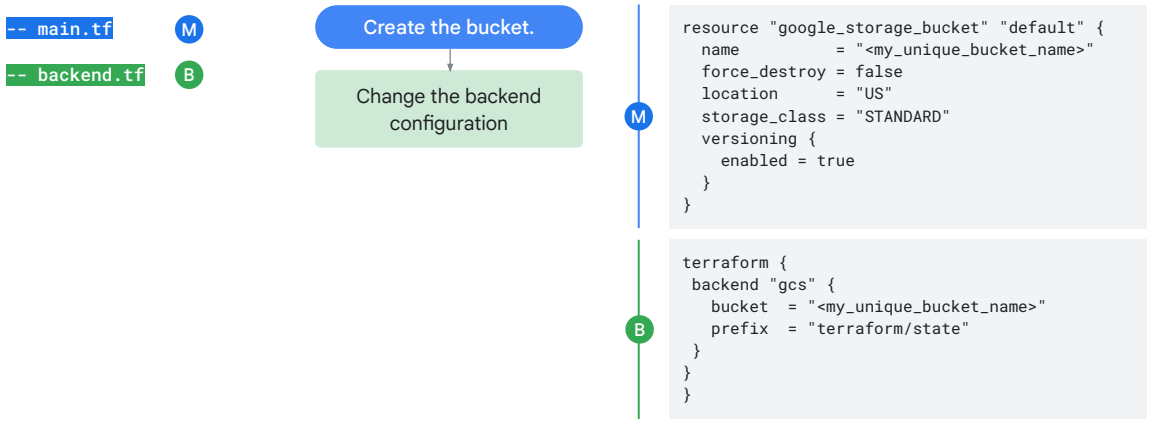


Let us explore the steps to store the Terraform state remotely in a Cloud Storage bucket:

1. Add the following google_storage_bucket Terraform resource to a Terraform config file, such as main.tf. In the code snippet, the location field is hard-coded to US (which means a multi-region bucket in the US is created). You can change this field to a location of your choice.

   Then run `terraform apply` to create the storage bucket.

Store Terraform state remotely in a Cloud Storage bucket



2. Add the code to a new Terraform configuration file called **backend.tf**. Ensure that the BUCKET_NAME is updated to match the name of your new Cloud Storage bucket. Run `terraform init` to configure your Terraform backend. Terraform detects that you already have a state file locally and prompts you to copy it to the new Cloud Storage bucket. Enter yes.

   After you run this command, your Terraform state is stored in the Cloud Storage bucket. Terraform pulls the latest state from this bucket before running a command, and pushes the latest state to the bucket after running it.

Example of a state file

```
{
  "version": 4,
  "terraform_version": "1.2.3",
  "serial": 2,
  "lineage": "f9e0d82a-4bb4-1db2-7912-3d31117903cc",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "google_storage_bucket",
      "name": "test-bucket-for-state",
      "provider": "provider[\"registry.terraform.io/hashicorp/google\"]",
      "instances": [
        {
          "schema_version": 0,
…
```

Snippet of the state file from the Cloud Storage bucket

Here is a snippet of the state file from the Cloud Storage bucket. The state includes the metadata of the resources created, such as the resource type, resource name and the provider name.

Topics

01 Terraform state overview

02 Different ways to store state files

03 Storing a state file in a remote location

04 Storing a state file in a Cloud Storage bucket

05 Terraform state best practices

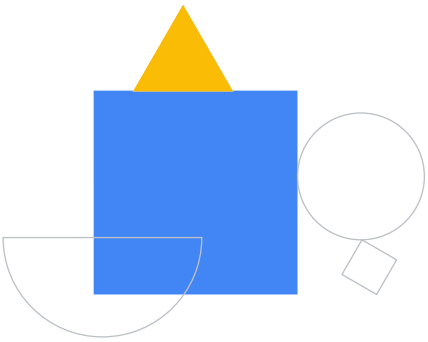
Terraform state best practices

Use remote state when working in teams	Remote state supports locking and versioning.
Encrypt state	Use customer-supplied encryption keys to add a layer of protection.
Don't store secrets in a state file	Avoid storing secrets in state because Terraform stores secret values in plaintext.
Don't modify state manually	Use the terraform state command when when you need to modify a state.

- Ensure that a remote state provider such as Google Cloud Storage is used for storing remote states. This supports locking and versioning, and you should avoid using a local state. For Google Cloud customers, we recommend using the Cloud Storage state backend. This approach locks the state to allow for collaboration as a team. It also separates the state and all the potentially sensitive information from version control. Ensure that only the build system and highly privileged administrators have access to the bucket that is used for remote state. To prevent accidentally committing development state to source control, use gitignore for Terraform state files.
- Encrypt state: although Google Cloud buckets are encrypted at rest, you can use customer-supplied encryption keys to provide an added layer of protection. Do this by using the GOOGLE_ENCRYPTION_KEY environment variable. Although no secrets should be in the state file, always encrypt the state as an extra measure of defense.
- Don't store secrets in a state: Many resources and data providers in Terraform store secret values in plain text in the state file. Where possible, avoid storing secrets in a state file.
- Don't modify Terraform state manually: The Terraform state file is critical for maintaining the mapping between Terraform configuration and Google Cloud resources. Corruption can lead to major infrastructure problems.

Lab

Creating a remote backend



In this lab, you will learn how to perform the following tasks:

- Create a local backend.
- Create a Cloud Storage backend.
- Refresh your Terraform state.

Quiz

Quiz | Question 1

Question

Select the three benefits of storing a Terraform state file remotely.

A. Usage Analytics

B. Locking

C. Secure access

D. Sharing and delegation

E. Automatic insights

Quiz | Question 1

Answer

Select the three benefits of storing a Terraform state file remotely.

A. Usage Analytics

B. Locking

C. Secure access

D. Sharing and delegation

E. Automatic insights

Quiz | Question 2

Question

A state file is stored by default in a local file named `terraform.tfstate`.

A. True

B. False

Quiz | Question 2

Answer

A state file is stored by default in a local file named `terraform.tfstate`.

A. True 

B. False

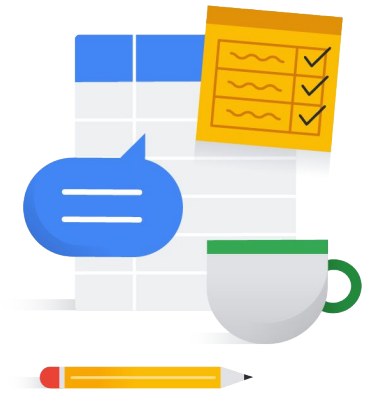
Module Review

01 Define Terraform State.

02 List the benefits of storing a state file remotely.

03 Explain how to store the state in a Google Cloud Storage Bucket.

04 Explain Terraform state best practices.



This short module covered the Terraform state and listed the benefits of storing the state in a remote location, including in a Google Cloud Storage Bucket. You also learned some best practices for states and applied what you learned in the lab.