spacelift // The Most Flexible IaC Management Platform

Terraform State

Knowing how to manage state is crucial when it comes to keeping your IaC safe.

State operations

- stores current infrastructure configuration
- keeps track of managed resources' properties and metadata
- ensures correct changes are made to infrastructure when Terraform runs
- can be stored locally or remotely for collaboration and versioning
- can track complete or subset of resources
- can be used for troubleshooting and auditing
- can be imported, exported, and deleted
- stored in JSON format
- locked during modification to avoid conflicts
- updated during apply/destroy/import/state/refresh operations

```
$ terraform state help
# Will list all available state commands
```

```
$ terraform state list
# Lists all resources in the Terraform state.
```

```
$ terraform state show
# Shows details of a specific resource in the Terraform state.
```

```
$ terraform state mv
# Renames a resource in the Terraform state.
```

```
$ terraform state pull
# Downloads the current state from a remote backend and saves it locally
```

```
$ terraform state push
# Uploads the current state to a remote backend.
```

```
$ terraform state rm
# Removes a resource from the Terraform state.
```

```
$ terraform state replace-provider
# Replaces a provider in the state
```

Local state

- By default, Terraform will keep your state on your local computer.
- This is not recommended, but it is useful when you are learning Terraform.
- When you are running **terraform apply** two files will be generated:
  - terraform.tfstate → this is generated on the first apply
  - terraform.tfstate.backup → this will be generated when there is a difference between your first apply and the second one
- You don't have to configure anything when you are using a local state.

Remote state

- State will be stored in a remote location → This facilitates collaboration between the engineers.
- Many supported options are available to host your remote state.
- You can lock the state, meaning that whenever somebody is running something against that state, they will acquire a lock and nobody else will be able to run other commands:
  - reduces the risk of overlapping with changes
  - keeps the integrity of the state
  - **the mechanism doesn't come by default, your remote backend should support it**
- Enables sharing outputs between different Terraform configurations by using the terraform_remote_state data source

S3 Backend

- Stores your state in an AWS S3 Bucket
- Supports state locking when you use AWS Dynamo DB table (the table must have partition key named LockID of String type, otherwise, state locking will not work)
- Requires you to configure your AWS credentials

S3 backend with no state lock

```
terraform {
 backend "s3" {
   bucket = "my-state-bucket" # state will be stored in this bucket
   key    = "path/to/my/state-file" # the name of the file in which to keep your state
   region = "eu-west-1" # AWS region
 }
}
```

S3 backend with state locking

```
terraform {
 backend "s3" {
   bucket         = "my-state-bucket"
   key            = "path/to/my/state-file"
   region         = "eu-west-1"
   encrypt        = true
   dynamodb_table = "my_table"
 }
}
```

AzureRM Backend

- Stores your state in a Blob (Blob → Blob Container → Blob Storage Account)
- Supports locking natively
- Possible to log in to Azure via CLI | Service Principal | OIDC | MSI | Azure AD, if you provide additional parameters to the backend configuration

AzureRM - CLI | Service principal

```
terraform {
 backend "azurerm" {
   resource_group_name  = "rg_of_storage_account"
   storage_account_name = "storage_acc_name"
   container_name       = "container_name"
   key                  = "state_file_name"
 }
}
```

GCS Backend

- Stores your state in a Google Cloud Bucket
- Supports locking natively

AzureRM - CLI | Service principal

```
terraform {
 backend "gcs" {
   bucket  = "state_bucket"
   prefix  = "terraform/state"
 }
}
```

HTTP Backend

- Stores the state using a REST client
- Supports state locking, but you will need to provide a locking endpoint and an unlocking endpoint
- Endpoints should return **423: Locked** or **409: Conflict with the holding lock info when it's already taken** or **200: OK**
- Locking mechanism won't work if endpoints return any other status

Http backend without lock

```
terraform {
 backend "http" {
   address = "http://state.api.com/state"
 }
}
```

Http backend with lock

```
terraform {
 backend "http" {
   address        = "http://state.api.com/state"
   lock_address   = "http://state.api.com/state"
   unlock_address = "http://state.api.com/state"
 }
}
```

Cloud Backend

- Terraform Cloud Backend preferred to the Remote Backend because it has more features.
- Works only with Terraform Cloud, creating a workspace that manages state for you

```
terraform {
 cloud {
   organization = "org"
   hostname = "app.terraform.io" # Optional; defaults to app.terraform.io

   workspaces {
     tags = ["workspace1"]
   }
 }
}
```