**Remote State**

**Remote State**

By default, Terraform stores state locally in a file named terraform.tfstate. When working with Terraform in a team, the use of a local file makes Terraform usage complicated because each user must make sure they always have the latest state data before running Terraform and make sure that nobody else runs Terraform at the same time.

With remote state, Terraform writes the state data to a remote data store, which can then be shared between all members of a team. Terraform supports storing state in Terraform Cloud, HashiCorp Consul, **Amazon S3**, Azure Blob Storage, Google Cloud Storage, Alibaba Cloud OSS, and more.

Remote state is implemented by a **backend**, which you can configure in your configuration's root module.

Remote state allows you to share output values with other configurations. This allows your infrastructure to be decomposed into smaller components.

Put another way, remote state also allows teams to share infrastructure resources in a read-only way without relying on any additional configuration store.

For fully-featured remote backends, Terraform can also use state locking to prevent concurrent runs of Terraform against the same state.

**Backends**

Each Terraform configuration can specify a backend, which defines where and how operations are performed, where state snapshots are stored, etc.

Backends are responsible for storing state and providing an API for state locking. **State locking** is optional. Backends are responsible for supporting state locking if possible. Not all backends support locking. If supported by your backend, Terraform will lock your state for all operations that could write state. This prevents others from acquiring the lock and potentially corrupting your state.

Despite the state being stored remotely, all Terraform commands such as terraform console, the terraform state operations, terraform taint, and more will continue to work as if the state was local.

Backends determine where state is stored. For example, the local (default) backend stores state in a local JSON file on disk.

**Amazon S3**

S3 Stores the state as a given key in a given bucket on Amazon S3. This backend also supports state locking and consistency checking via Dynamo DB, which can be enabled by setting the dynamodb\_table field to an existing DynamoDB table name. A single DynamoDB table can be used to lock multiple remote state files. Terraform generates key names that include the values of the bucket and key variables.

You can enable **Bucket Versioning** on the S3 bucket to allow for state recovery in the case of accidental deletions and human error.

Terraform state can contain sensitive data, depending on the resources in use and your definition of "sensitive." The state contains resource IDs and all resource attributes. For resources such as databases, this may contain initial passwords. When using remote backend Amazon S3, you can also enable encryption on the S3 bucket to encrypt to data.

**Configuration of S3 Backend**

First, we assume that we have a bucket created called mybucket.

Next, add the following terraform block to your config file.

terraform {

backend "s3" {

bucket = "mybucket"

key = "path/to/my/key"

region = "us-east-1"

}

}

The Terraform state is written to the key path/to/my/key.