* Terraform must store state about your managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.
* Default file – terraform.tfstate, Stored remotely too
* Terraform uses this local state to create plans and make changes to your infrastructure. Prior to any operation, Terraform does a [refresh](https://www.terraform.io/docs/commands/refresh.html) to update the state with the real infrastructure.
* Direct file editing of the state is discouraged. [terraform state](https://www.terraform.io/docs/commands/state/index.html) – used for modification.
* The CLI manages backups for you automatically. If you make a mistake modifying your state, the state CLI will always have a backup available for you that you can restore.
* Terraform state determines what resources it manages based on what exists in the state. This is how terraform plan determines what isn't created, what needs to be updated, etc.

**Purpose:**

* Mapping to real-world objects
* Metadata
* Performance
* Syncing

**Import:**

* Terraform is able to import existing infrastructure. This allows you take resources you've created by some other means and bring it under Terraform management
* Currently, only state can be imported, configuration has to be written manually.
* It can only import one resource at a time

resource "aws\_instance" "example" {

# ...instance configuration...

}

$ terraform import aws\_instance.example i-abcd1234

* As a result of the above command, the resource is recorded in the state file. You can now run terraform plan to see how the configuration compares to the imported resource, and make any adjustments to the configuration to align with the current (or desired) state of the imported object.
* Complex Imports : For example, an AWS security group imports an aws\_security\_group but also one aws\_security\_group\_rule for each rule. In this scenario, the secondary resources will not already exist in configuration, so it is necessary to consult the import output and create a resource block in configuration for each secondary resource. If this is not done, Terraform will plan to destroy the imported objects on the next run.

**Backends:**

* A "backend" in Terraform determines how state is loaded and how an operation such as apply is executed.
* By default, Terraform uses the "local" backend.
* Benefits of backends:
* **Working in a team**: Backends can store their state remotely and protect that state with locks to prevent corruption. Some backends such as Terraform Enterprise even automatically store a history of all state revisions.
* **Keeping sensitive information off disk**: State is retrieved from backends on demand and only stored in memory. If you're using a backend such as Amazon S3, the only location the state ever is persisted is in S3.
* **Remote operations**: For larger infrastructures or certain changes, terraform apply can take a long, long time. Some backends support remote operations which enable the operation to execute remotely. You can then turn off your computer and your operation will still complete. Paired with remote state storage and locking above, this also helps in team environments.

1. **Backend Initialization:**

The init command must be called:

* On any new environment that configures a backend
* ­­On any change of the backend configuration (including type of backend)
* On removing backend configuration completely

If you don’t init, error will be thrown.

1. **Backend Configuration:**

terraform {

backend "consul" {

address **=** "demo.consul.io"

path **=** "example\_app/terraform\_state"

}

}

* Only one backend may be specified and the configuration **may not contain interpolations**.
* All you have to do configure the backend is run the terraform init command.
* Partial Configuration: When some or all of the backend attributes are omitted. The remaining arguments can be passed through interactively via command-line, file and command-line key-value pairs.
* If backend settings are provided in multiple locations, the top-level settings are merged such that any command-line options override the settings in the main configuration and then the command-line options are processed in order, with later options overriding values set by earlier options.
* Changing of the backend configuration can be done using terraform init command. Terraform automatically detetcts any changes in the backend configuration and requests a reinitialiization.
* Backend can be unconfigured by simply removing the configuration block.

1. **State Storage and Locking:**

* When using a non-local backend, Terraform will not persist the state anywhere on disk except in the case of a non-recoverable error where writing the state to the backend failed. This behavior is a major benefit for backends: if sensitive values are in your state, using a remote backend allows you to use Terraform without that state ever being persisted to disk.
* You can still manually retrieve the state from the remote state using the terraform state pull command. This will load your remote state and output it to stdout. You can choose to save that to a file or perform any other operations.
* You can also manually write state with terraform state push. **This is extremely dangerous and should be avoided if possible.** This will overwrite the remote state. This can be used to do manual fixups if necessary.

1. **Backend Types:**

* **Standard**: State management, functionality covered in [State Storage & Locking](https://www.terraform.io/docs/backends/state.html)
* **Enhanced**: Everything in standard plus [remote operations](https://www.terraform.io/docs/backends/operations.html).

**State Locking:**

* If supported by your [backend](https://www.terraform.io/docs/backends), Terraform will lock your state for all operations that could write state. This prevents others from acquiring the lock and potentially corrupting your state.
* Terraform has a [force-unlock command](https://www.terraform.io/docs/commands/force-unlock.html) to manually unlock the state if unlocking failed. If you unlock the state when someone else is holding the lock it could cause multiple writers. Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed.

**Workspaces:**

* A *workspace* is a named container for Terraform state. With multiple workspaces, a single directory of Terraform configuration can be used to manage multiple distinct sets of infrastructure resources.
* To create a new workspace and switch to it, you can use terraform workspace new; to switch environments you can use terraform workspace select.

$ terraform workspace new bar

Created and switched to workspace "bar"!

You're now on a new, empty workspace. Workspaces isolate their state,

so if you run "terraform plan" Terraform will not see any existing state

for this configuration.

* Workspace can be switched to the default one using command “terraform workspace select default”.
* Current Workspace Interpolation : Within your Terraform configuration, you may include the name of the current workspace using the ${terraform.workspace} interpolation sequence. This can be used anywhere interpolations are allowed.
* For non-default workspaces, it may be useful to spin up smaller cluster sizes. For example:

resource "aws\_instance" "example" {

count **=** "${terraform.workspace == "default" ? 5 : 1}"

# ... other arguments

}

* Another popular use case is using the workspace name as part of naming or tagging behavior:

resource "aws\_instance" "example" {

tags {

Name **=** "web - ${terraform.workspace}"

}

# ... other arguments

}

* For *each* smaller configuration, you can use workspaces to model the differences between development, staging, and production. However, if you have one large Terraform configuration, it is riskier and not recommended to use workspaces to handle those differences.
* Workspaces are technically equivalent to renaming your state file.
* For local state, Terraform stores the workspace states in a directory called “terraform.tfstate.d”.
* Workspace commands – ***show, list, select, new, delete***

**Remote State:**

* By default, Terraform stores state locally in a file named "terraform.tfstate".
* Delegation and Teamwork: remote state also allows teams to share infrastructure resources in a read-only way. Remote state gives you more than just easier version control and safer storage.
* Locking and Teamwork: Terraform will automatically lock state depending on the [backend](https://www.terraform.io/docs/backends) used.

**Sensitive Data:**

* The state contains resource IDs and all resource attributes. For resources such as databases, this may contain initial passwords.
* Some resources (such as RDS databases) have options for PGP encrypting the values within the state. This is implemented on a per-resource basis and you should assume the value is plaintext unless otherwise documented.
* Storing state remotely may provide you encryption at rest depending on the backend you choose.
* [Terraform Enterprise](https://www.hashicorp.com/products/terraform/) is a commercial product from HashiCorp that also acts as a [backend](https://www.terraform.io/docs/backends) and provides encryption at rest for state. Terraform Enterprise also knows the identity of the user requesting state and maintains a history of state changes. This can be used to provide access control and detect any breaches.

**State Command:**

* Rather than modify the state directly, the terraform state commands can be used in many cases instead.
* The Terraform state subcommands all work with remote state just as if it was local state. Reads and writes may take longer than normal as each read and each write do a full network roundtrip.
* All terraform state subcommands that modify the state write backup files except for read-only(such as [list](https://www.terraform.io/docs/commands/state/list.html)) since they aren't modifying the state. The path of these backup file can be controlled with -backup.
* Subcommands:

1. List - terraform state list [options] [address...]

The command will list all resources in the state file matching the given addresses (if any). If no addresses are given, all resources are listed.

1. Mv - The terraform state mv command is used to move items in a [Terraform state](https://www.terraform.io/docs/state/index.html). This command can move single resources, single instances of a resource, entire modules, and more. This command can also move items to a completely different state file, enabling efficient refactoring.

terraform state mv [options] SOURCE DESTINATION

This command will move an item matched by the address given to the destination address. This command can also move to a destination address in a completely different state file.

1. Pull - The terraform state pull command is used to manually download and output the state from [remote state](https://www.terraform.io/docs/state/remote.html). This command also works with local state.
2. Push - The terraform state push command is used to manually upload a local state file to [remote state](https://www.terraform.io/docs/state/remote.html). This command also works with local state.

terraform state push [options] PATH

This command will push the state specified by PATH to the currently configured [backend](https://www.terraform.io/docs/backends).

1. Rm - The terraform state rm command is used to remove items from the [Terraform state](https://www.terraform.io/docs/state/index.html). This command can remove single resources, single instances of a resource, entire modules, and more.

terraform state rm [options] ADDRESS...

Remove one or more items from the Terraform state.

Items removed from the Terraform state are not physically destroyed. Items removed from the Terraform state are only no longer managed by Terraform.

The most common use case for removing resources is refactoring a configuration to no longer manage that resource (perhaps moving it to another Terraform configuration/state).

1. Show - The terraform state show command is used to show the attributes of a single resource in the [Terraform state](https://www.terraform.io/docs/state/index.html).