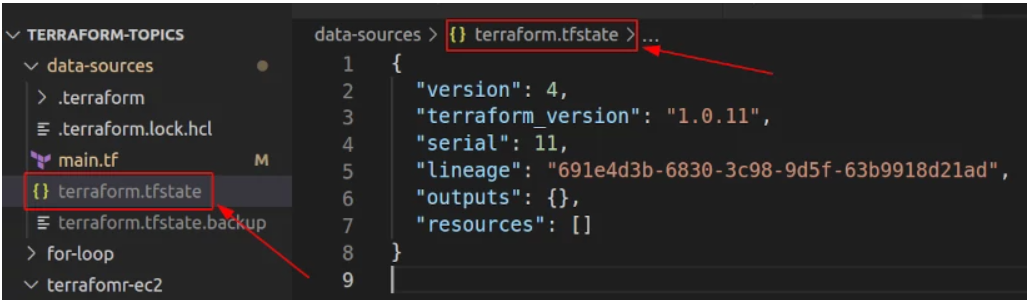
1. What is Terraform State(terraform.tfstate) file?

Terraform state file acts as a recorder for your infrastructure setup.

Terraform state file(*terraform.tfstate*) is created from the very beginning when you run your first *terraform apply* command.

Here is a screenshot from terraform workspace containing *terraform.tfstate* file -

Terraform state file(terraform.tfstate)

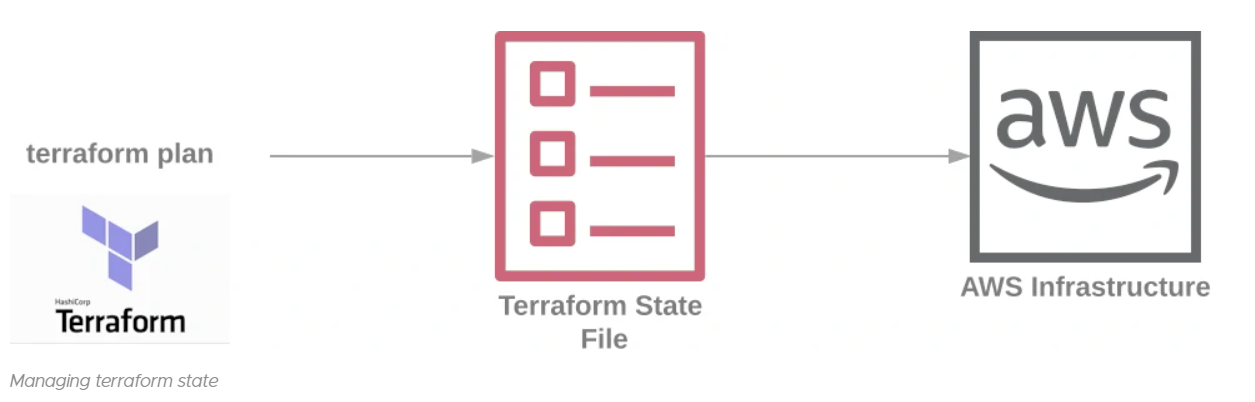


How does Terraform state file store information?

Terraform state file stores the information(metadata) about the infrastructure resources. Any change you plan to make or already made into your infrastructure then that changed information will be stored inside the Terraform state file.

You do not have to store the information inside the Terraform state file. *terraform plan* will update terraform state file before making any change to infrastructure.

Managing terraform state

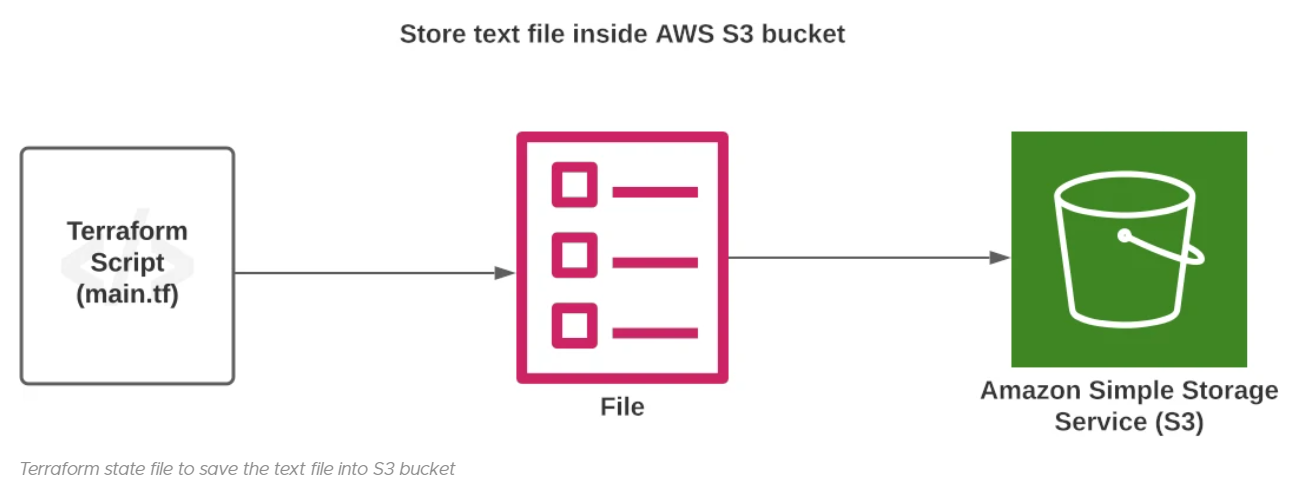


How to save terraform state file?

**1. Saving Locally -** If you are the only developer working with Terraform and managing your infrastructure then you do not have to do anything special *terraform apply* command will generate terraform state file for your workspace and save it inside your working directory.

**2. Saving Remotely -** If you are working in a team where multiple developers writing the terraform code to manage the infrastructure then it is highly recommended to store terraform files remotely([AWS S3 Bucket](https://www.terraform.io/docs/language/settings/backends/s3.html)) on a central location so that with every infrastructure change your terraform state file is up to date and in [sync](https://www.terraform.io/docs/language/state/purpose.html#syncing) with other.

Terraform state file to save the text file into S3 bucket



2. What is the Purpose of Terraform state file?

Any change which you make to your resource will be first reflected into the terraform state file before updating it onto the cloud infrastructure. So terraform state file will start storing each action which you take on the terraform resource.

Before you use *terraform apply* or *terraform destroy* command terraform always validates your resource configuration with the terraform state file and if the changes seem to be valid then your terraform state the file is first going to be updated and then your actual resource will be updated.

***So the question comes how can terraform differentiate the multiple resources of the same type running in the cloud environment?***

Terraform has its own local database known as Terraform state file where each resource is tagged with its own unique id and this unique id is stored in the state file in the form of metadata.

3. Terraform state file Metadata

Any information which is stored inside the *terraform state file* is known as metadata. As we discussed in the previous points all the infrastructure changes are stored in the form of *Metadata* but terraform also maintains order for managing the Metadata inside the Terraform state file.

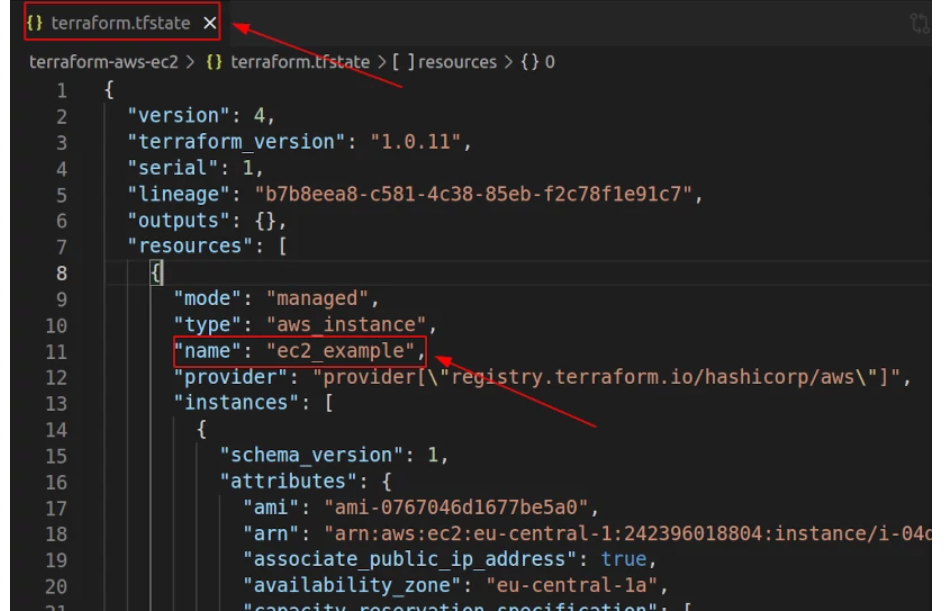
Let's take an example -

1. **Step-1:** You have written a resource block *resource "aws\_instance" "ec2\_example"*
2. **Step-2:** Terraform has generated a terraform.state file to store metadata about *ec2\_instance*
3. **Step-3:** You have added an *S3 bucket* manually from AWS Console without making any changes to terraform file.
4. **Step-4:** Now you have run the *terraform destroy* command.
5. **End Result:** Terraform only delete *EC2 instance*

***Conclusion -*** It can only find [*EC2 instance*](https://aws.amazon.com/ec2) resource metadata into terraform state file and there is no reference of *S3 bucket* because you have created the bucket manually and terraform state file have no clue of *S3 Bucket*.

This is how Terraform maintains a dependency order for maintaining the infrastructure inventory. It is highly recommended to always have provision or update your infrastructure only with the terraform.

Here is a sample copy of metadata of my terraform state file –



terraform state file metadata

=================================

*{*

*"version": 4,*

*"terraform\_version": "1.0.11",*

*"serial": 1,*

*"lineage": "b7b8eea8-c581-4c38-85eb-f2c78f1e91c7",*

*"outputs": {},*

*"resources": [*

*{*

*"mode": "managed",*

*"type": "aws\_instance",*

*"name": "ec2\_example",*

*"provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",*

*"instances": [*

*............*

*]*

*}*

*]*

*}*

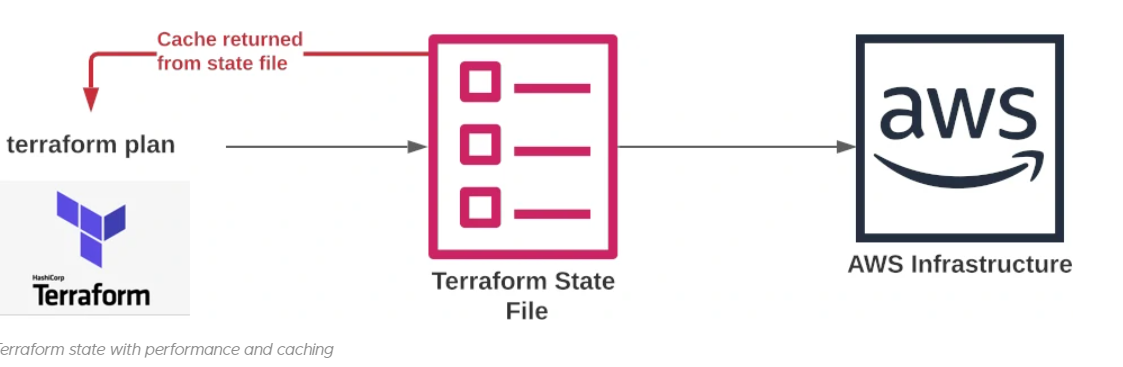
*===========================================*

4. Terraform State Performance and Caching

The next benefit of Terraform state is file *Caching*. As terraform, state file acts as a local database for your terraform code, so before making any request to cloud infrastructure it first checks the local terraform state file to show the current status of the cloud infrastructure

Imagine you have a really large infrastructure and if you wanna make a query to get information status about each resource running in cloud infrastructure. It could take really long and would not be a very effective way to know the state of your cloud infrastructure.

Terraform state with performance and caching

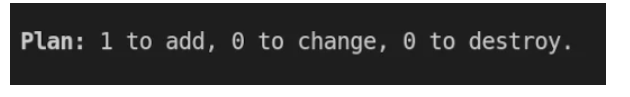


*terraform plan -* Terraform plan command is a really good example of caching and information return from *terraform state* file. Whenever we run terraform plan command it returns you back with 3 information -

1. Resources to be added
2. Resources to be changed
3. Resources to be destroyed

You can simply rely on your *terraform state* file and *terraform plan* command to get the performance benefit. Here is a sample screenshot result of *terraform plan* command stating how many resources it going to add, change and destroy -

terraform metadata performance and caching



5. What is terraform\_remote\_state Data source?

Let's assume you are a team of multiple developers working on the same Terraform project to set up and maintain your cloud infrastructure.

Considering the team size all the developers will maintain a single terraform state file so that every change which they make is always in sync with other teammates.

***But is there a way to retrieve the metadata information from the remote state file?***

Well yes, you can use *terraform\_remote\_state* [data source](https://jhooq.com/terraform-data-sources) to retrieve the metadata information of the remote state file.

**Remote backend -**

Here is an example code snippet of terraform\_remote\_state with remote backend -

*data "terraform\_remote\_state" "remote\_state" {*

*backend = "remote"*

*config = {*

*organization = "jhooq"*

*workspaces = {*

*name = "jhooq-test"*

*}*

*}*

*}*

*# Terraform >= 0.12*

*resource "aws\_instance" "jhooq-test" {*

*subnet\_id = data.terraform\_remote\_state.remote\_state.outputs.subnet\_id*

*}*

**Local Backend –**

============================================

data "terraform\_remote\_state" "local\_state" {

backend = "local"

config = {

path = "/home/rwagh/terraform/terraform.tfstate"

}

}

# Terraform >= 0.12

resource "aws\_instance" "jhooq-test" {

# ...

subnet\_id = data.terraform\_remote\_state.local\_state.outputs.subnet\_id

}

===========================================================

6. Terraform State Storage, Manual state pull/push and Locking

6.1 Storage

Terraform state files is can be stored on local as well as on remote it depends on your need. Storage of terraform state file is defined by the backend.

**Local Storage-** Consider the following example of where terraform state storage is local –

data "terraform\_remote\_state" "remote\_state" {

backend = "local"

}

1. When using the local storage terraform will persist/save the state file on the local disk
2. But if you are using the local storage then you have to manually push and pull the Terraform state file.

**Remote Storage-** Here is one more example in which we have defined the storage as remote –

data "terraform\_remote\_state" "remote\_state" {

backend = "remote"

}

*BASH*

1. When using the remote storage terraform will not persist/save the state file on the local disk.
2. Remote storage of Terraform state files is really beneficial for teams with multiple developers

6.2 Manual state pull/push

In the previous point, we discussed how to store state files locally and remotely. But apart from storing the state file there two more important operations -

1. Pull terraform state file
2. Push terraform state file

**1. Pull terraform state file-** It is quite often for a developer to take a break from work and during the break if developer's terraform project is out of sync then it is always recommended to get in sync with all the updates which has happened.

So apart from doing the normal *git pull* it is also good to do the *terraform state pull* also so that all the latest changes which have been done on to terraform state file will be retrieved -

1terraform state pull

*BASH*

**2. Push terraform state file -** It is a dangerous command to work with. It can potentially lead you to an inconsistent state where you might override other team member changes on to terraform state file. It is often discouraged to use *terraform state push* command.

1terraform state push

*BASH*

**How to terraform prevents us from accidental state push?**

**1. Differing Lineage-** Terraform has a safety mechanism on to terraform state file. Each time you create a terraform state file it will assign a *lineage ID* to the state file. If you are attempting to push the terraform state file with a different *lineage ID* then terraform will not allow it.

**2. Higher Serial Number-** Apart from the unique lineage ID terraform also assigns a unique and higher serial number to terraform state file. If you are attempting to push terraform state file which has a lower higher serial number than Terraform will not allow you to push terraform state file.

**How to force push to terraform state file?**

Considering all the safety mechanisms (lineage, Higher serial number) provided by Terraform, if you still wanna override those safety settings then you can use ***-force*** flag to push your Terraform state file.

1terraform state push -force

*BASH*

6.3 Terraform State Locking

Terraform has one more safety mechanism known as [*locking*](https://jhooq.com/terraform-state-file-locking) by default terraform puts a lock on the terraform state file(*if the backend supports the locking*).

[Terraform state locking](https://jhooq.com/terraform-state-file-locking) will prevent any other developer from updating terraform state file. Terraform locking always happens behind the scene but if terraform locking fails then terraform will not let you continue.

**How to disable the terraform state locking?**

Terraform state locking can be disabled by passing the flag ***-lock*** but is not at all recommended to use.

**How to Force Unlock Terraform state file?**

If someone has already acquired a lock on Terraform state file then you can use the ***force-unlock*** flag to override the lock. But there is a potential risk while using the ***force-unlock*** flag because multiple developers can perform the write operation on terraform state files which can lead to inconsistent behavior.

7. Terraform workspaces and state file

[Terraform workspaces](https://jhooq.com/terraform-workspaces) are a little special because terraform always creates a state file for each workspace but it has a little different name which looks like *terraform.tfstate.d*

The rest of Terraform state file handling is still the same.

Conclusion

Terraform state file is always the core heart of terraform project. You must treat the terraform state file with care otherwise, you might end up in a situation where your terraform project is full of mess and hard to manage.

The best practices for using terraform state file which I could recommend is -

1. If you have more than one developer working on Terraform project then it is always recommended to use remote backend to store your state file remotely.
2. If you are storing terraform state file remotely then you should always use *terraform state pull* before you start working with your terraform project
3. For sensitive data always consider securing the terraform state file because by default terraform store the state file in the JSON format.
4. Do not try to force push the terraform state file, it will always lead you to inconsistent behavior.
5. Use [terraform workspaces](https://jhooq.com/terraform-workspaces) and [terraform modules](https://jhooq.com/terraform-module) more often to effectively manage the Terraform state file.