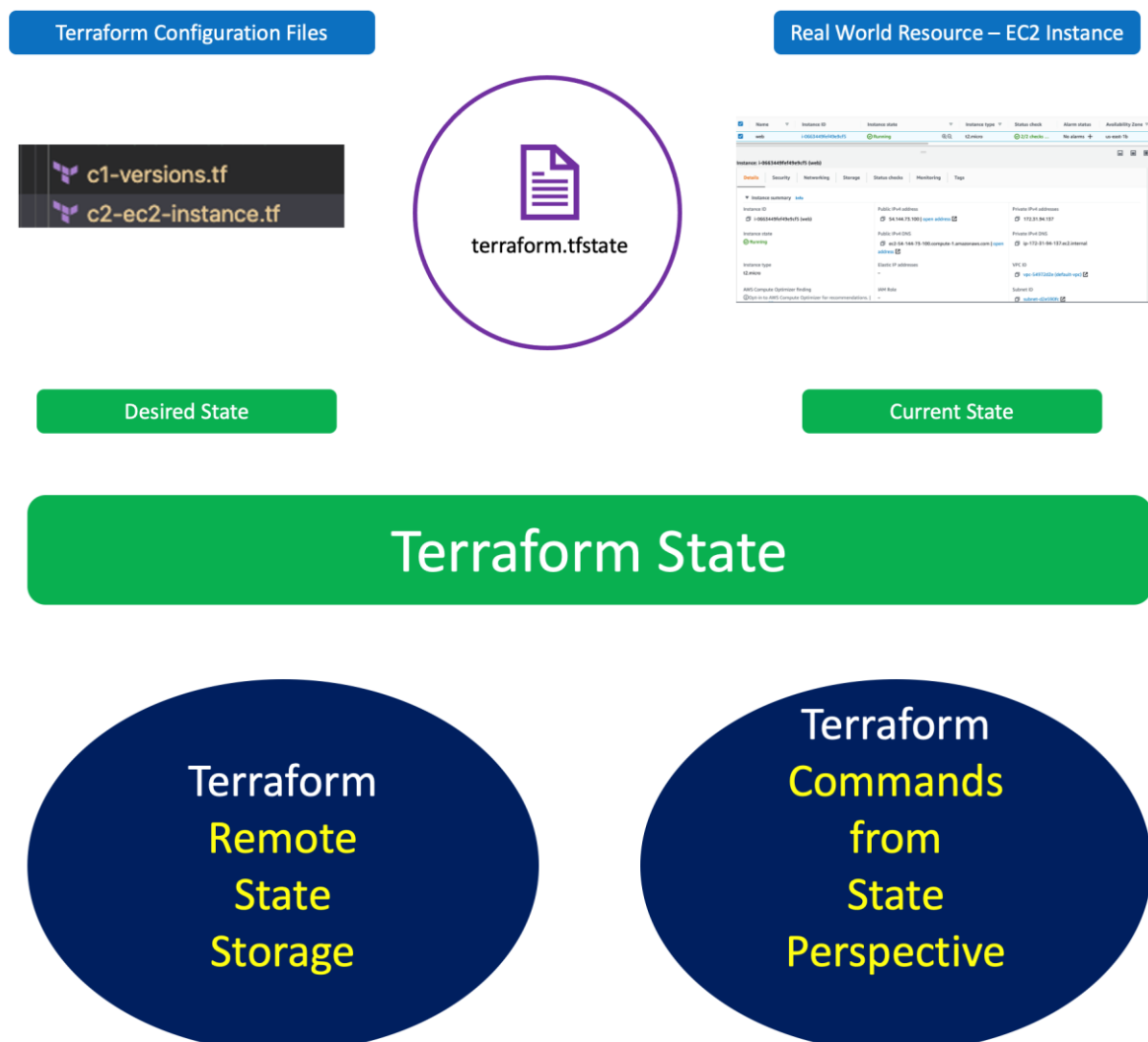


Terraform Remote State Storage and State Locking with AWS S3 and DynamoDB :

Step-01: Introduction

- Understand Terraform Backends
- Understand about Remote State Storage and its advantages
- This 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.
- Create AWS S3 bucket to store terraform.tfstate file and enable backend configurations in terraform settings block
- Understand about State Locking and its advantages
- Create DynamoDB Table and implement State Locking by enabling the same in Terraform backend configuration

Desired & Current Terraform States



What is Terraform Backend ?

Backends are responsible for storing state and providing an API for state locking.

Terraform
State Storage



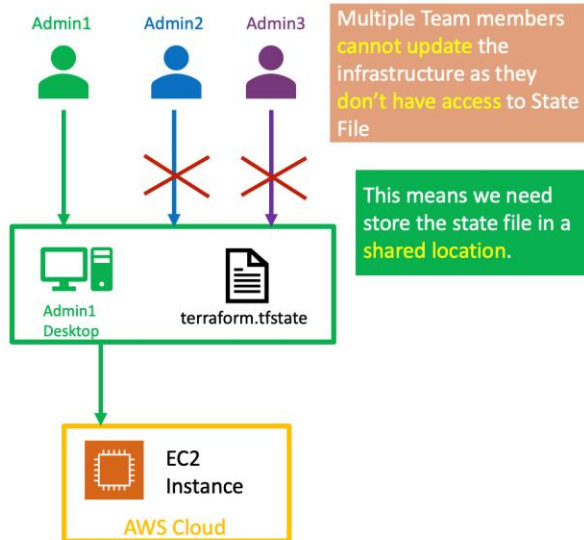
AWS S3 Bucket

Terraform
State Locking

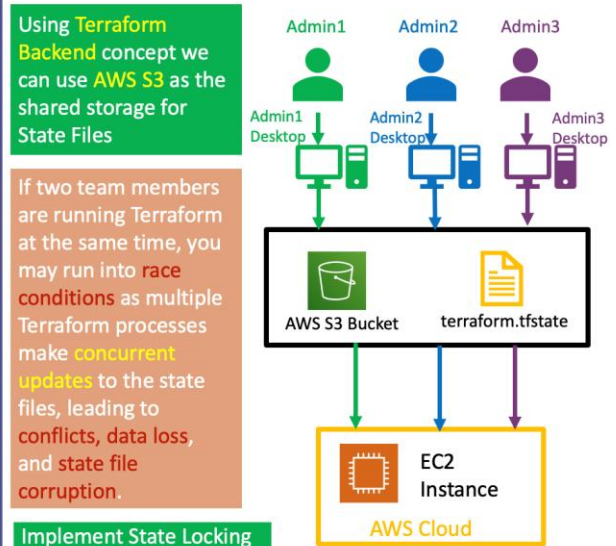


AWS DynamoDB

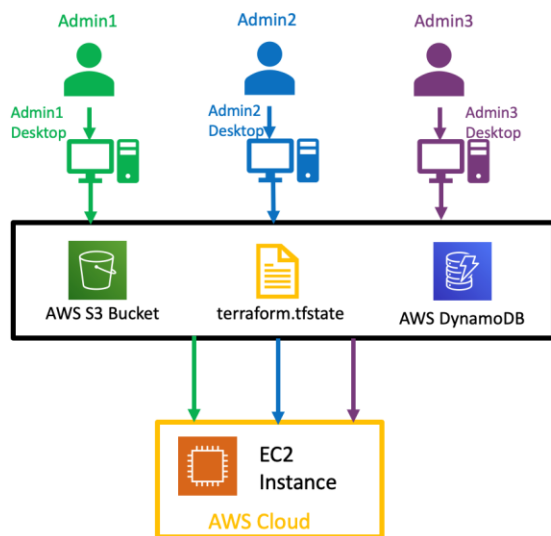
Local State File



Remote State File



Terraform Remote State File with State Locking



Not all backends support State Locking. AWS S3 supports State Locking

State locking happens automatically on all operations that could **write state**.

If state locking **fails**, Terraform **will not continue**.

You can **disable** state locking for most commands with the **-lock** flag but it is **not recommended**.

If acquiring the lock is taking **longer** than expected, Terraform will output a **status message**.

If Terraform doesn't output a message, state locking is still **occurring** if your backend supports it.

Terraform has a **force-unlock** command to manually unlock the state if unlocking failed.

Terraform Remote State File with State Locking

Terraform State Storage to Remote Backend

Terraform State Locking

```
# Terraform Block
terraform {
  required_version = "~> 0.14" # which means any version greater than 0.14
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Adding Backend as S3 for Remote State Storage
backend "s3" {
  bucket = "terraform-stacksimplify"
  key    = "dev/terraform.tfstate"
  region = "us-east-1"
}

# Enable during Step-09
# For State Locking
dynamodb_table = "terraform-dev-state-table"
```

Step-02: Create S3 Bucket

- Go to Services -> S3 -> Create Bucket
- Bucket name: terraform-on-aws-for-ec2
- Region: US-East (N.Virginia)
- Bucket settings for Block Public Access: leave to defaults
- Bucket Versioning: Enable
- Rest all leave to defaults
- Click on Create Bucket
- Create Folder

- Folder Name: dev
- Click on Create Folder
- Create Folder
 - Folder Name: dev/project1-vpc
 - Click on Create Folder

Step-03: Terraform Backend Configuration

- Reference Sub-folder: terraform-manifests
- [Terraform Backend as S3](#)
- Add the below listed Terraform backend block in Terraform Settings block in main.tf

Adding Backend as S3 for Remote State Storage

```
backend "s3" {
  bucket = "terraform-on-aws-for-ec2"
  key    = "dev/project1-vpc/terraform.tfstate"
  region = "us-east-1"
```

Enable during Step-09

For State Locking

```
dynamodb_table = "dev-project1-vpc"
}
```

Step-04: Terraform State Locking Introduction

- Understand about Terraform State Locking Advantages

Step-05: Add State Locking Feature using DynamoDB Table

- Create Dynamo DB Table
 - Table Name: dev-project1-vpc
 - Partition key (Primary Key): LockID (Type as String)
 - Table settings: Use default settings (checked)
 - Click on Create

Step-06: Execute Terraform Commands

Initialize Terraform

```
terraform init
```

Observation:

Successfully configured the backend "s3"! Terraform will automatically use this backend unless the backend configuration changes.

Terraform Validate

terraform validate

Review the terraform plan

terraform plan

Observation:

1) Below messages displayed at start and end of command

Acquiring state lock. This may take a few moments...

Releasing state lock. This may take a few moments...

2) Verify DynamoDB Table -> Items tab

Create Resources

terraform apply -auto-approve

Verify S3 Bucket for terraform.tfstate file

dev/project1-vpc/terraform.tfstate

Observation:

1. Finally at this point you should see the terraform.tfstate file in s3 bucket

2. As S3 bucket version is enabled, new versions of `terraform.tfstate` file new versions will be created and tracked if any changes happens to infrastructure using Terraform Configuration Files

Step-07: Destroy Resources

- Destroy Resources and Verify Bucket Versioning

Destroy Resources

terraform destroy -auto-approve

Clean-Up Files

rm -rf .terraform*

rm -rf terraform.tfstate* # This step not needed as we are using remote state storage here

Step-08: Little bit theory about Terraform Backends

- Understand little bit more about Terraform Backends
- Where and when Terraform Backends are used ?
- What Terraform backends do ?
- How many types of Terraform backends exists as on today ?

Terraform Backends

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

Where Backends are Used

Backend configuration is only used by **Terraform CLI**.

Terraform Cloud and **Terraform Enterprise** always use their **own state storage** when performing **Terraform runs**, so they ignore any **backend block** in the configuration.

For **Terraform Cloud** users also it is always **recommended** to use **backend block** in Terraform configuration for commands like **terraform taint** which can be executed only using Terraform CLI

Terraform Backends

What Backends Do

There are two things backends will be used for

1. Where state is **stored**
2. Where **operations** are performed.

Store State

Terraform uses **persistent state data** to keep track of the resources it manages.

Everyone working with a given collection of infrastructure resources must be able to **access** the **same** state data (**shared state storage**).

State Locking

State **Locking** is to **prevent conflicts and inconsistencies** when the operations are being performed

What are Operations ?
terraform apply
terraform destroy

Operations

"Operations" refers to **performing API requests** against infrastructure services in order to **create, read, update, or destroy** resources.

Not every terraform subcommand performs API operations; many of them only **operate on state data**.

Only two backends actually perform operations: **local** and **remote**.

The **remote** backend can perform API operations remotely, using **Terraform Cloud** or **Terraform Enterprise**.

Terraform Backends

Backend Types

Enhanced Backends

Enhanced backends can both **store state** and **perform operations**. There are only two enhanced backends: **local and remote**

Example for Remote Backend
Performing Operations : Terraform **Cloud**, Terraform Enterprise

Standard Backends

Standard backends **only store state**, and **rely** on the local backend for performing operations.

Example: AWS S3, Azure RM, Consul, etcd, gcs http and many more

References

- [AWS S3 Backend](#)
- [Terraform Backends](#)
- [Terraform State Storage](#)
- [Terraform State Locking](#)
- [Remote Backends - Enhanced](#)