Terraform: Remote State in Terraform

- Terraform is able to find the resources it created and update them accordingly. We have seen Terraform Plan and terraform apply commands.
- Terraform records information about what infrastructure it created in a Terraform state file.
  - File called **terraform.tfstate**
  - Terraform also maintain the back-up of earlier statefile in file named **terraform.tfstate.backup**
  - On command **Terraform Apply** terraform backup is written and new state file created.

- If the remote state is changed and user executes terraform apply again. Terraform will make the changes to meet the correct remote state.
- Problems while team is working on terraform -
- **Shared storage for state files** - To be able to use Terraform to update your infrastructure, each of your team members needs access to the same Terraform state files. That means you need to store those files in a shared location.
- **Locking state files** - As soon as data is shared, you run into a new problem: **locking**. Without locking, if two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

- **Isolating state files** - When making changes to your infrastructure, it's a best practice to isolate different environments.
- Most common technique for allowing multiple team members to access a common set of files is to put them in version control (e.g. Git). But this is a bad idea.
- **Manual error**: It's too easy to forget to pull down the latest changes from version control before running Terraform or to push your latest changes to version control after running Terraform.
- **Locking**: Most version control systems do not provide any form of locking that would prevent two team members from running **terraform apply** on the same state file at the same time.

- **Secrets**: All data in Terraform state files is stored in plain text. This is a problem because certain Terraform resources need to store sensitive data.
- Instead of using version control, the best way to manage shared storage for state files is to use Terraform's built-in support for **remote backends**. A Terraform *backend* determines how Terraform loads and stores state.

- **Remote Backend**: Remote Backend solves all problems we listed earlier.
- **Manual error**: Once you configure a remote backend, Terraform will automatically load the state file from that backend every time you run plan or apply and it'll automatically store the state file in that backend after each apply, so there's no chance of manual error.
- **Locking**: Most of the remote backends natively support locking. To run terraform apply, Terraform will automatically acquire a lock; if someone else is already running apply, they will already have the lock, and you will have to wait.
- **Secrets:** Most of the remote backends natively support encryption in transit and encryption on disk of the state file.

- Store state in S3 Bucket:

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = “us-east-1"
  }
}
```

- While using AWS S3 as backend, it’s recommended to use AWS configure instead of AWS creds in variables.

Will see you in Next Lecture…

See you in next lecture …