Terraform helps us build, evolve, and manage our infrastructure using its configuration files across multiple providers. We have knowledge about the basics of terraform from [previous article](https://medium.com/@mitesh_shamra/infrastructure-as-a-code-with-terraform-e7021bf28d7d).

When we build infrastructure with terraform configuration, a state file gets created in the local workspace directory named “terraform.tfstate”. This state file contains information about the provisioned infrastructure which terraform manage. Whenever we change the configuration file, it automatically determines which part of your configuration is already created and which needs to be changed with help of state file.

State helps provides idempotence to terraform as it already knows if one resource is present and prevent it to be created again when the same configuration executes.



State file data when we created a EC2 instance

State file data (specified above) provides all details regarding EC2 instance we have created using terraform. In the same way, we store the content of all other resources created by terraform which can be used later during modification of resources.

State file is essential for working with terraform which has below benefits:

1. Real world mapping of resources.
2. Track metadata of resources such as IP address of EC2 instance, its dependency with respect to other resources etc.
3. Cache resource attributes.
4. Enables collaboration among teams.

***Teams uses terraform as a key part of infrastructure change management and deployment pipeline.***

***Network team works*** on **networking aspects of infrastructure** whereas ***application team works*** on ***managing instances*** in given ***network space***. In the same way, ***other teams work on provisioning*** other ***aspects of infrastructure*** which are ***managed by the same terraform configuration***. As multiple members of a team work on provisioning infrastructure, they all need to have the **same state file**. Use of local file makes terraform usage hard among team members because every user needs to have the latest state file data before running terraform configuration. Everyone among team needs to know the current state of infrastructure so they can create what is not present and modify what is present to achieve desired result. Team members need to make sure that no two members run terraform configuration at the same time to prevent corruption, data loss, inconsistent state.

Terraform state can contain sensitive data of resources generated by terraform. When we use local state file, the state is stored in JSON format which is easy to understand by humans.

The remote state can help prevent most of above mentioned problems, where state file is stored at some remote datastore like S3, consul etc.

Benefits of remote state:

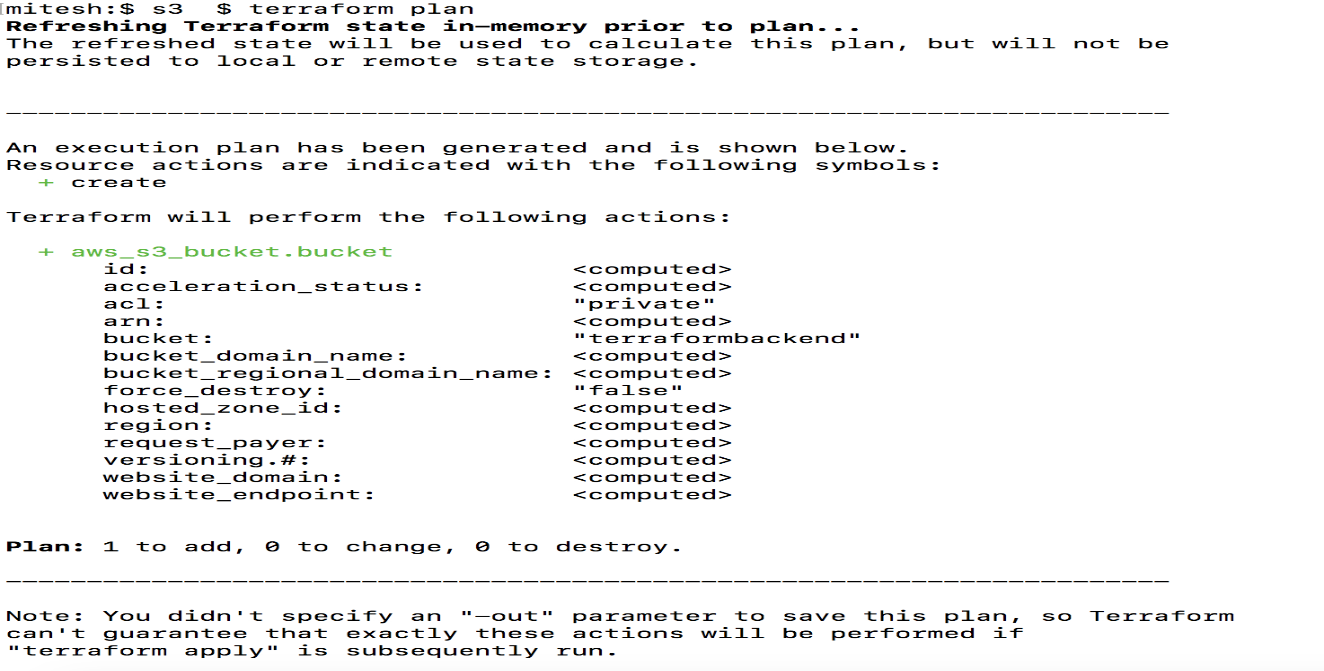
1. Safer storage: Storing state on the remote server helps prevent sensitive information. State file remains same but remote storage like S3 provides a layer to security like making S3 bucket private and giving limited access.
2. Auditing: Invalid access can be identified by enabling logging.
3. Share data: Remote storage helps share state file with other members of team.

This allows us to break our infrastructure in different components which are managed by different teams, as everyone shares a common view of infrastructure.

Let’s us use S3 as our remote storage for terraform configuration to create EC2 instance. To use S3 as a backend, we first need to create an S3 bucket, let’s call it “terraformbackend”.

**Terraform configuration to create S3:**

resource "aws\_s3\_bucket" "bucket" {  
 bucket = "terraformbackend"  
}



We need to define provider with access and secret key for all separate terraform configurations:

provider "aws" {  
 access\_key = "ACCESS\_KEY\_HERE"  
 secret\_key = "SECRET\_KEY\_HERE"  
 region = "us-east-2"  
}

Once we have our S3 bucket ready, let’s setup S3 as our remote backend by adding below code in our existing terraform.tf code.

terraform {  
 backend "s3" {  
 bucket = "terraformbackend"  
 key = "terraform"  
 region = "us-east-2"  
 }  
}

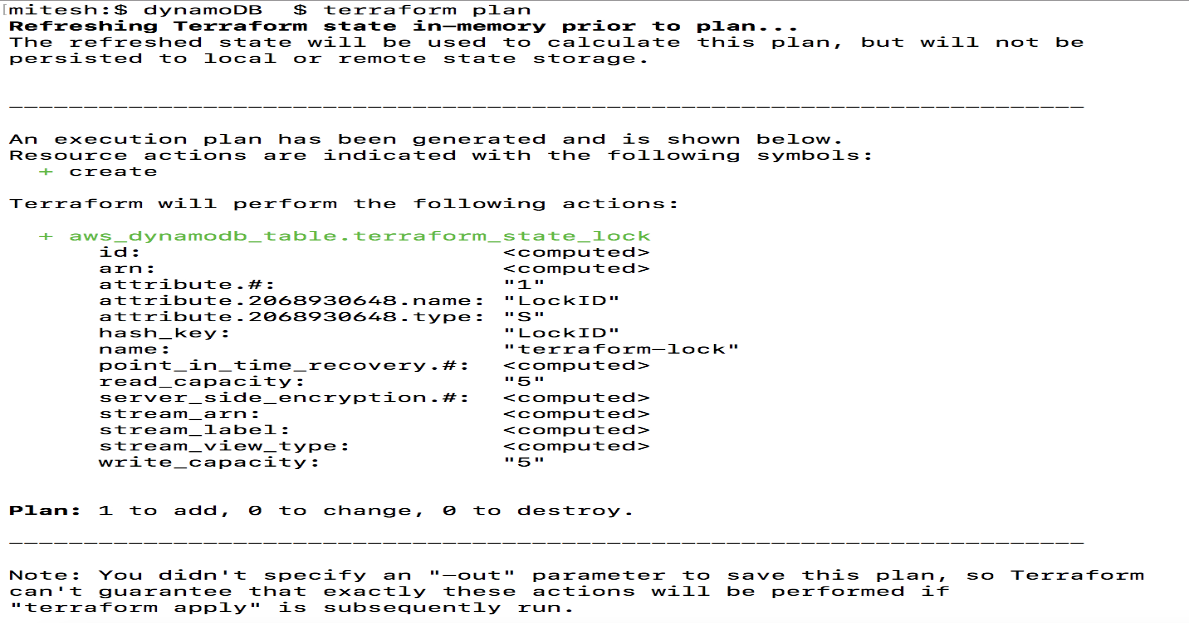
With this code, we have informed terraform to use backend as S3 with bucket name to be “terraformbackend” to store its state. The path to the state file inside the bucket can be defined using the key. In our case, state file name is “terraform” and located in region “us-east-2”. There are more parameters to tune our backend which can be found [here](https://www.terraform.io/docs/backends/types/s3.html). With S3 backend, we need to define an IAM user with ListBucket permissions and permission to GetObject and PutObject in our “terraformbackend” s3 bucket. Once we are done with our configuration, we need to run “terraform init” to make sure S3 backend comes into play.

Terraform provides locking to prevent concurrent runs against the same state. Locking helps make sure that only one team member runs terraform configuration. Locking helps us prevent conflicts, data loss and state file corruption due to multiple runs on same state file.

[DynamoDB](https://aws.amazon.com/dynamodb/) can be used as a locking mechanism to remote storage backend S3 to store state files. The DynamoDB table is keyed on “LockID” which is set as a bucketName/path, so as long as we have a unique combination of this we don’t have any problem in acquiring locks and running everything in a safe way.

To use DynamoDB as a locking mechanism, we first need to create a dynamoDB table, let’s call it “terraform-lock”.

resource "aws\_dynamodb\_table" "terraform\_state\_lock" {  
 name = "terraform-lock"  
 read\_capacity = 5  
 write\_capacity = 5  
 hash\_key = "LockID"attribute {  
 name = "LockID"  
 type = "S"  
 }  
}

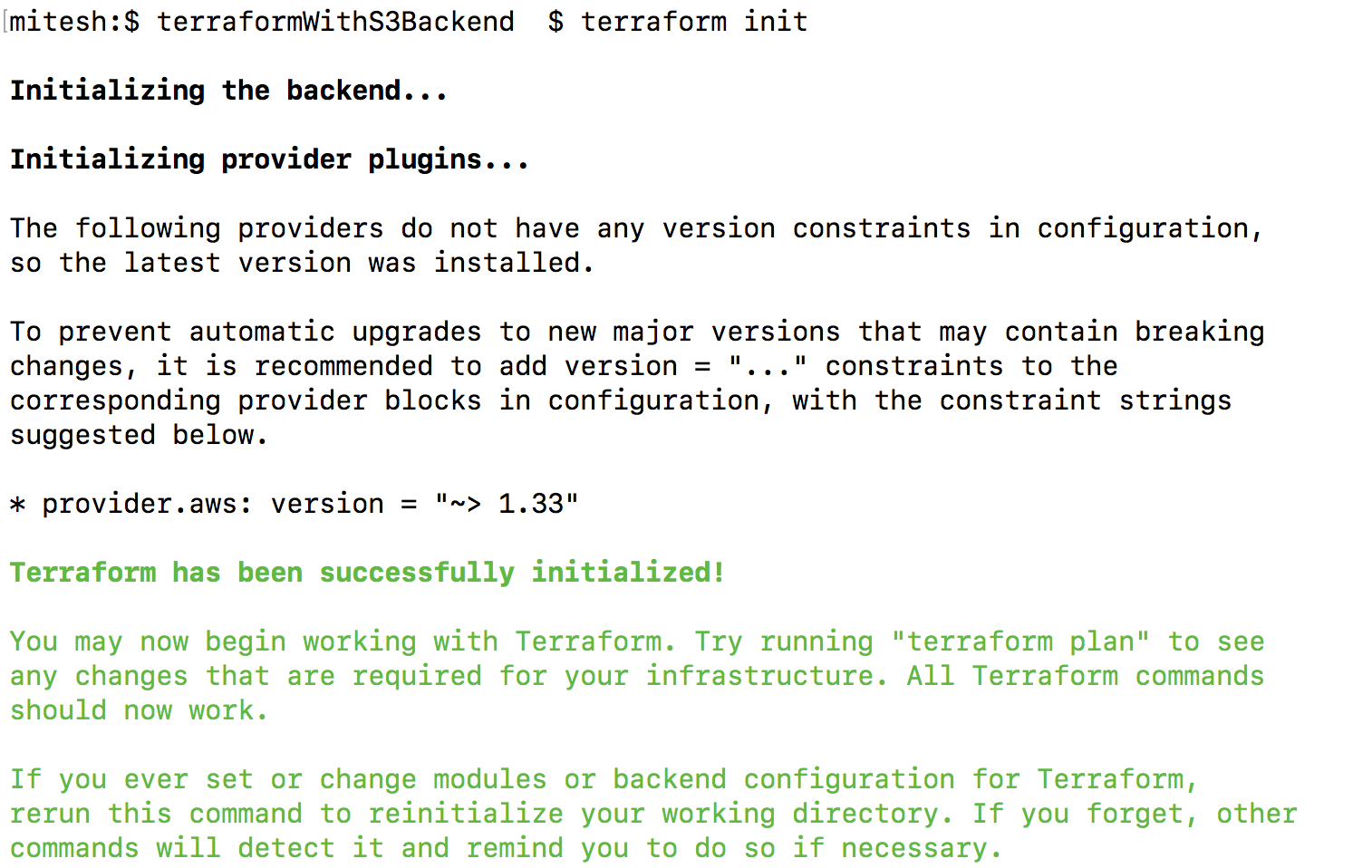


This terraform code is going to create a dynamo DB table with name “terraform-lock” with key type string named “LockID” which is also a hash key.

Once we have our dynamo DB table “terraform-lock” ready, let’s setup dynamo DB as our locking mechanism with s3 remote backend by adding below code in our existing terraform.tf code:

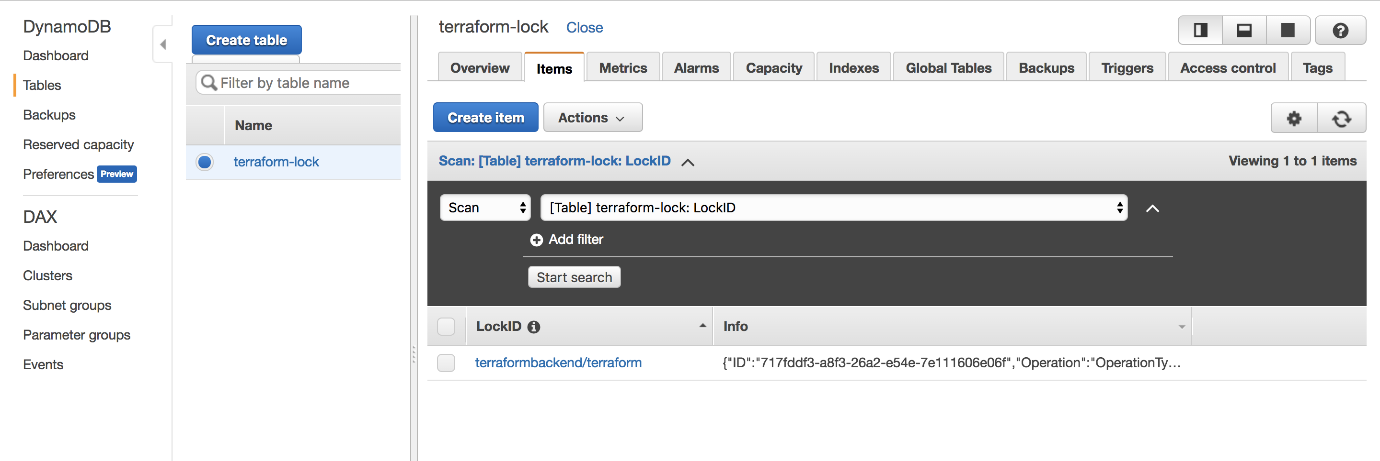
terraform {  
 backend "s3" {  
 bucket = "terraformbackend"  
 key = "terraform"  
 region = "us-east-2"  
 dynamodb\_table = "terraform-lock"  
 }  
}

Once this is added, we need to run “terraform init” to make sure our backend is initialised properly for usage by terraform.

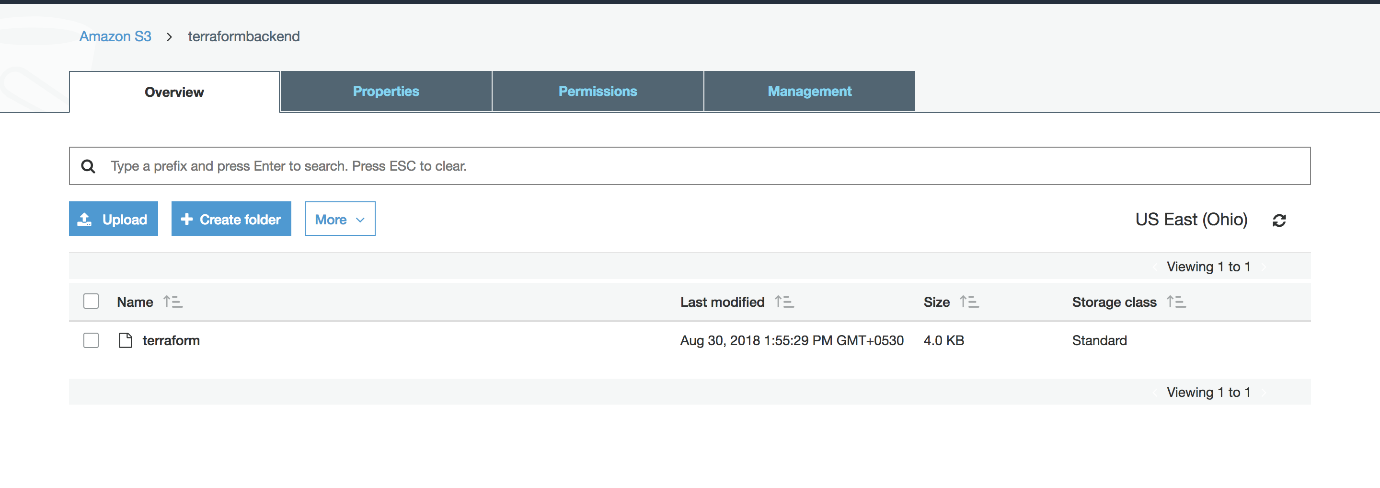


Backend init being part of terraform init command

With this, we can use S3 as our storage and dynamoDB as our locking mechanism. When we run “terraform apply”, it first acquires lock on dynamo db using key as “terraformbackend/terraform” with some unique value, once lock is acquired, then it start applying changes on infrastructure and then store state file on S3.



LockID column has lock key named “terraformbackend/terraform” for acquiried lock



State data is stored in file named “terraform”