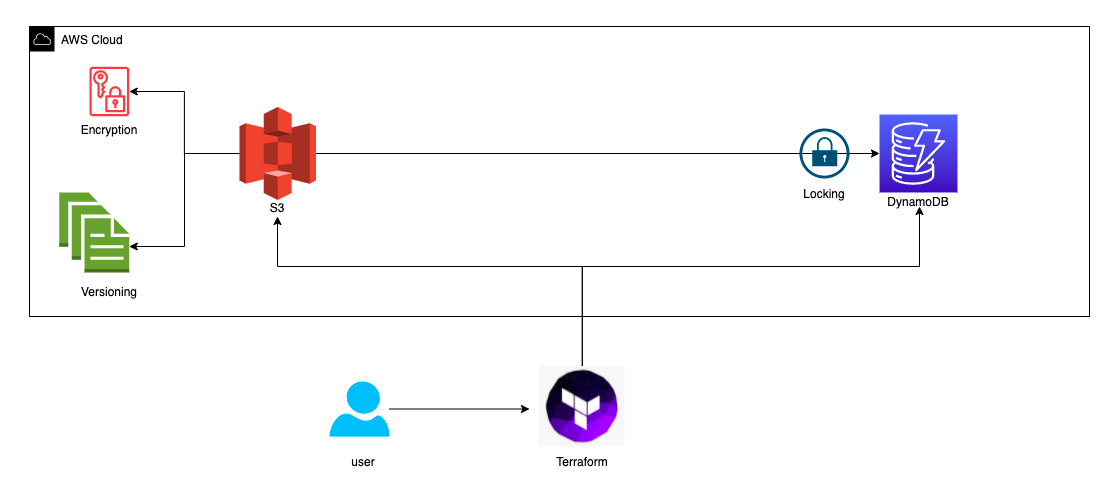
**AWS S3 and DynamoDB remote state locking and storage with Versioning and Encryption**

When a terraform stack is deployed, terraform creates a state file.

The state file keeps track of what resources have been deployed, all parameters, IDs, dependencies, failures, and outputs defined in your stack.

In a not so distant past, this state file (JSON) would be committed to the repository containing your terraform stack code.

Few problems if we commit a state file to a repository:



**Concurrency:** If 2 or more developers are working in the stack, they won’t see the other state until it’s pushed to the repository

**Automated Deployment:** A CI tool that deploys the stack automatically would need to commit the new state file to the repository

**Easily corruptible:** In case of a merge conflict or human error, the state file can be corrupted or gone.

Here is how we can fix it:

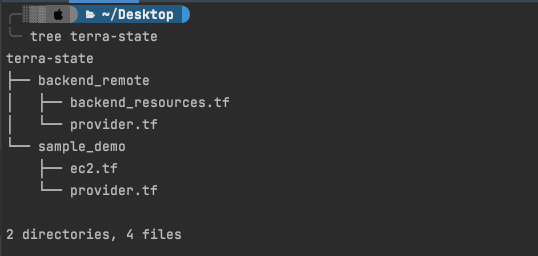
Instead of having a local JSON file holding the state, the state file is uploaded to an S3 bucket.

And when two or more users work on the infrastructure simultaneously, problems may arise in the creation of the resources, as there will be a situation such as executing another process before the state is finalised. Therefore, Terraform can lock your state to prevent other users from breaking our infrastructure using the same state file at the same time. However, not every backend supports this feature. But *AWS S3 bucket* supports this feature with the *Amazon DynamoDB* table.

So, we can use the locking feature using *AWS S3* and *DynamoDB* table. In other words, while a user is working on the infrastructure with Terraform, another user cannot work on the same state file simultaneously

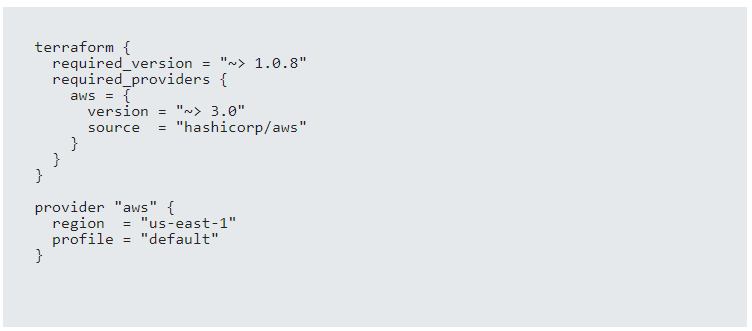
Our showcase

Now, let’s create a folder named terra\_state. Then enter the folder and create two folder names, sample\_demo, and backend\_remote. Next, enter the sample\_demo and create two files with names and create a file name provider.tf & ec2.tf in the backedn\_remote folder create two files with names provider.tf & backend\_resources.tf.

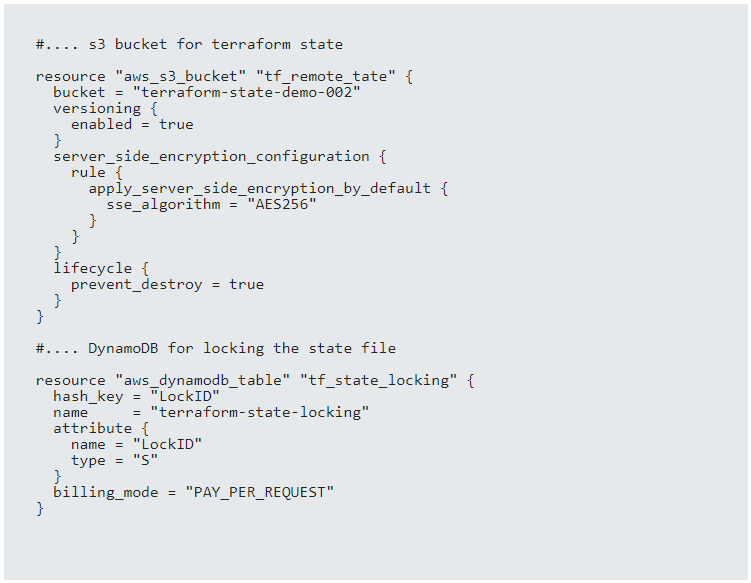


If you want to use S3 as a backend in Terraform, first, you must create an S3 bucket and then specify that bucket in your config file as backend. Now we create our S3 bucket for remote state and Amazon DynamoDB table for Locking state. Go to the backend\_remote folder and add the following to the respective .tf file.

backend\_remote/provider.tf



backend\_remote/backend\_resources.tf



b**ucket:** It must be the name of the bucket. Our bucket name must be *unique*. So you can add some words or letters to make it a unique

**lifecycle (prevent\_destroy):** This argument prevents us from accidentally or wrongly deleting the bucket.

**versioning:** This allows us to keep all versions of our objects.

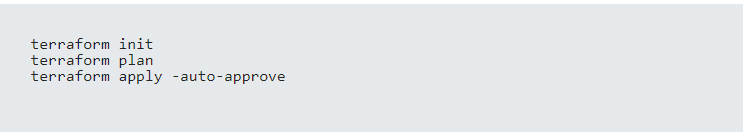
**server\_side\_encryption\_configuration:** You can use encryption for stored data in the S3 bucket.

**DynamoDB table (Locking)**

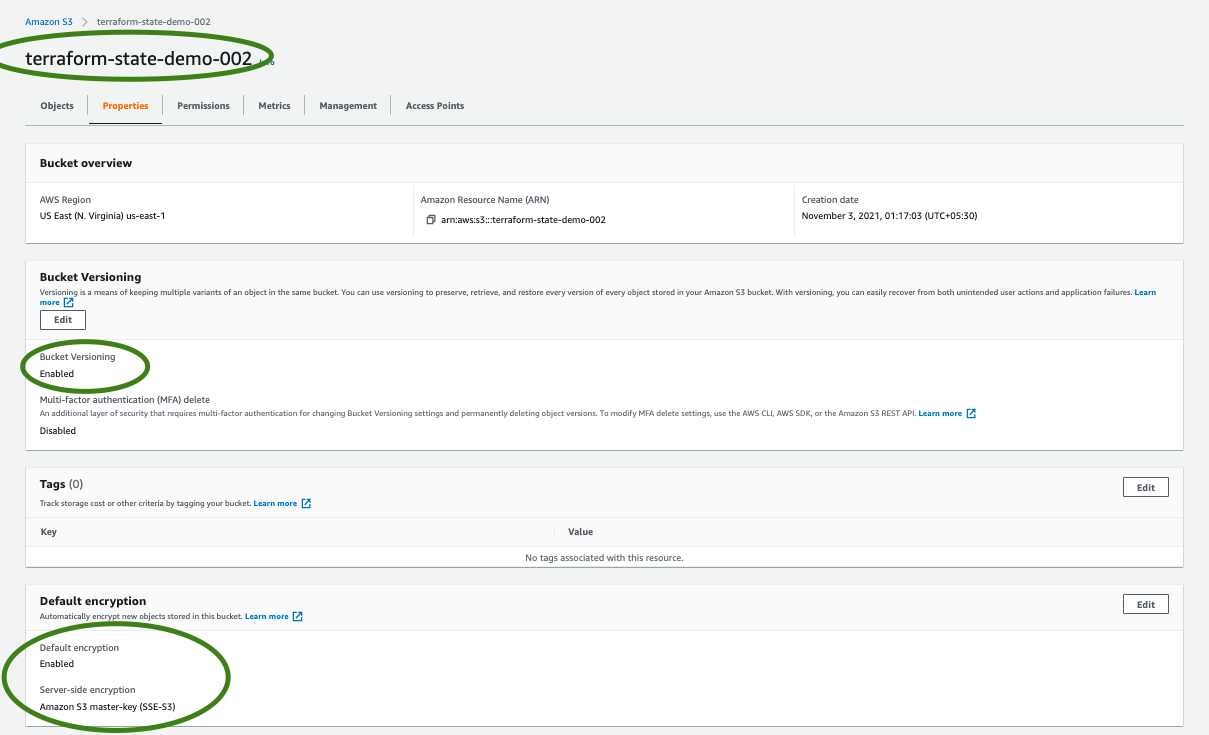
If we are using the DynamoDB table for locking, the primary key must be **LockID** (type: string). Otherwise, the locking feature will not work.

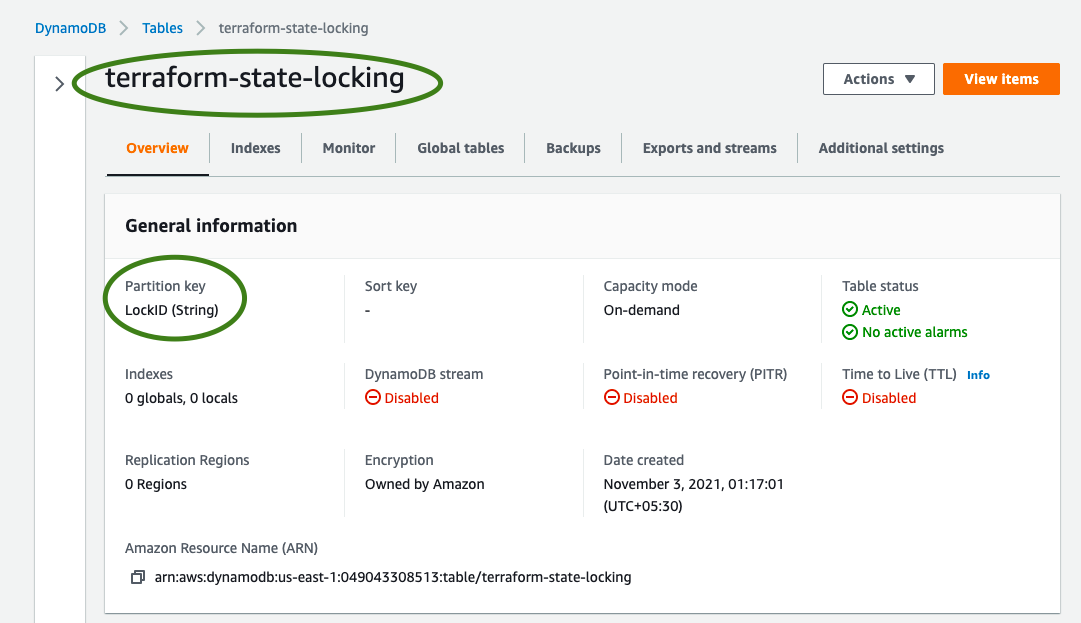
So, we use the arguments hash\_key and attribute as above.

Now, in the backend-remote folder, run the below commands First, initialize Terraform, plan & then create an S3 bucket and DynamoDB table.



Once the creation is complete go to the AWS management console and check the S3 bucket and DynamoDB.





Now, we can create our infrastructure and make the state in the S3 backend and lock. Go to the sample\_demo folder and add the following to the provider.tf & ec2.tf.

sample\_demo/provider.tf



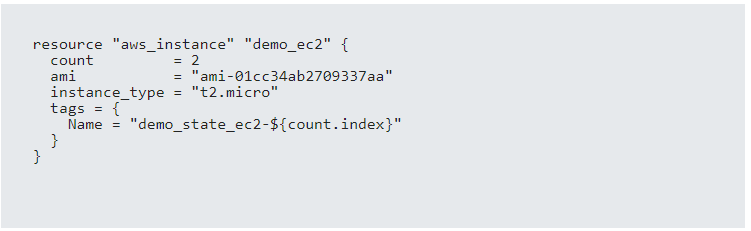
**bucket:** Name of S3 bucket we just created.

**key:** Path of our Terraform state in S3 bucket

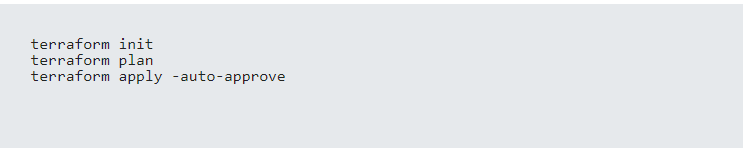
**dynamodb table:** Name of the DynamoDB table we just created for locking.

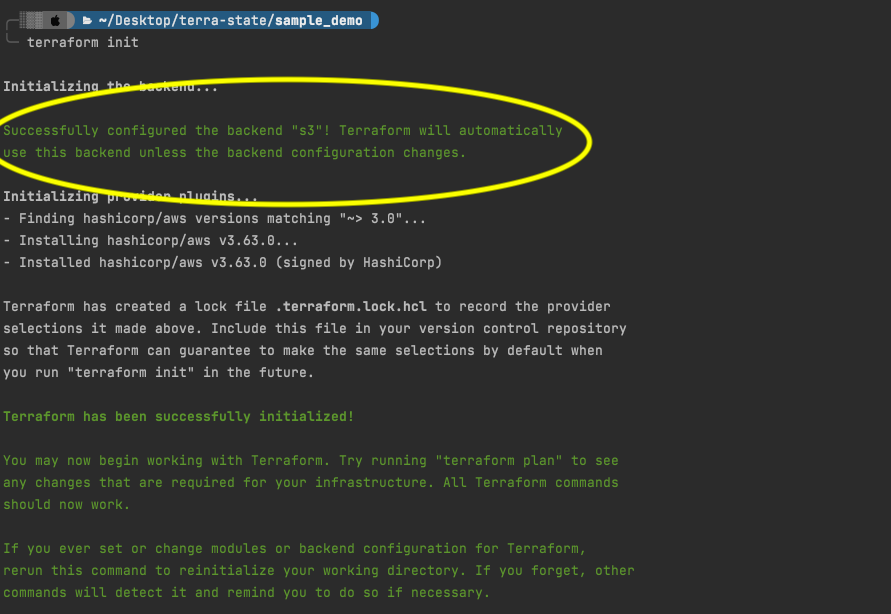
**encrypt:** Enable encryption

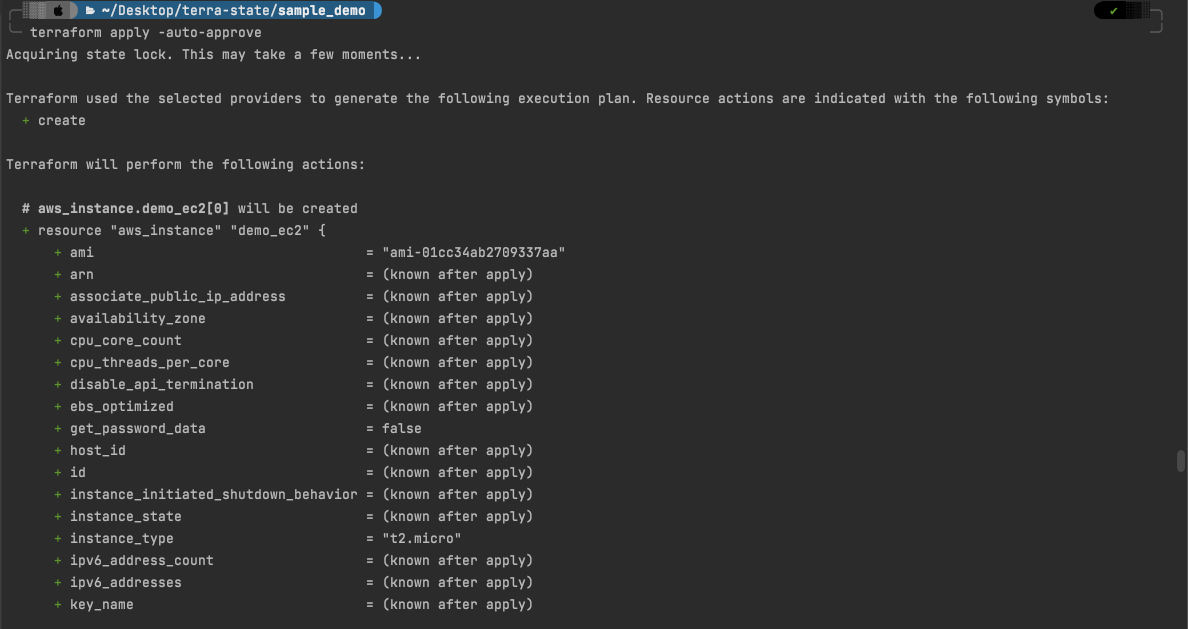
sample\_demo/ec2.tf



Next, run the commands belove in the sample\_demofolder. Initialize Terraform and create an AWS EC2 instance using the S3 backend and lock.





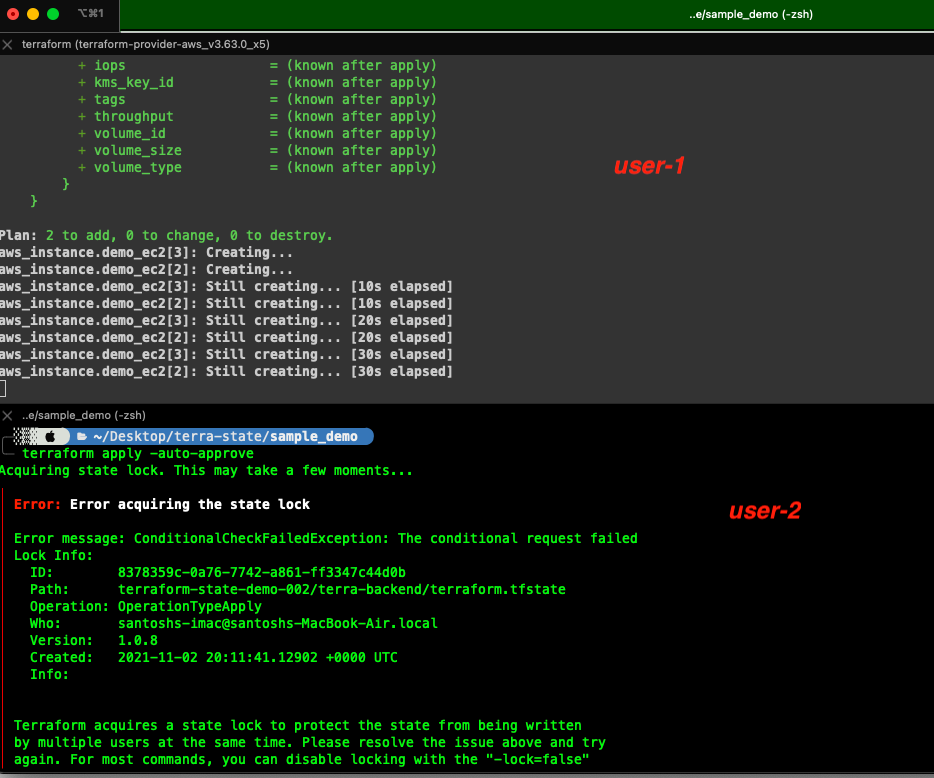


once the creation is complete the state file will be stored in the s3 bucket.

It’s time to try the locking and we will simulate two different users working on the same state file simultaneously. Then see if the state lock.

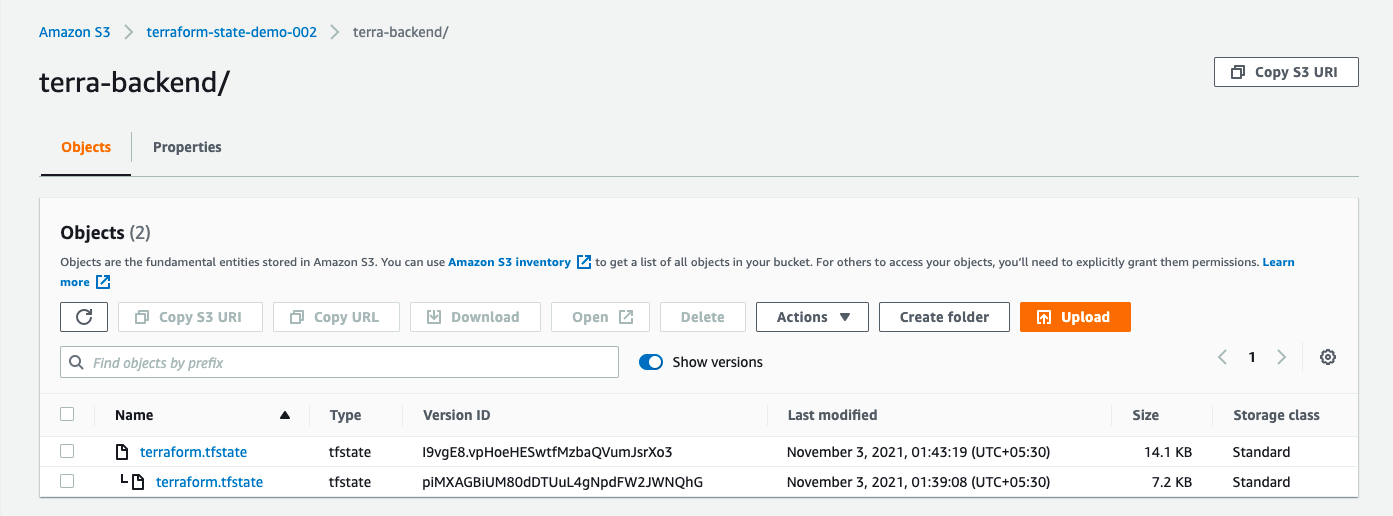
go to the sample\_demo/ec2.tf and increase the count to 4. It is just an example of making any change on the config file.

Now we will simulate two different users are trying to access the terraform state file and for one user he will be able to access the state file and for others, it will be locked.

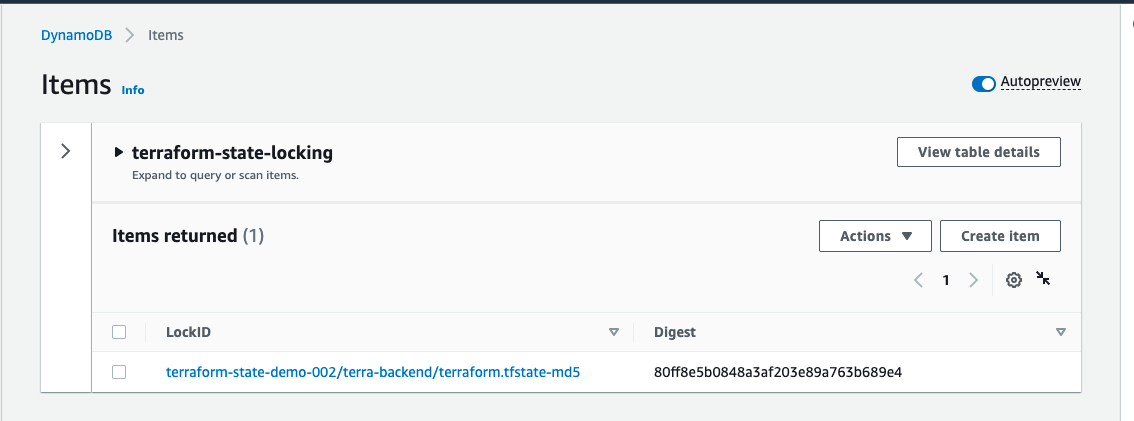


As we can see, in our case the user 1 ran command first, the command worked, and it succeeded. However, user 2 was not allowed to run because the state was locked.

Now we go to the s3 and dynamo DB to see the state file and locking.



the S3 bucket started to take versions of the state



In the end, don’t forget to run the terraform destroy command. Also, as we use the prevent destroy argument while creating the S3 bucket, the S3 bucket, and DynamoDB table will not be deleted. So you can delete them from the AWS console.