**Terraform**

Terraform is a command line tool for creating, updating and versioning infrastructure in the cloud.

Terraform describes infrastructure in a file using the language called Hashicorp Configuration Language (HCL) with the extension of .tf

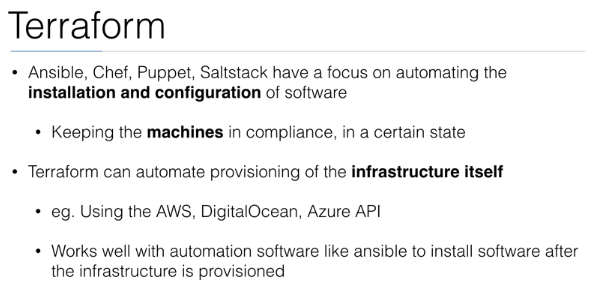
It is a declarative language that describes infrastructure in the cloud. When we write our infrastructure using HCL in .tf file.

Terraform generates an execution plan that describes what it will do to reach the desired state.

Once execution plan is ready, terraform executes the plan and generates a state file by the name terraform.tfstate by default.

This file maps resource Meta data to the actual resource ID and lets terraform knows what it is managing in the cloud.

* Chef, Ansible and puppet focus on installations and configurations of software, when we want to automate the machine itself then chef and ansible are better than terraform.
* We can use AWS and terraform to spin up the services and with the help of chef and ansible we can handle the installations and configuration updates on those machines.



**Installation:**

cd /usr/local/src

wget <https://releases.hashicorp.com/terraform/0.8.5/terraform_0.8.5_linux_386.zip>

unzip terraform\_0.8.5\_linux\_386.zip

mv terraform /usr/local/bin/

export PATH=$PATH: /root/terraform

Once done create a ec2 user on IAM

* Create a user
* Create a group and pass the permission as full access to EC2 instance via AWS management console
* Pass the tag name and create a EC2user

To deploy an EC2 instance through terraform create a file with extension .tf. This file contains namely two section.

* The first section declares the provider (in our case it is AWS). In provider section we will specify the access key and secret key that is written in the CSV file which we have downloaded earlier while creating EC2 user. Also choose the region of your choice.
* The resource block defines what resources we want to create. Since we want to create EC2 instance therefore we specified with "aws\_instance" and the instance attributes inside it like ami, instance\_type and tags.

Now create a terraform file with .tf extension vi aws.tf

provider "aws" {

access\_key = "ZKIAITH7YUGAZZIYYSZA"

secret\_key = "UlNapYqUCg2m4MDPT9Tlq+64BWnITspR93fMNc0Y"

region = "ap-southeast-1"

}

resource "aws\_instance" "example" {

ami = "ami-83a713e0"

instance\_type = "t2.micro"

tags {

Name = "your-instance"

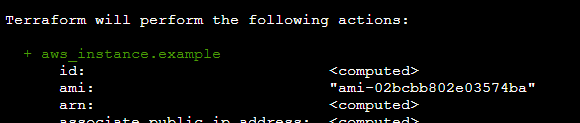
}

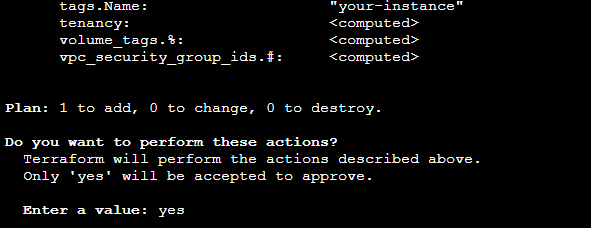
}

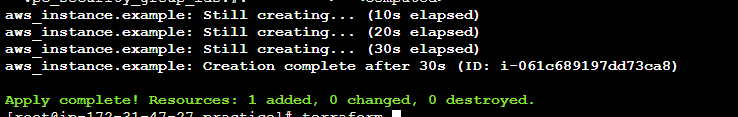
Now do terraform init

Terraform plan

Terraform apply







<https://linoxide.com/devops/install-terraform-provision-aws-ec2-instance/>

**Common commands:**

**apply Builds or changes infrastructure**

console Interactive console for Terraform interpolations

**destroy Destroy Terraform-managed infrastructure**

env Workspace management

fmt Rewrites config files to canonical format

get Download and install modules for the configuration

graph Create a visual graph of Terraform resources

import Import existing infrastructure into Terraform

**init Initialize a Terraform working directory**

output Read an output from a state file

**plan Generate and show an execution plan**

providers Prints a tree of the providers used in the configuration

push Upload this Terraform module to Atlas to run

refresh Update local state file against real resources

show Inspect Terraform state or plan

taint Manually mark a resource for recreation

untaint Manually unmark a resource as tainted

validate Validates the Terraform files

version Prints the Terraform version

workspace Workspace management

All other commands:

debug Debug output management (experimental)

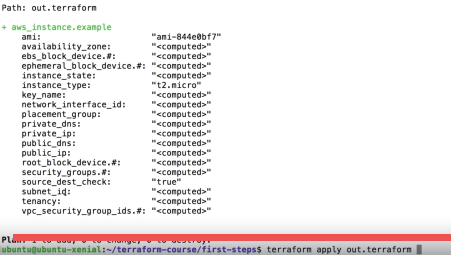
force-unlock Manually unlock the terraform state

**state Advanced state management**

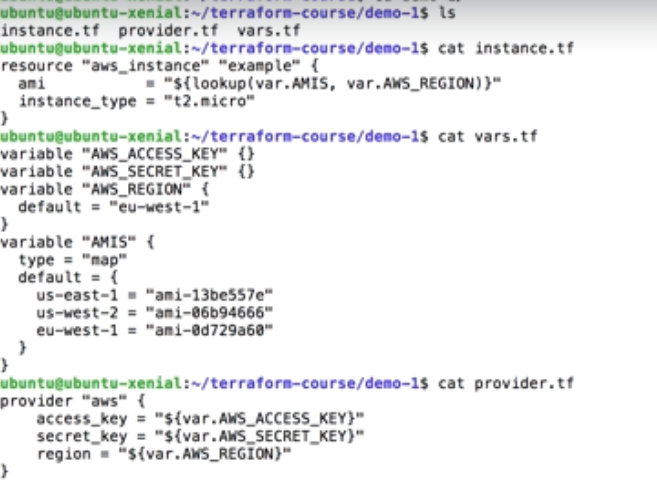
**Terraform init** will initialize the provider plugins (AWS), this command we always pass when we create a new directory.

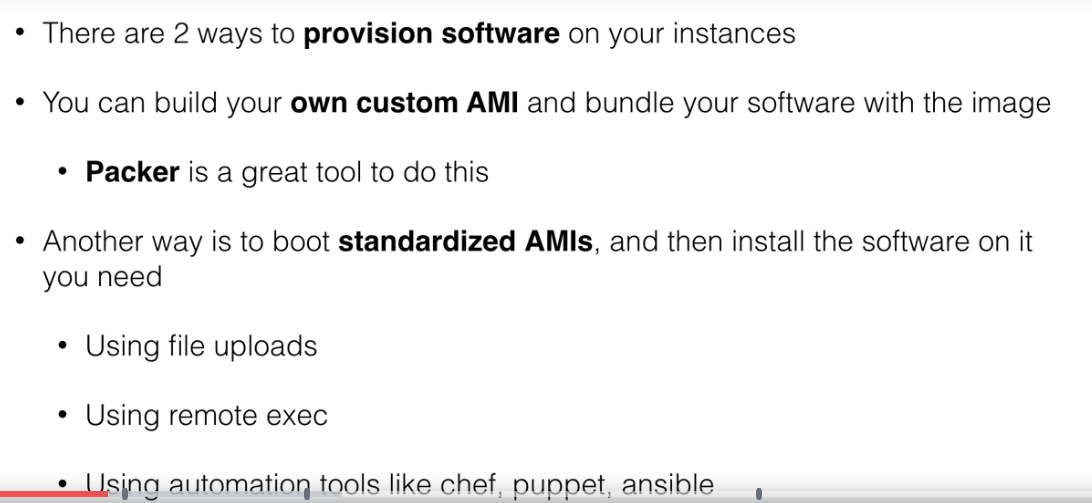
Terraform plan –out out.tf will save the changes from the main .tf file and save it to the out.tf and after that we can do terraform apply out.tf, so what it does is it make only changes to that part that we have passed in main .tf file

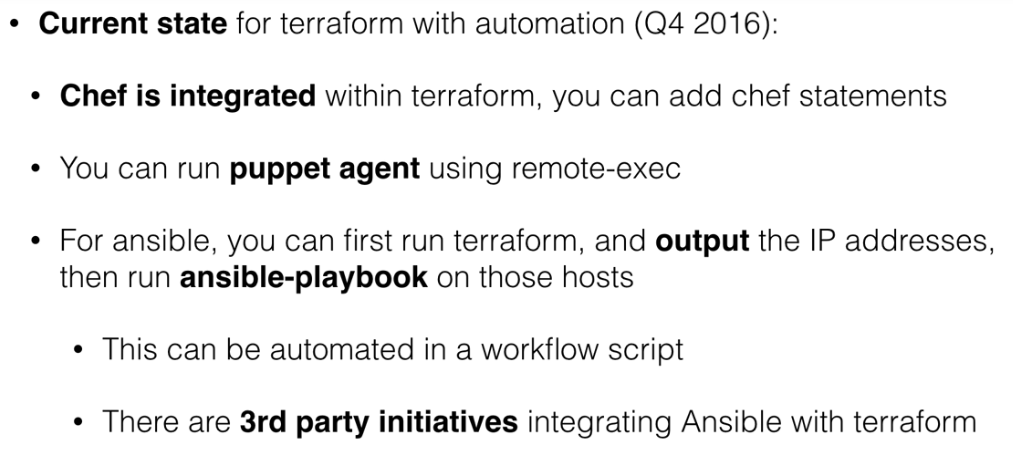
Or, terraform plan –out file; terraform apply file; rm file

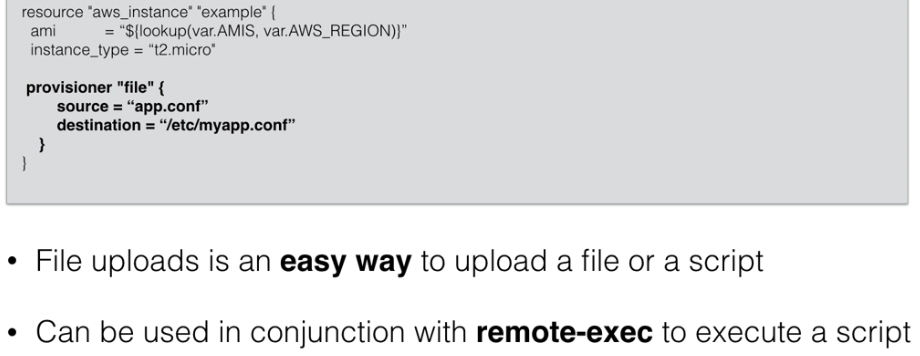


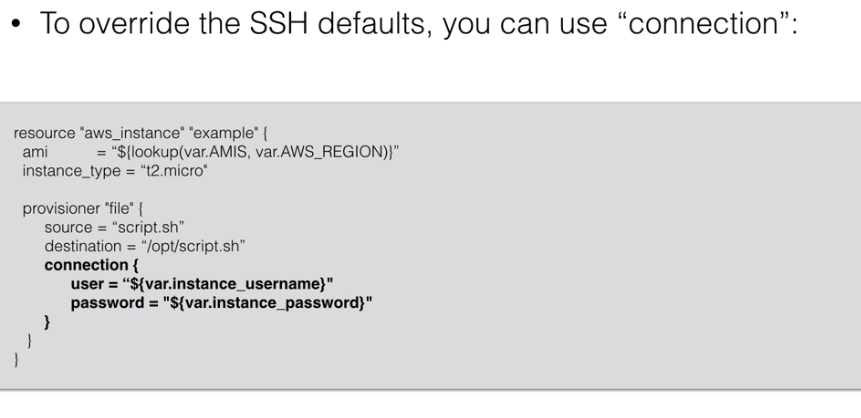
The output in the out.tf will be the changes that will be applied to the infrastructure, so it’s always safe to follow this process.

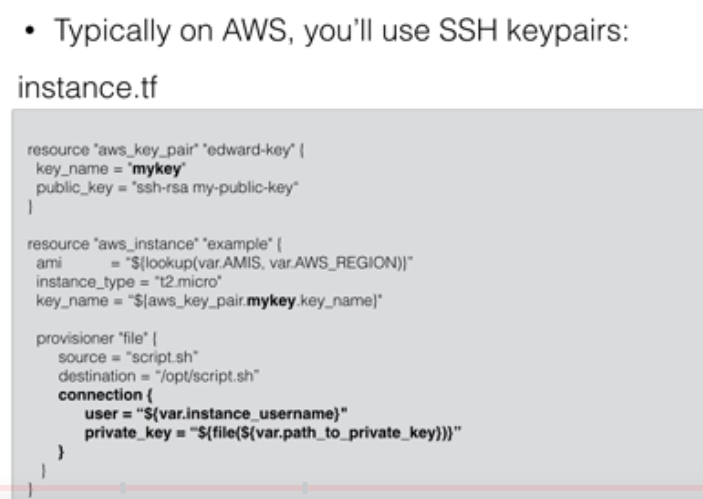


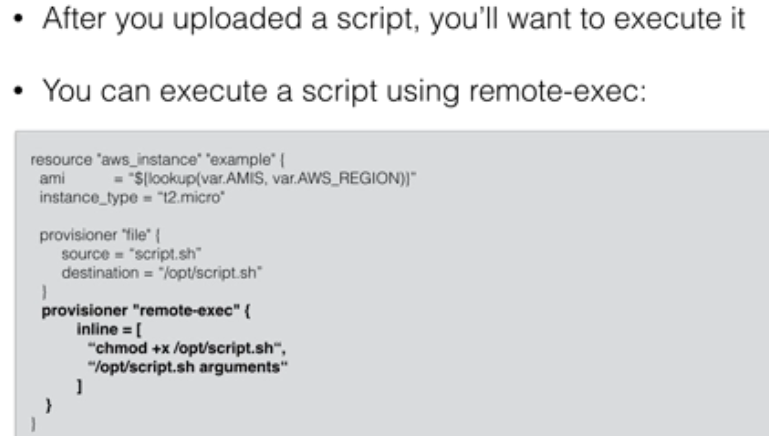






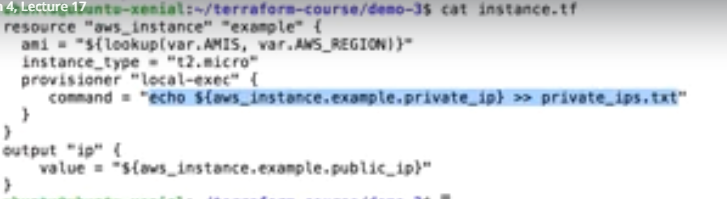


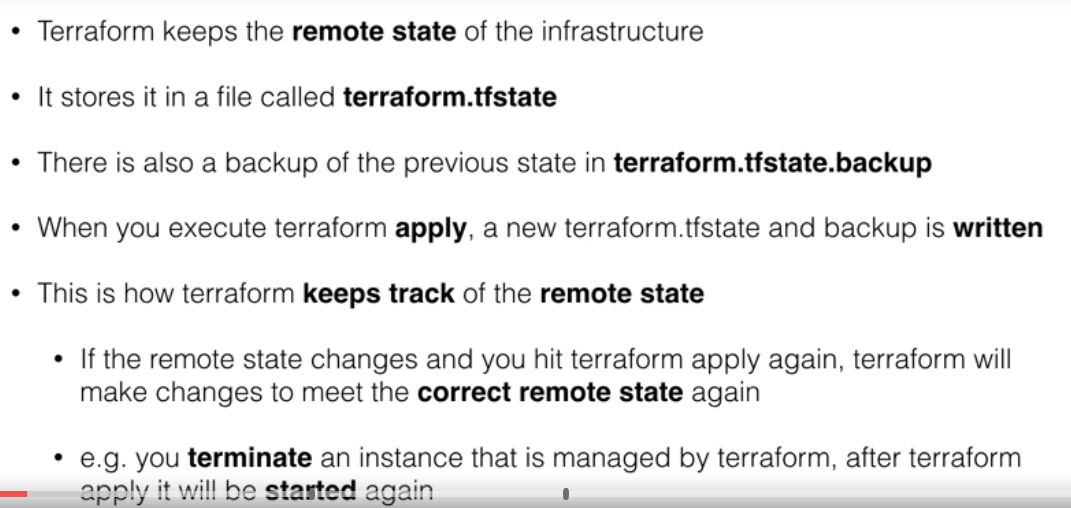




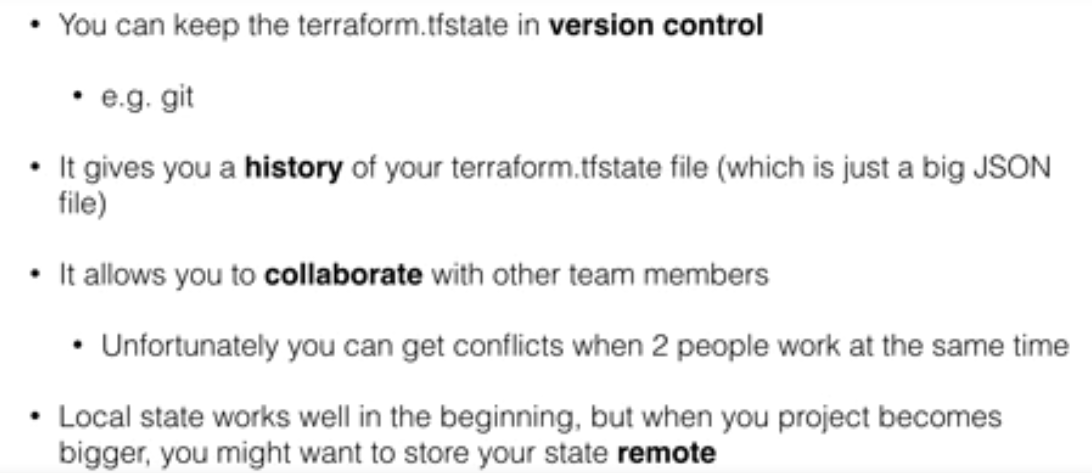
We can achieve provisioning through terraform as well.

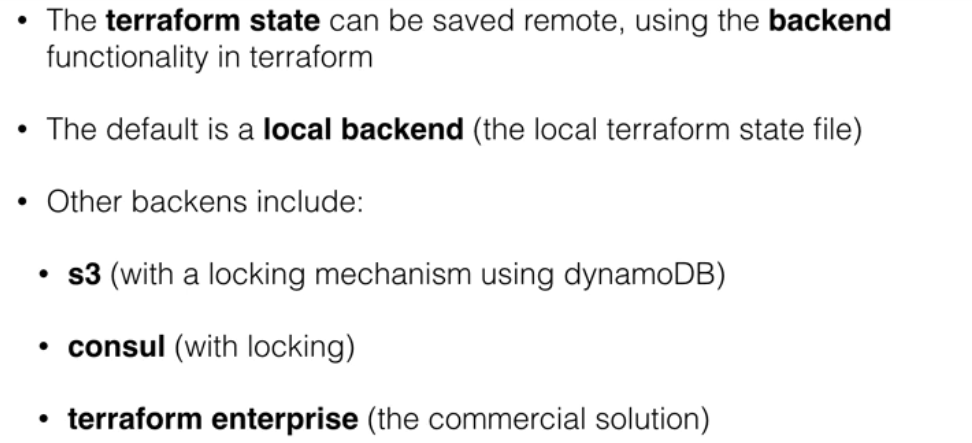
<https://wipro.udemy.com/learn-devops-infrastructure-automation-with-terraform/learn/v4/t/lecture/5933958?start=0>

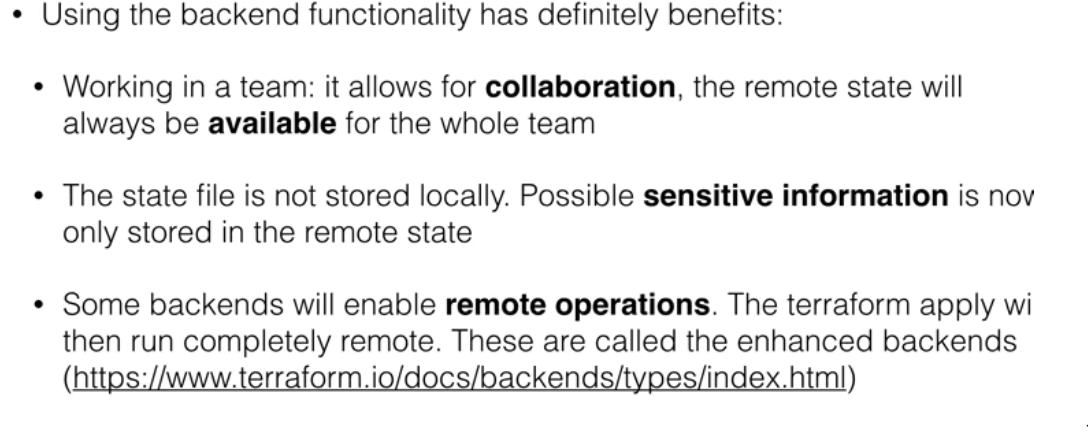


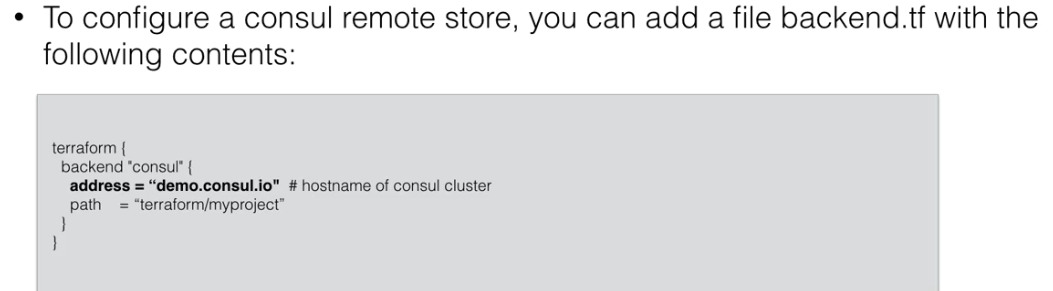


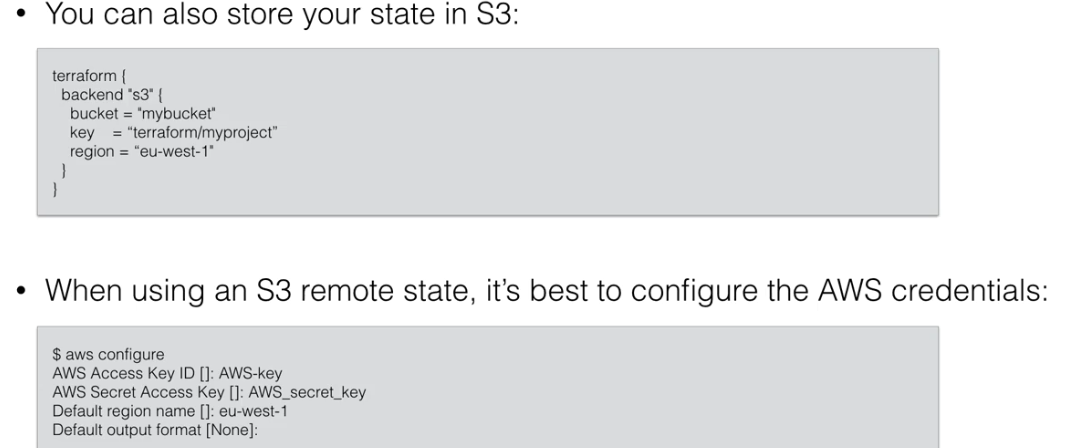
Let’s say we have terminated an instance manually and when we do terraform apply, it see that the instance doesn’t exist anymore, so what it does is that it will relaunch the instance.

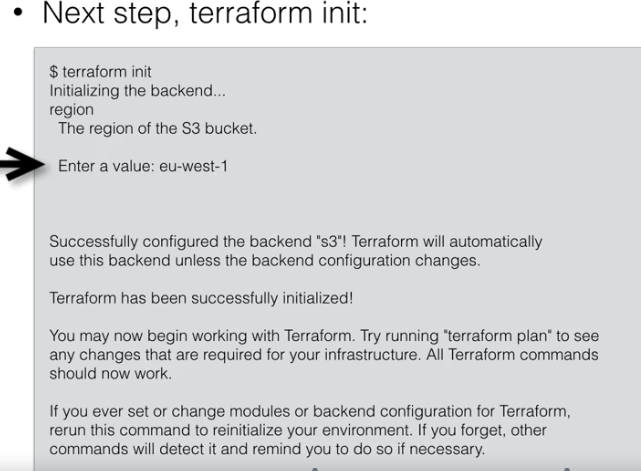




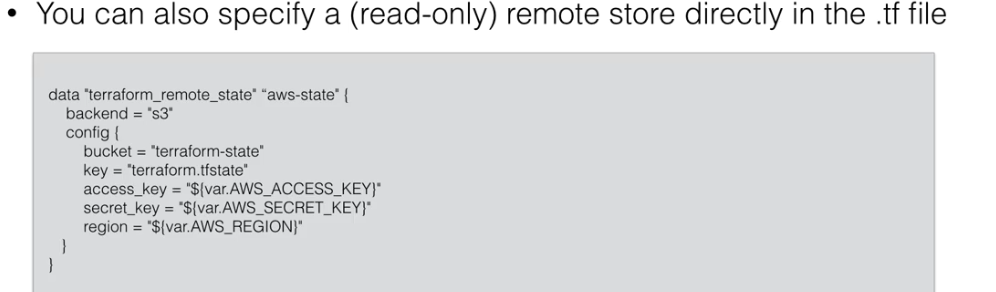








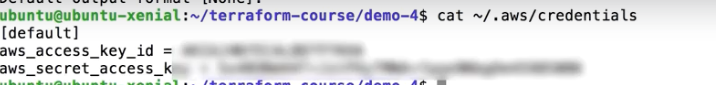




Remote state in S3

Create a bucket and enable versioning, versioning helps in recovery on files that deleted accidently.

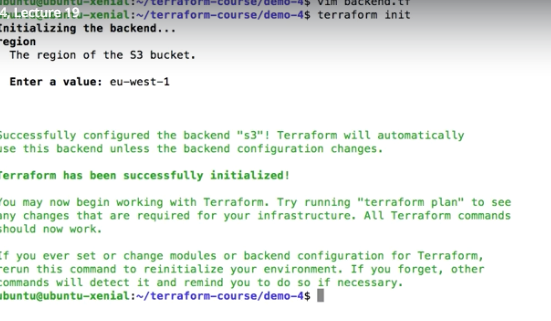
AWS stores the credentials in the below path, terraform will be looking for credentials at below path.



Create a file backend.tf



Now do terraform init and it will ask for region and once passed it will configure backend S3.



Then go ahead with teraform apply

State file will be created in the S3 bucket, which is nothing but Json file.