1	**After Installation Check Terrafom version**
	$ terraform version
	$ terraform -help
	Usage: terraform [-version] [-help] <command> [args]
	$ terraform -help apply or $ terraform apply -help
2	**Creating An AWS EC2 Instance**
	** The set of files used to describe infrastructure in Terraform is known as a Terraform configuration. You'll write your first configuration file to launch a single AWS EC2 instance.
	** Each configuration should be in its own directory. Create a directory ("terraform-aws") for the new configuration and change into the directory.
	Document Refenece:
	https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
	$ mkdir terraform-aws && cd terraform-aws && touch main.tf
	*** Create a file named **main.tf** for the configuration code and copy and paste the following content.*
	terraform { **required_providers {** **aws = {** **source = "hashicorp/aws"** **version = "3.73.0"** **}** **}** **}** **provider "aws" {** **region = "us-east-1"** **}** **resource "aws_instance" "tf-ec2" { ## We give the name "tf-ec2" as local name of resource for Terraform. This name won't be seen on AWS console but terraform state file.** **ami = "ami-0742b4e673072066f"** **instance_type = "t2.micro"** **tags = {** **"Name" = "created-by-tf"** **}** **}**

	Initialize the directory: When you create a new configuration you need to initialize the directory with terraform init.
	$ terraform init
	** Terraform downloads the aws provider and installs it in a hidden subdirectory (.terraform) of the current working directory. The output shows which version of the plugin was installed.
	**** Create infrastructure: run terraform plan. You should see an output similar to the one shown below.**
	$ terraform plan
	*** After this command (terraform plan) you will see an output that shows the execution plan, describing which actions Terraform will take in order to change real infrastructure to match the configuration.*
	** Run **terraform apply**. You should see an output similar to the one shown above.
	$ terraform apply
	** Terraform will wait for your approval before proceeding. If anything in the plan seems incorrect it is safe to abort (ctrl+c) here with no changes made to your infrastructure.
	** If the plan is acceptable, type "yes" at the confirmation prompt to proceed. Executing the plan will take a few minutes since Terraform waits for the EC2 instance to become available.
	** Visit the EC2 console to see the created EC2 instance.
	Inspect state
	** When you applied your configuration, Terraform fetched data from resources into a file called terraform.tfstate. It keeps track of resources' metadata.
	** Terraform has a command called **terraform state** for advanced state management. For example, if you have a long state file (detailed) and you just want to see the name of your resources, which you can get them by using the **list** subcommand.
	$ terraform state list
	aws_instance.tf-ec2
3	**Creating An AWS S3 Bucket**
	** Create a S3 bucket. Go to the **main.tf** and add the followings.

	terraform { **required_providers {** **aws = {** **source = "hashicorp/aws"** **version = "3.73.0"** **}** **}** **}** **provider "aws" {** **region = "us-east-1"** **}** **esource "aws_instance" "tf-ec2" {** **ami = "ami-0742b4e673072066f"** **instance_type = "t2.micro"** **key_name = "ec2_key" # write your pem file without .pem extension>** **tags = {** **"Name" = "tf-ec2"** **}** **}** **resource "aws_s3_bucket" "tf-s3" {** **bucket = "yalcin-tf-test-bucket-handson-1"** **acl = "private"** **}**
	** Write your pem file without .pem extension and change the "addwhateveryouwant" part of the bucket name. Because bucket name must be unique.
	** Run the command terraform plan and terraform apply.
	terraform apply -auto-approve
	** **auto-approve** means to skip the approval of plan before applying.
	terraform plan -out=justs3
	** Now we have just an S3 bucket in justs3. Check that terraform.tfstate file has both ec2 and s3 bucket (real infrastructure). If we apply justs3 file it will delete the EC2 instance and modify the tfstate file. You can save your plans with -out flag.
	terraform apply justs3
4	**Terraform Commands**

	** ***Validate command:*** *Go to the terminal and run terraform validate. It validates the Terraform files syntactically correct and internally consistent.*
	$ terraform validate
	** **fmt command: Format command & it reformat your configuration file in the standard style if there is any wrong intendation or any false format.**
	terraform fmt
	** ***terraform console:****This command provides an interactive command-line console for evaluating and experimenting with expressions. This is useful for testing interpolations before using them in configurations, and for interacting with any values currently saved in state. You can see the attributes of resources in tfstate file and check built in functions before you write in your configuration file. Lets create a file under the terraform-aws directory and name it "cloud" and paste "hello devops engineers". Run the following commands.*
	terraform console **> aws_instance.tf-ec2** **> aws_instance.tf-ec2.private_ip** **> min (1,2,3)** **> lower("HELLO")** **> file("${path.module}/cloud")** **> aws_s3_bucket.tf-s3** **> aws_s3_bucket.tf-s3.bucket** **> exit or (ctrl+c)**
	** **show command:** *You can see tfstate file or plan in the terminal. It is* ***more readable*** *than* ***terraform.tfstate.***
	terraform show
	** **graph command:** It creates a visual graph of Terraform resources. The output of "terraform graph" command is in the DOT format, which can easily be converted to an image by making use of dot provided by GraphViz. Copy the output and paste it to the https://dreampuf.github.io/GraphvizOnline. Then display it. If you want to display this output in your local, you can download graphviz (sudo yum install graphviz) and take a graph.svg with the command terraform graph | dot -Tsvg > graph.svg.
	terraform graph
	** **output command:** ***terraform output*** *command is used for reading an output from a state file. It reads an output variable from a Terraform state file and prints the value. With no additional arguments, output will display all the outputs for the (parent) root module. If NAME is not specified, all outputs are printed.*

	output "tf-example-public-ip" { **value = aws_instance.tf-ec2.public_ip** **}** **output "tf-example-s3-meta" {** **value = aws_s3_bucket.tf-s3.region** **}**
	$ terraform apply
	Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
	Outputs:
	tf-example-public-ip = "18.204.210.119"
	tf-example-s3-meta = "us-east-1"
	$ terraform output
	tf-example-public-ip = "18.204.210.119"
	tf-example-s3-meta = "us-east-1"
	$ terraform output -json
	$ terraform output tf-example-public-ip
	refresh command: The terraform refresh command is used to **update the state file with the real-world infrastructure**. This can be used to detect any drift from the last-known state, and to update the state file. First, check the current state of your resources with terraform state list. Then go to the AWS console and delete your S3 bucket. Display the state list again and refresh the state. Run the following commands.
	Before Deletion of S3 on AWS Console
	$ terraform state list aws_instance.tf-example-ec2 aws_s3_bucket.tf-example-s3
	After Deletion of S3 on AWS Console: *Still shows S3. Because Terraform looks its state file rather than AWS console and state file still included S3!*
	$ terraform state list aws_instance.tf-example-ec2 aws_s3_bucket.tf-example-s3
	$ terraform refresh
	$ terraform state list aws_instance.tf-example-ec2
	*** Now, you can see the differences between files **terraform.tfstate** and **terraform.tfstate.backup**. From tfstate file S3 bucket is deleted but in backup file you can see the S3 bucket.*

	*** Run terraform **apply -auto-approve** and create S3 bucket again.*
	$ terraform apply -auto-approve
	terraform apply -refresh=false : When you add something to config (.tf) file & run this comand, Terraform only check/refresh new adding or changings rather than whole file and this makes the execution duration shorter. This is not recommeded by Terraform.
	Make the new changes in the main.tf file.

	output "tf-example-private-ip" {
	value = aws_instance.tf-ec2.private_ip
	}

	$ terraform apply -refresh=false
	An execution plan has been generated and is shown below. Resource actions are indicated with the following symbols: Terraform will perform the following actions: Plan: 0 to add, 0 to change, 0 to destroy. Changes to Outputs: + tf-example-private-ip = "172.31.22.95" Do you want to perform these actions? Terraform will perform the actions described above. Only 'yes' will be accepted to approve. Enter a value: yes Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
	$ terraform destroy : Remove all resources on infrastructure