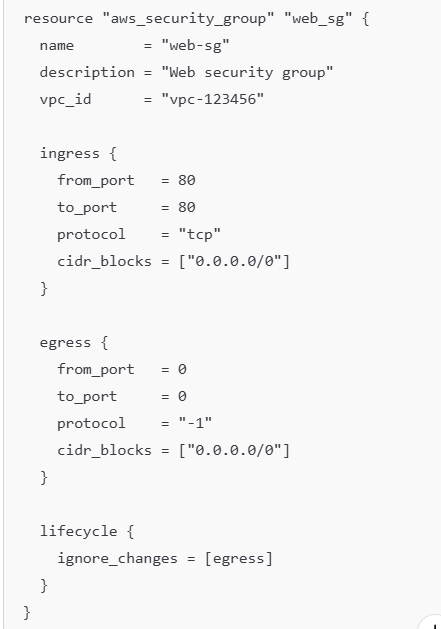
**Terraform Interview Questions:**

***Suppose you created a security group using terraform, after 6 months new members they were not aware like this security group is created by terraform they have updated the outbound rules and they are valid.Now when we run the security module again it will try to change those manual changes.How can we avoid that situation.***  
  
Use lifecycle.ignore\_changes  
Terraform will **stop tracking the egress rules**.



## **All Lifecycle Meta-Arguments in Terraform**

Here are the **4 lifecycle meta-arguments** you can use:

|  |  |
| --- | --- |
| **Lifecycle Option** | **Description** |
| create\_before\_destroy | Create a new resource **before destroying** the old one. Useful to avoid downtime. |
| prevent\_destroy | Prevents a resource from being **accidentally deleted** via Terraform. |
| ignore\_changes | Tells Terraform to **ignore changes** to specific attributes, often used for manual or auto-managed changes. |
| replace\_triggered\_by | Forces a **replacement** of a resource when another resource changes. Useful in advanced dependency scenarios. |

Terraform lifecycle block and when you should use them:

## **1️⃣ create\_before\_destroy**

🔄 **Scenario**: Preventing downtime during resource replacement.

### **✅ When to Use:**

* You're replacing resources like **load balancers**, **EKS node groups**, or **EC2 instances** and want to **create new ones first** before destroying the old.
* Used when Terraform detects a **change that requires replacement** (e.g., immutable field change).

h

CopyEdit

lifecycle {  
 create\_before\_destroy = true  
}

### **🛠 Example:**

You're changing a **subnet's CIDR block**. Without this, Terraform will destroy the old one first → downtime!

## **2️⃣ prevent\_destroy**

🛑 **Scenario**: Critical resources must not be deleted under any condition.

### **✅ When to Use:**

* You want to **protect production** resources like:
  + VPCs
  + Databases
  + IAM roles
* Prevent team errors like running terraform destroy by accident.

lifecycle {  
 prevent\_destroy = true  
}

### **🛠 Example:**

Someone tries to delete your production RDS instance. Terraform will throw an error and stop.

## **3️⃣ ignore\_changes**

🙈 **Scenario**: Certain fields are updated **outside Terraform** and you don’t want Terraform to overwrite them.

### **✅ When to Use:**

* Fields managed by external tools/scripts
* **Manual AWS Console updates** that should not be reverted
* Tags or startup scripts that change often

h

CopyEdit

lifecycle {  
 ignore\_changes = [user\_data, tags["Owner"]]  
}

### **🛠 Example:**

Your team updates EC2 user\_data manually for patching. You don’t want Terraform to reset it on the next apply.

## **4️⃣ replace\_triggered\_by**

🔁 **Scenario**: You want to **replace a resource** when another related resource changes.

### **✅ When to Use:**

* One resource depends on another **without direct dependency**
* You want to **trigger replacement manually** on external change

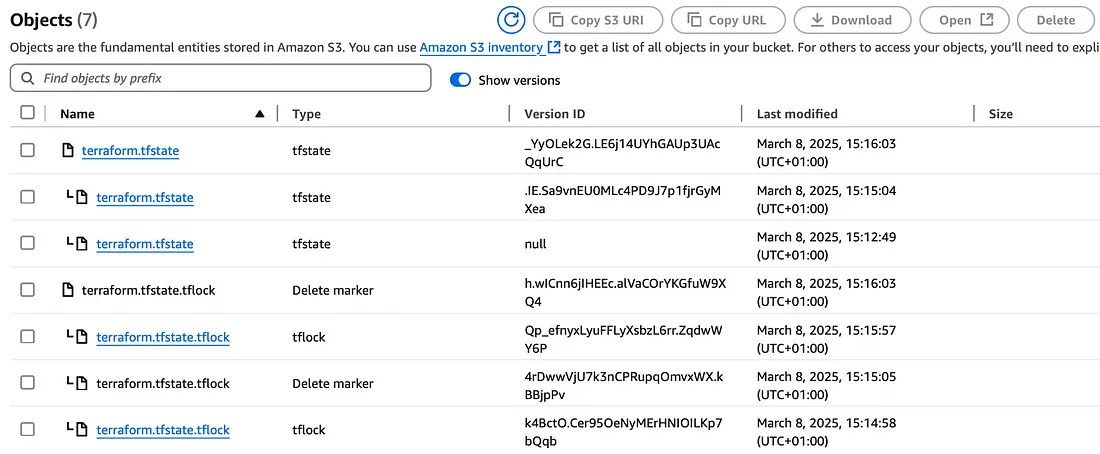
hcl

CopyEdit

lifecycle {  
 replace\_triggered\_by = [aws\_key\_pair.my\_key]  
}

### **🛠 Example:**

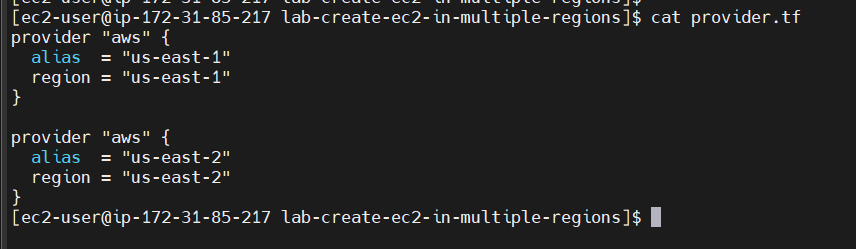
You have an EC2 instance tied to a key pair. If the key changes, the instance must be recreated — this forces that.

*Initially we use to have dynamodb for state locking now it is not required can you tell me which service we can use?*  
S3-native state locking  
With S3-native state locking, we no longer need DynamoDB for state locking. Instead, we use a lock file directly within the S3 bucket.   
To configure the S3 backend with native state locking, simply add the parameter use\_lockfile and set it to true:  
  
When you deploy resources, you’ll see a lock file appear temporarily in your S3 bucket. With versioning enabled, a delete marker will appear for the lock file once your terraform apply completes successfully.  
  
If the state lock file does not clear automatically after a failed or interrupted deployment, preventing further Terraform runs, simply delete the lock file manually from the S3 bucket. This process is simpler compared to manually deleting lock records from DynamoDB.

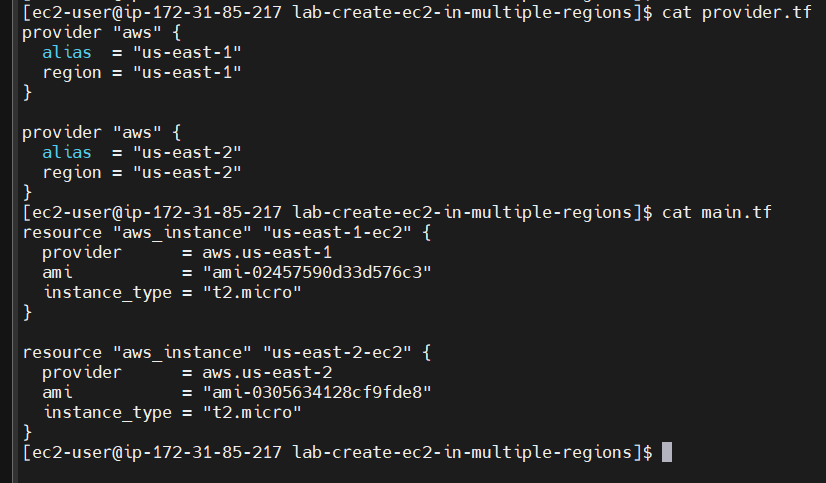
# **Migrating from DynamoDB to S3-Native State Locking**

If you currently use DynamoDB for state locking and have upgraded to Terraform version 1.11.0 or higher,migrating to S3-native state locking is straightforward:  
 \* Add the use\_lockfile parameter to your backend configuration.  
 \* Set the parameter to true.  
If you prefer testing first, you can temporarily use both S3-native and DynamoDB state locking simultaneously to ensure the new method works properly before fully migrating:  


Q) How to create two ec2 instance in 2 different regions with single main.tf file?

To create EC2 instances in two different AWS regions using a single main.tf file, you'll leverage multiple provider configurations and aliases within Terraform. This allows you to specify different regions for different EC2 instances within the same Terraform execution.  
  
provider.tf  


Main.tf



**Terraform advantages?**

* 1. To Provision the infrastructure so that with the help of terraform script we can easily automate our aws

Infrastructure.

* 1. Managing the existing infrastructure with the help of state file.
  2. Replicating infrastructure.

How terraform is different from other IaC tools?  
Terraform is way more advance than other IaC tools.  
Coparing to Ansible terraform has state management using state

**Is there any line of code that which prevent resources from being destroyed even when you apply terraform destroy?**

To prevent a specific resource from being destroyed even when you apply terraform destroy, you need to add an attribute "prevent\_destroy" to that specific resource.

lifecycle {

prevent\_destroy = true

}

Life cycle of terraform   
Terraform life cycle involves.  
Terraform init  
terraform plan

Terraform apply

Terraform destroy

How do you use terraform to create an eks cluster?

What is Drift in Terraform?   
Drift occurs when the actual state of infrastructure (in the cloud) differs from the state known to Terraform (stored in its state file).  
We can detect drift using terraform plan   
  
**How to find there is a drift ?**  
*# 1. Pull latest state*  
terraform refresh  
  
*# 2. Compare with configuration*  
terraform plan  
  
*# 3. Review changes and decide whether to:*  
*# - Accept (import or update config)*  
*# - Reject (terraform apply to revert)*  
  
  
How do you manage sensitive varibales and secrets in terraform ?  
To hide the sensitive variables we can use sensitive = true , in variable block.  
This will mask the variable in the plan and apply output on the terminal.   
Better use a secrets.tfvars file and store all the sensitive varibales in that and pass that file using –var-file when using terraform apply   
Ex: terraform apply –var-file =secrets.tfvars  
  
To manage secrets we can use AWS secrets manager or Hashicorp vault to manage secrets effectively.

**If you are using terraform workspaces the different workspace will have a different state file how can you manage these state files ?**  
When using workspaces (e.g., default, dev, prod, etc.), **Terraform automatically separates the state files** for each workspace.  
  
terraform.tfstate # for default workspace

terraform.tfstate.d/

dev/

terraform.tfstate

prod/

Terraform.tfstate  
  
  
---------------------------------------------------------------------------------------------------------------------------------------------------------------

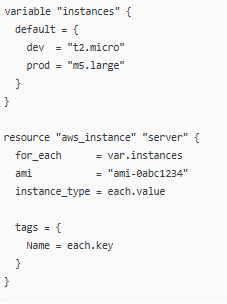
**`terraform state rm**` does — it removes a resource from Terraform’s state file, but leaves the actual infrastructure untouched.  
When Should You Use It?  
  
1️⃣ A resource was created manually and you no longer want Terraform to manage it  
2️⃣ You’re moving a resource to another Terraform project or module  
3️⃣ You’re renaming or refactoring resources  
4️⃣ A resource was deleted outside of Terraform but still shows up in state  
5️⃣ You’re trying to fix dependency or destroy/apply errors  
6️⃣ You want to hand over the resource to another team or tool for manual management

**Write a terraform script to fetch the ami id dynamically for creating ec2 instance.**  

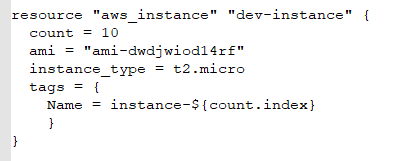

Create an ec2 instance ?  
  

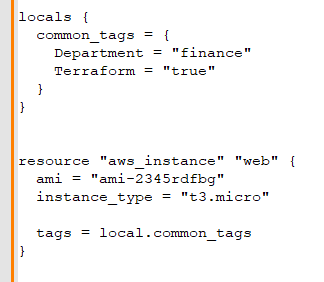

**Create resource based on variable ?**  
We can use count and for\_each arguments for this   
COUNT:

|  |  |
| --- | --- |
|  | dynamically setting count using list variable |

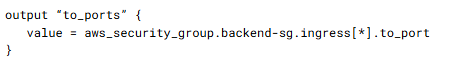
for\_each:  


Create 10 instances with single reource block each should have a different tag ?

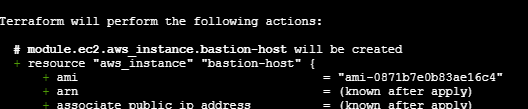
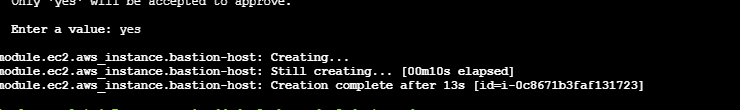


Q) what are local values in terraform ?  
  
Local values in terraform can simplify your configiration by eliminating the repetitive code.  
Example:  
  
  
**Q) suppose your terraform state file gets corrupted or out of sync . What steps would you take to recover from that ?**  
**Make a backup** of the current state file immediately:  
$ terraform state pull > corrupted\_state\_backup.json  
**Verify the corruption** by running:  
$ terraform plan  
Recovery Options:  
Manual State Manipulation  
Edit the state file (for simple issues):  
  
$ terraform state list # Identify problematic resources  
$ terraform state rm <resource.address> # Remove problematic resource  
$ terraform import <resource.address> <resource.id> # Re-import if needed  
  
Prevention:  
**Enable remote state** with locking (S3 + DynamoDB)  
  
**If someone manually changes s3 bucket policy that was originally created with terraform, how would you deal with that kind of drift ?**  
  
Import the Modified Policy (If You Want to Keep Some Changes)  
*# First, get the current policy from AWS*  
aws s3api get-bucket-policy --bucket YOUR\_BUCKET\_NAME > current\_policy.json  
  
*# Modify your Terraform configuration to incorporate desired changes*  
*# Then import the policy to update Terraform's state*  
terraform import aws\_s3\_bucket\_policy.your\_policy\_resource YOUR\_BUCKET\_NAME  
  
Prevention: Use IAM Permissions Boundaries  
Restrict who can modify bucket policies by implementing IAM boundaries:  
  
  
How do you detect drift in terraform ?  
Use terraform plan command regularly to detect drift in terraform   
  
  
**Ideally there are two resources A and B. Everytime resource A has to be created before the resource B should be created ?**  
We can use depends\_on block in the resource B block and provide the address of resource A then resource A will be created before resource B.  
  
**What all terraform functions you used ?**  
There are many built in functions are available in terraform   
I used collection functions like   
length(var.instance\_type)  
merge for merging my local tags and as well as custom tags on resources.  
  
**In a variable I do have a list of key value pair.How do you iterate and pick a specific value among them**   
  
We can use for\_each meta arguments to oick key value pairs while creating resources   
  
Or   
we can also use lookup function  
  
output "value\_with\_default" {  
 value = lookup(var.key\_value\_pairs, "app4", "default\_value")  
}

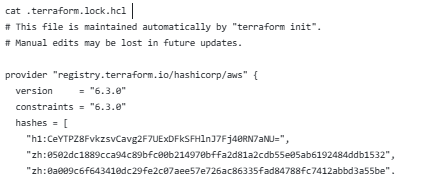
----------------------------------------------------  
**What all steps involved in creating an eks cluster using terraform ?**  
For eks cluster it requirs multiple resource blocks   
For example   
In eks cluster -- main.tf file we need   
 an aws\_iam\_role that which assumes a role on eks service  
 iam role attachment reosurce block with policy\_arn and role  
 eks\_cluster configuration resource block   
Coming to eks\_node\_group  
 In this node group resource block we mention the cluster name, node group name subnet ids  
 scaling config which includes min\_size, desired\_size, max\_size   
 remote access – ec2\_ssh\_key , source\_security\_group\_ids  
 instance\_type , ami\_type

We need to attach policy like AmazonEKSWorkerNodePolicy, AmazonEKS\_CNI\_Policy, AmazonEC2ContainerRegistryReadOnly  
  
**Splat Expressions ?**   
 Allows you to extract specific attributes from a list of blocks. For instance if you want to output all the to\_port values. From the dynamically generated ingress rules. You can use the following output variable.  
  
  
  
**Dynamic Block:**  
**Use dynamic block to automatically generate the repeatable nested blocks for example the ingress blocks in aws\_security\_group resource based on a variable of type list/map**

|  |  |
| --- | --- |
|  | It is also possible to specify a custom iterator name instead of using the default. Using an iterator named port, the dynamic block can be rewritten as follows: |

**What happens when we remove a resource from statefile?**   
  
  
**Below is terraform plan output**   
  
  
**$ terraform apply**   
**Will create a new resource**   
  
Here in some cases it depends on the resource type   
For ex: if we remove an ec2 instance from the statefile. Terraform will try to create the ec2 again again if we remove an eks cluster from the state file it will throw error like.  
  
  
  
**How terraform is different from other IaC tools?**  
- For example, lets say we are creating 2 ec2 instances with ansible as ansible used procedural approach we need to define every step explicitly in the playbook. Once we run the playbook it creates 2 ec2 instances and again if we execute the playbook 2 time it again created 2 ec2 instances as it does not have state management. To manage the behavior we need additional parameters in the playbook to be included.  
- Whereas with terraform we use declarative style of approach if we create two ec2 instances with terraform it stores its state in a state file and has info about it and on subsequent execution terraform will not make any changes to the infrastructure it says current state matches the configuration without any additional paramenters.  
  
**Resources:**  
Lets you create new resources this block startes with the keyword resource and is enclosed in curly braces. It tells terraform whcih infrastructure object to create.  
  
  
**Q)what is the purpose of .terrafom.lock.hcl file**   
This file is generated when we first initilize the terrafom directory where have our provider configuration.in general terraform init will initialize the working directory and installs the provider plugin and creates two file .terraform and .terrafom.lock.hcl

This .terrafom.lock.hcl contains exact providers version that are used to create the terrafom managed infrastructure.it will lock the exact version to maintain consistancy across all the environments.

cat .terraform.lock.hcl  


--------------------------------------------------------------------------------------------------- **Q)How do you upgrade plugins in terraform?**  
TO update the provider plugins.

1) we need to update the desire version in the provider.tf or required\_providers block in the terraform configuration.  
2) Then execute terrafom init -upgrade   
This command will download the latest version of the provider that matches your version constraint  
Updates the .terraform.lock.hcl file with the new version and checksums  
3) then verify the .terrafom.lock.hcl file   
**Steps:**  
updated provider from 6.3.0 to 6.5.0 in provider.tf file under required\_providefr block.   
executed   
***$ terraform init -upgrade***   
output:  
[ec2-user@ip-172-31-47-110 terraform]$ terraform init -upgrade

Initializing the backend...

Upgrading modules...

- ec2 in modules/ec2

- efs in modules/efs

- eks in modules/Eks

- node\_group in modules/eks-node-group

- sg in modules/security-groups

- vpc in modules/vpc

Initializing provider plugins...

- Finding hashicorp/aws versions matching "6.5.0"...

- Installing hashicorp/aws v6.5.0...

- Installed hashicorp/aws v6.5.0 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded

in the .terraform.lock.hcl file. Review those changes and commit them to your

version control system if they represent changes you intended to make.

Terraform has been successfully initialized!

$ cat .terraform.lock.hcl

# This file is maintained automatically by "terraform init".

# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {

version = "6.5.0"

constraints = "6.5.0"

hashes = [

"h1:Tn/mGUS27xOhYi1yGXJfQXQtScNvyuTjd49KX5ZjhBM=",

"zh:0257c2719dc8508bc3ef5ac8df3c84b3ef61211ec46b6e5ed951681bbfe08d22",

"zh:3828d4409e2a68fccc9f9fb583167501cc4d38a5ecbb2408cb5781096739311b",

"zh:3cf7062a4a2530c2137473cc4281fd088cfe0059ad8cdb766e2083ac02c85aa9",

"zh:44c2caadd5d3ad4a69a646251319cce406c9800b2b823c2c59e8b0a3ea73fabd",

"zh:4924d88dbb45c9a01dc69323f731b969c2562631832509525ad44331e3682f43",

"zh:7df20d6ce088b131501f5dae9c3de763f81ac266000c19d4d53be79f568ecd24",

" ]

}  
  
---------------------------------------------------------------------------------------------------

**terraform validate** command checks whether your configuration syntax is correct and internally consistent. When the configuration is correct, it shows a success. Message success The configuration is valid.  
  
 **terraform fmt**. Commands can solve configuration files in the current directory and reformats them into a canonical format. This improves readability and consistency across your files.  
  
 **Terraform show** command displays the current state of your managed infrastructure. This includes A detailed view of resources and their attributes.  
  
 **terraform providers** command list all providers declared in your configuration.  
  
 **terraform output** command retrieves the output variables defined in your configuration. You can display either all outputs or a specific one by providing the output name.   
  
 **terraform refresh** command explicitly syncs. The state file with the actual infrastructure.  
  
 **terraform graph** command generates a visual representation of resource dependencies in your configuration. Our execution plan.  
  
 **terraform state list** command to display all resources recorded in the state file.  
  
 **terraform state show** command displays attributes of single resource from the state file.  
Terraform state show aws\_s3\_bucket.dev-bucket.

**terraform state mv** command is used to rename or move a resource within the state file.  
  
 **terraform state rm** command allows you to remove one or more resources from the state file. This is useful when you no longer want Terraform to manage a resource. After removing the resource from the state file, you should manually remove the configuration from the main dot TF file.   
  
 **terraform state push** command updates the remote state by pushing your local state file.  
  
**To enable logging in terraform**  
$export= TF\_LOG=info [info,warning,error,debug,trace]  
$ export TF\_LOG\_PATH=/tmp/terraform.log  
  
Q) how do you import pre-existing resources under terraform?  
Q) how do you include manually created resource under terraform or any other tools?  
 1st. Define empty resource block.  
 execute terraform import aws\_instance.dev-instance instance-id  
refer state file for sttributes   
update the configuration file   
verfify the configuration by running terraform plan --> It should show No changes.Infrastructure ui is up-to-date.  
  
**Terraform Provisioners:**  
Provisioners enable you to execute commands or scripts either locally or on remote resources.  
  
*1) “remote-exec” provisioner* --> Allows you to run commands on a remote resource, such as a web server instance.   
For example, after creating a web server, you might want to update the package list, install nginx, enable it at startup and then start the service.

|  |  |
| --- | --- |
| resource "aws\_instance" "webserver" {  ami = "ami-0edab43b6fa892279"  instance\_type = "t2.micro"    provisioner "remote-exec" {  inline = [  "sudo apt update",  "sudo apt install nginx -y",  "sudo systemctl enable nginx",  "sudo systemctl start nginx",  ]  }    connection {  type = "ssh"  host = self.public\_ip  user = "ubuntu"  private\_key = file("/root/.ssh/web")  }    key\_name = aws\_key\_pair.web.id  vpc\_security\_group\_ids = [aws\_security\_group.ssh-access.id] }   resource "aws\_key\_pair" "web" {  << code hidden >> } | To facilitate secure authentication, define a connection block inside your resource configuration.   The connection block below enables SSH access by setting the host to the instance's public IP, the user to "ubuntu," and retrieves the private key from a local file:  When you run **terraform apply**, Terraform connects to the instance via SSH as specified in the connection block and executes the inline script. |

*2) “local-exec”*   
The local-exec provisioner runs commands on the local machine where Terraform is executed. It’s useful for tasks like logging or writing data to a file. For example, to save the public IP address of an EC2 instance to **/tmp/ips.txt**, you can configure the local-exec provisioner as follows:

|  |  |
| --- | --- |
| resource "aws\_instance" "webserver" {  ami = "ami-0edab43b6fa892279"  instance\_type = "t2.micro"    provisioner "local-exec" {  command = "echo ${aws\_instance.webserver.public\_ip} >> /tmp/ips.txt"  } } | After running **terraform apply**, the instance's public IP will be appended to the file on your local system.  Provisioners run by default after resource creation (create-time provisioners), but you can also execute them before resource destruction. In the example below, one local-exec provisioner runs post-creation and another runs pre-destruction:  resource "aws\_instance" "webserver" {  ami = "ami-0edab43b6fa892279"  instance\_type = "t2.micro"    provisioner "local-exec" {  command = "echo Instance ${aws\_instance.webserver.public\_ip} Created! > /tmp/instance\_state.txt"  }    provisioner "local-exec" {  when = destroy  command = "echo Instance ${aws\_instance.webserver.public\_ip} Destroyed! > /tmp/instance\_state.txt"  } } |

*on\_failure settings to control behavior in case of provisioner execution errors, such as continue (ignore error) or fail (stop execution and mark resource as tainted).*  
  
  
  
***File Provisioner:***

Transfers files or directories from the local machine to a remote resource.

Often used to copy configuration files, scripts, or application binaries to the provisioned instance before executing other commands.  
  
  
  
**Q) Whats the difference between .tfvars and .auto.tfvars file**

Both are used to input variables in terraform configuration.to use tfvars we need to specify the -var-file while running the terraform apply command.   
While .auto.tfvars will be loaded automatically when we execute terraform plan or terrafom apply commands.