Terraform

"Terraform is an open-source Infrastructure as Code (IaC) tool that lets you define and manage your cloud resources—like servers, databases, and networks—using simple configuration files. Instead of clicking around in the cloud console, you write code, and Terraform automatically creates, updates, or deletes resources to match that code. It works with multiple cloud providers (AWS, Azure, GCP, etc.) and keeps track of your infrastructure state so changes are predictable and repeatable."

human-understandable:

"Think of Terraform like a blueprint for your cloud setup. You describe what you want—like 'I need 2 servers and a database'—and Terraform talks to the cloud and builds it for you. If you later change the blueprint, Terraform updates the real setup to match it."

Key points interviewers expect you to cover:

- **IaC**: Write code to manage infrastructure.
- **Multi-cloud**: Works with AWS, Azure, GCP, and more.
- **Declarative**: You describe the desired state, Terraform figures out how to get there.
- **State management**: Keeps track of what's deployed.
- **Automation**: Repeatable and version-controlled infrastructure.

Terraform init

terraform init initializes a Terraform working directory by installing the required provider plugins, setting up the backend for storing state files, and preparing modules. It ensures Terraform has the necessary binaries and configuration to manage infrastructure according to the .tf files in the directory.

Key points (interview-safe):

- Initializes the project.
- Downloads required providers (plugins that talk to cloud services).
- Prepares modules (reusable code blocks).
- Creates the hidden .terraform folder where all this setup is stored.

Points to highlight:

1. **Provider Installation** – Downloads provider binaries (e.g., AWS, Azure, GCP) specified in the configuration.
2. **Module Initialization** – Fetches any referenced modules from local paths or remote sources (Git, Terraform Registry, etc.).
3. **Backend Configuration** – Sets up where Terraform state will be stored (local or remote, e.g., S3, GCS, etc.).
4. **Workspace Setup** – Prepares the .terraform directory for state and cache files.
5. **Dependency Lock File** – Creates/updates terraform.lock.hcl to ensure consistent provider versions across environments.

💡 **In short:** It sets up the working directory by downloading the required providers (like AWS, Azure, GCP), installing modules, and getting everything ready to run Terraform commands."

Terraform Validate

"Terraform validate checks your Terraform configuration files for syntax errors and internal consistency, but it does not check against real cloud resources."

Key points (interview-friendly):

- Ensures .tf files are **syntactically correct**.
- Checks for **basic logical issues** (e.g., missing variables, unsupported arguments).
- Does **not** contact the cloud provider (no API calls).
- Helps catch mistakes early before running plan or apply.

Analogy:

It's like proofreading your blueprint before sending it to the builder — it makes sure the instructions are written correctly, but doesn't check if the materials exist in the store.

Terraform plan

terraform plan shows you what changes Terraform will make before actually applying them. It compares your configuration files with the current state and gives a preview of resources to be created, updated, or destroyed.

Analogy: It's like reviewing a shopping list before going to the store—terraform plan shows what will be added, changed, or removed, but doesn't actually do the shopping.

Key points (interview-friendly):

- It’s a dry run (no real changes happen).
- Compares desired state (your .tf code) with current state (infra already running).
- Outputs an execution plan (like a preview).
- Helps avoid mistakes before running terraform apply.

Terraform apply

terraform apply actually makes the changes to your cloud. It takes the plan created by Terraform, then creates, updates, or deletes resources so that your real infrastructure matches your code.

Analogy: If terraform plan is like previewing a shopping list, terraform apply is actually going to the store and buying those items.

Key points (interview-safe):

- Executes the changes shown in terraform plan.
- Provisions, updates, or destroys resources in the cloud.
- Asks for confirmation before applying (unless -auto-approve is used).
- Updates the Terraform state file after changes are applied.

Terraform destroy

terraform destroy removes all the resources that Terraform created, tearing down the infrastructure defined in your code.

Analogy: If terraform apply is building a house from a blueprint, terraform destroy is demolishing that house and clearing the land.

Key points (interview-safe):

- Deletes all resources managed by Terraform.
- Useful for cleaning up test or demo environments.
- Updates the state file after destroying resources.
- Asks for confirmation before deleting (unless-auto-approve is used).

Terraform Command Lifecycle (for Interviews)

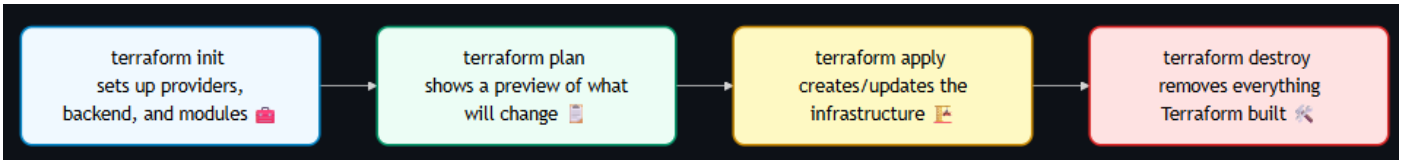
Command	Purpose (Simple)	Purpose (Technical)
terraform init	Get tools ready	Downloads provider plugins, configures backend, fetches modules, creates .terraform directory and terraform.lock.hcl.
terraform plan	Preview changes	Creates an execution plan showing what will be created, updated, or destroyed, based on .tf code vs. current state.
terraform apply	Build/update infra	Executes the plan to provision/update resources in the target cloud environment.
terraform destroy	Tear down infra	Destroys all managed resources defined in the Terraform state.

Analogy for interviews (easy + technical):

- init = Setting up the toolbox 🧰
- plan = Drawing the blueprint 📐
- apply = Building as per blueprint 🏗️
- destroy = Demolishing the setup 🛠️

Mermaid diagram you can use to visually explain the Terraform lifecycle

(init → plan → apply → destroy) in an interview:



Terraform Commands with Examples

1. terraform init – Initialize a project

- Prepares the working directory.
- Downloads providers & modules.
- Sets up the backend.

```
terraform init

# Child options:

terraform init -upgrade # → Upgrade providers/modules to the latest version.

terraform init -backend=false # → Skip backend setup.

terraform init -reconfigure # → Reinitialize backend ignoring old settings
```

2. terraform validate – Validate configuration

- Checks .tf files for syntax & logical errors.
- Doesn't talk to the provider API.

```
terraform validate

# Child options:

terraform validate -json # → Output validation results in JSON format.
```

3. terraform plan – Preview execution plan

- Shows execution plan (preview changes).
- Compares desired vs. current state.
- Shows what Terraform *will do* without making changes.

```
terraform plan

# Child options:

terraform plan -out=tfplan # -out=planfile → Save the plan to a file.

terraform plan -var="instance_type=t2.micro" # -var / -var-file → Pass variables.

terraform plan -refresh=false # → Don't refresh remote state before planning.
```

4. terraform apply – Apply changes

Executes the plan to provision/update infrastructure.

```
terraform apply

# Child options

terraform apply -auto-approve # → Skip manual approval.

terraform apply tfplan(planfile)   # → Apply a previously saved plan(apply saved plan).

terraform apply -var / -var-file # → Pass variables.
```

5. terraform destroy – Destroy infrastructure

Tears down resources created by Terraform. (or) Deletes all managed resources.

```
terraform destroy
# Child options:
terraform destroy -auto-approve # → Skip confirmation.
terraform destroy -target=aws_instance.my_ec2 # -target=resource → Destroy only a specific resource.
```

6. terraform show – Show state or plan details

Displays current state or a saved plan file.

```
terraform show
# Child options:
terraform show tfplan
terraform show -json # → Show state in JSON format.
```

7. terraform state – Manage state files

Inspect and modify Terraform state.

or

Manage and inspect Terraform state files.

```
terraform state list # → List resources in state.
terraform state show aws_instance.my_ec2 # → Show a specific resource.
terraform state rm aws_instance.my_ec2 # → Remove resource from state (doesn't destroy it).
terraform state mv aws_instance.old aws_instance.new # → Move items within state.
```

8. terraform fmt – Format configuration files

Auto-formats .tf files into standard style.

```
terraform fmt
terraform fmt -recursive # → Format all subdirectories.
```

9. terraform output – Show output values

Displays outputs defined in configuration.

```
terraform output
terraform output -json # → Show outputs in JSON format.
terraform output instance_ip
```

10. terraform providers – Show providers used

Lists providers used in configuration.

```
terraform providers

terraform providers mirror # → Copy providers to a local directory.

terraform providers schema # → Show provider schema.
```

11. terraform workspace – Manage workspaces

Manages multiple workspaces.

Useful for multiple environments (dev, staging, prod).

Child commands:

- list → Show all workspaces.
- new → Create a new workspace.
- select → Switch workspace.
- delete → Delete a workspace.
- show → Show current workspace.

```
terraform workspace list

terraform workspace new dev

terraform workspace select dev

terraform workspace show
```

terraform workspace delete dev

12. terraform version – Show version info

Shows Terraform version and installed providers.

```
terraform version
```

13. terraform graph – Show dependency graph

Generates a visual graph of resource dependencies.

```
terraform graph | dot -Tpng > graph.png
```

14. terraform login/logout – Terraform Cloud auth

```
terraform login # → Authenticate with Terraform Cloud.

terraform logout # → Remove saved credentials.
```

15. terraform import – Import existing resource

Brings unmanaged resources into Terraform state.

or

Bring existing resources under Terraform management.

```
terraform import aws_instance.my_ec2 i-0abcd1234efgh5678
```

16. terraform taint/untaint – Force recreation

```
terraform taint aws_instance.my_ec2 # → Mark a resource for recreation on next apply.

terraform untaint aws_instance.my_ec2 # → Remove taint mark.
```

Terraform Lifecycle (core commands):

init → validate → plan → apply → destroy

Helper commands:

fmt, output, show, graph, version

State & environment commands:

state, workspace, providers, import, taint
