Terraform Cheat Sheet

1. Provider Block

```
provider "aws" {
  region = "ap-south-1"
}
```

Purpose: Tells Terraform which cloud/service provider to use.

Explanation: Here, you're using AWS, and provisioning will happen in the Mumbai region (ap-south-1).

2. Resource Block

```
resource "aws_instance" "example" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
}
```

Purpose: Creates an actual infrastructure component.

Explanation: This will create an EC2 instance using the specified AMI and instance type.

3. Variable Block

```
variable "instance_type" {
  description = "EC2 type"
  type        = string
  default     = "t2.micro"
}
```

Purpose: Defines input parameters.

Explanation: Use this to make your code flexible. You can pass values at runtime or use terraform.tfvars.

4. Output Block

```
output "instance_id" {
  value = aws_instance.example.id
}
```

Purpose: Shows useful values after provisioning.

Explanation: Outputs the EC2 instance ID to the console after terraform apply.

5. Locals Block

```
locals {
  name_prefix = "demo"
}
```

Purpose: Defines temporary values (like variables) for reuse.

Explanation: Local variables are useful for code simplification and reuse.

6. Data Block

```
data "aws_ami" "latest" {
  most_recent = true
  owners      = ["amazon"]
}
```

Purpose: Reads existing resources/data without creating them.

Explanation: Useful for fetching latest AMIs, VPCs, subnets, etc., instead of hardcoding them.

7. Conditionals

```
instance_type = var.env == "prod" ? "t3.medium" : "t2.micro"
```

Purpose: Make decisions dynamically in code.

Explanation: Sets a value based on condition (if/else style).

8. Loops a. count

```
resource "aws_instance" "web" {
  count         = 3
  ami           = "ami-123"
  instance_type = "t2.micro"
}
```

Purpose: Create multiple resources of the same type.

Explanation: This creates 3 EC2 instances.

b. for_each

```
resource "aws_s3_bucket" "buckets" {
  for_each = toset(["logs", "images", "backups"])
  bucket   = "my-${each.key}-bucket"
}
```

Purpose: Loop over maps or sets to create multiple resources with different values.

Explanation: Creates a bucket for each key.

9. String Interpolation

```
tags = {
  Name = "${var.env}-server"
}
```

Purpose: Embed variables inside strings.

Explanation: Combines static and dynamic values.

10. String Functions

```
lower("HELLO")      # Converts to lowercase
join("-", ["a", "b"]) # Outputs "a-b"
length(var.list)     # Gets length of a list
```

Purpose: Manipulate text and lists.

Explanation: Handy for formatting tags, resource names, conditions.

11. File Functions

```
user_data = file("init.sh")
```

Purpose: Reads content from a local file.

Explanation: Common for passing shell scripts to EC2 user_data.

12. Terraform CLI Commands

```
terraform init     # Set up the working directory and install providers
terraform plan     # Preview what changes Terraform will make
terraform apply    # Apply the actual changes
terraform destroy  # Tear down the infrastructure
```

Purpose: CLI commands to run your infrastructure code.

Explanation: Standard Terraform workflowinit, plan, apply, destroy.

Summary

Syntax Type	Use Case
provider	Set the cloud provider
resource	Create infrastructure
variable	Accept inputs dynamically
output	Show results post-provisioning
locals	Temporary reusable values
data	Fetch existing infrastructure
count / for_each	Create multiple resources
conditional	Set values based on logic
file()	Read scripts/files
CLI commands	Interact with Terraform engine