Terraform Modular Structure

Modular structure of Terraform

- Promotes code reusability and cleaner organization.
- Allows flexibility by passing variables to customize each environment.
- Outputs key values, helping validate resource creation and enabling further integration.
- Each module is independent and performs a specific function, which aligns well with infrastructure-as-code (iac) principles.
- By using modules, our configuration becomes scalable and maintainable, making it easy to manage cloud infrastructure effectively.

Key Points :

- ✓ **Modular Approach**: The infrastructure is broken down into separate, reusable modules.
- ✓ **Independent Manageability**: Each module can be updated independently, reducing the risk of affecting the entire setup.
- ✓ **Clarity and Ease of Updates**: The structure simplifies updates, making it easy to modify specific parts.
- ✓ **Scalability**: This modular setup can be scaled up for larger infrastructure needs.
- ✓ **Flexibility for Extension**: New modules can be added to expand functionality as requirements grow.

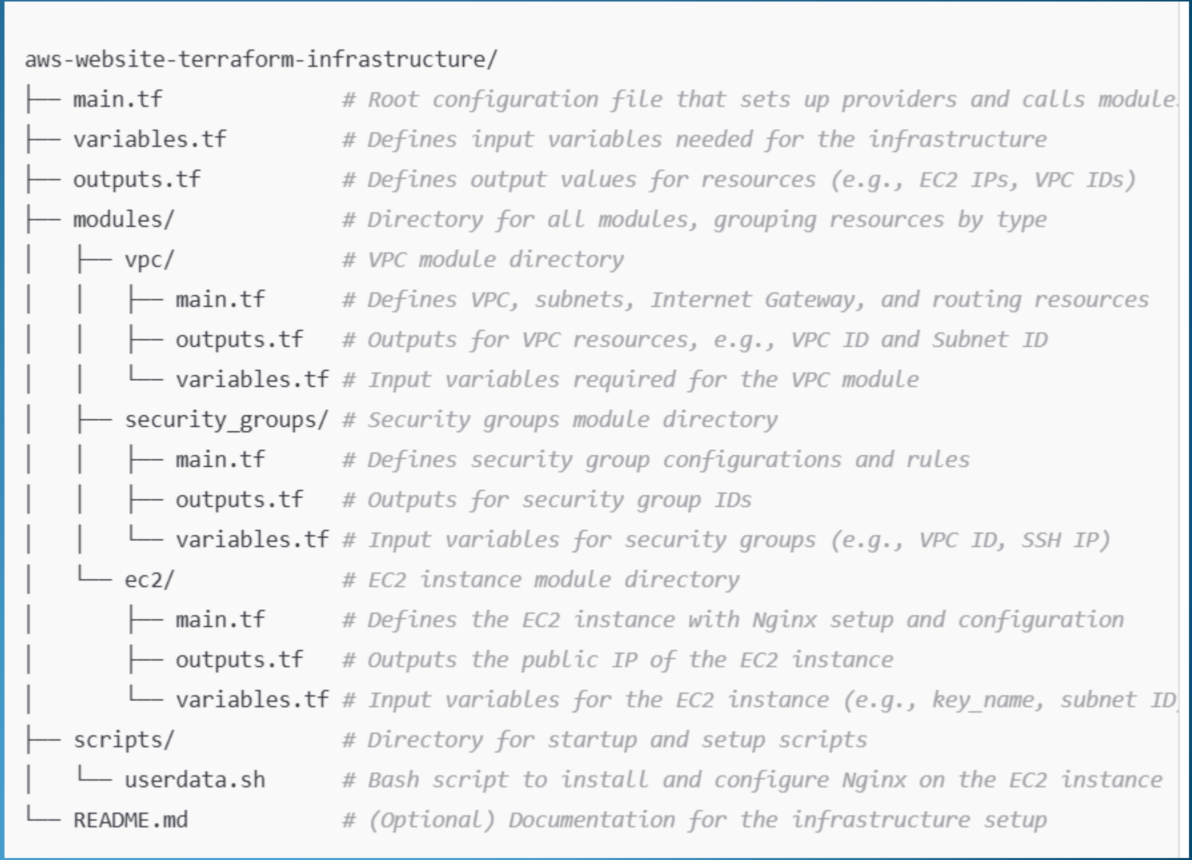
❑ This approach ensures efficient management, adaptability, scalability for the infrastructure

Directory modular Structure

- The modular structure of this Terraform setup organizes your code into separate, reusable modules, making it easier to manage and understand.

- Each module is organized into specific subdirectories based on the resources they manage.
- The root directory (terraform) serves as the main configuration point. t:

A breakdown of the directory structure with an explanation of each part:



```
aws-website-terraform-infrastructure/
├── main.tf            # Root configuration file that sets up providers and calls module
├── variables.tf       # Defines input variables needed for the infrastructure
├── outputs.tf         # Defines output values for resources (e.g., EC2 IPs, VPC IDs)
├── modules/           # Directory for all modules, grouping resources by type
│   ├── vpc/           # VPC module directory
│   │   ├── main.tf    # Defines VPC, subnets, Internet Gateway, and routing resources
│   │   ├── outputs.tf # Outputs for VPC resources, e.g., VPC ID and Subnet ID
│   │   └── variables.tf # Input variables required for the VPC module
│   ├── security_groups/ # Security groups module directory
│   │   ├── main.tf    # Defines security group configurations and rules
│   │   ├── outputs.tf # Outputs for security group IDs
│   │   └── variables.tf # Input variables for security groups (e.g., VPC ID, SSH IP)
│   └── ec2/           # EC2 instance module directory
│       ├── main.tf    # Defines the EC2 instance with Nginx setup and configuration
│       ├── outputs.tf # Outputs the public IP of the EC2 instance
│       └── variables.tf # Input variables for the EC2 instance (e.g., key_name, subnet ID
├── scripts/           # Directory for startup and setup scripts
│   └── userdata.sh    # Bash script to install and configure Nginx on the EC2 instance
└── README.md          # (Optional) Documentation for the infrastructure setup
```

Explanation of Each Component

1. Root Directory:

- Contains main.tf, variables.tf & outputs.tf files.
- This is where the main configuration is specified, including provider setup & module blocks that call each module defined in modules.
- The variables & outputs at the root level control the overall infrastructure behavior.

2. Modules Directory:

- Houses individual modules (vpc, security_groups, and ec2).
- Each module encapsulates a group of related resources for specific functions within the infrastructure.

  - **VPC Module** (modules/vpc):

    - Defines resources related to networking, such as the VPC itself, subnets, an internet gateway, route tables.
    - By modularizing the VPC setup, we can easily **reuse** this configuration across different environments or projects.

  - **Security Groups Module** (modules/security_groups):

    - Manages security rules, specifically defining SSH and HTTP access to the EC2 instance.
    - Passing the ssh_ip variable here allows restriction of SSH access to specific IP addresses for enhanced security.

- **EC2 Module (modules/ec2):**

  - Configures the EC2 instance that hosts the web application.

  -This module sets up the instance type, attaches the necessary security group, and runs a startup script (userdata.sh) to install and configure Nginx.

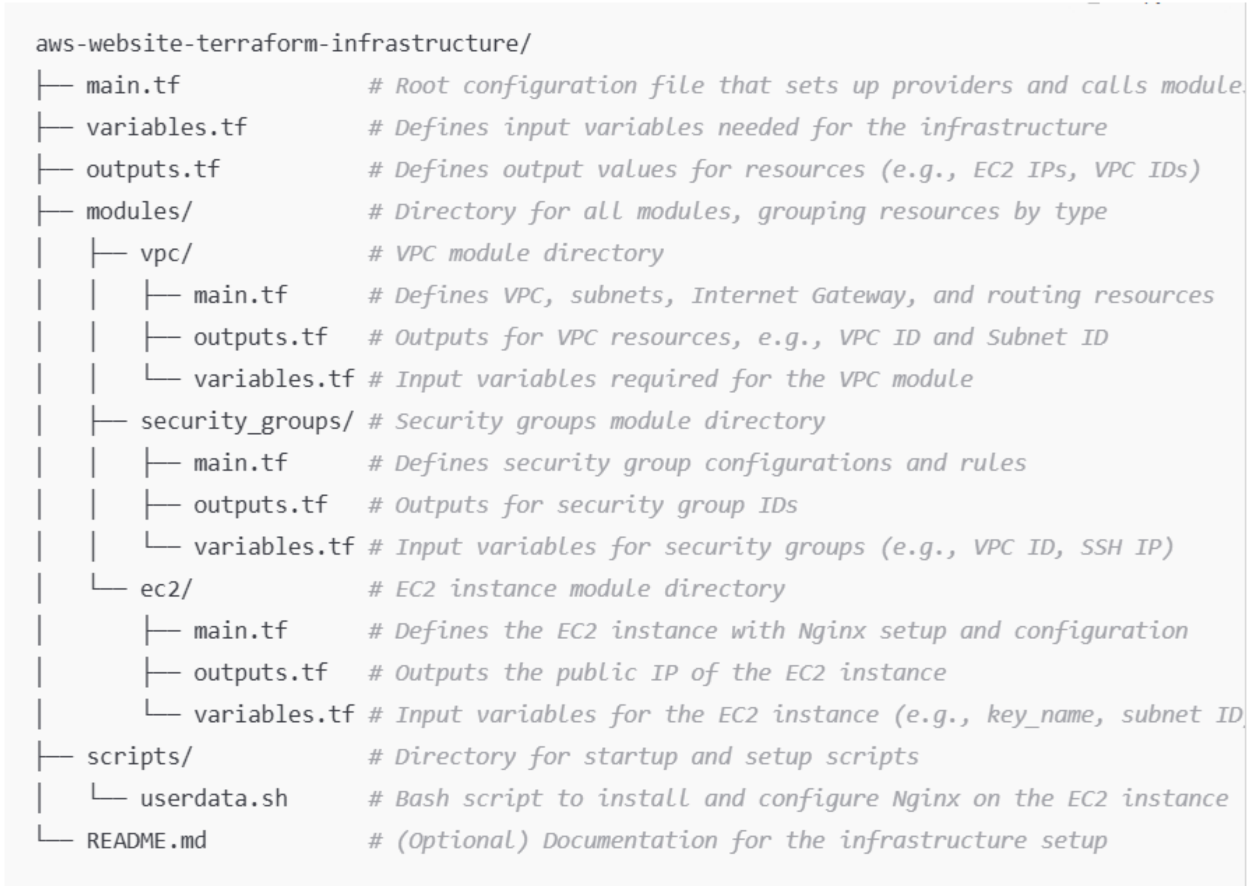
3. Scripts Directory (scripts/):

- Contains shell scripts that run during the EC2 instance's initialization.
- Here, userdata.sh is used to set up Nginx and deploy the website files, making the server ready to serve web requests immediately after launch.

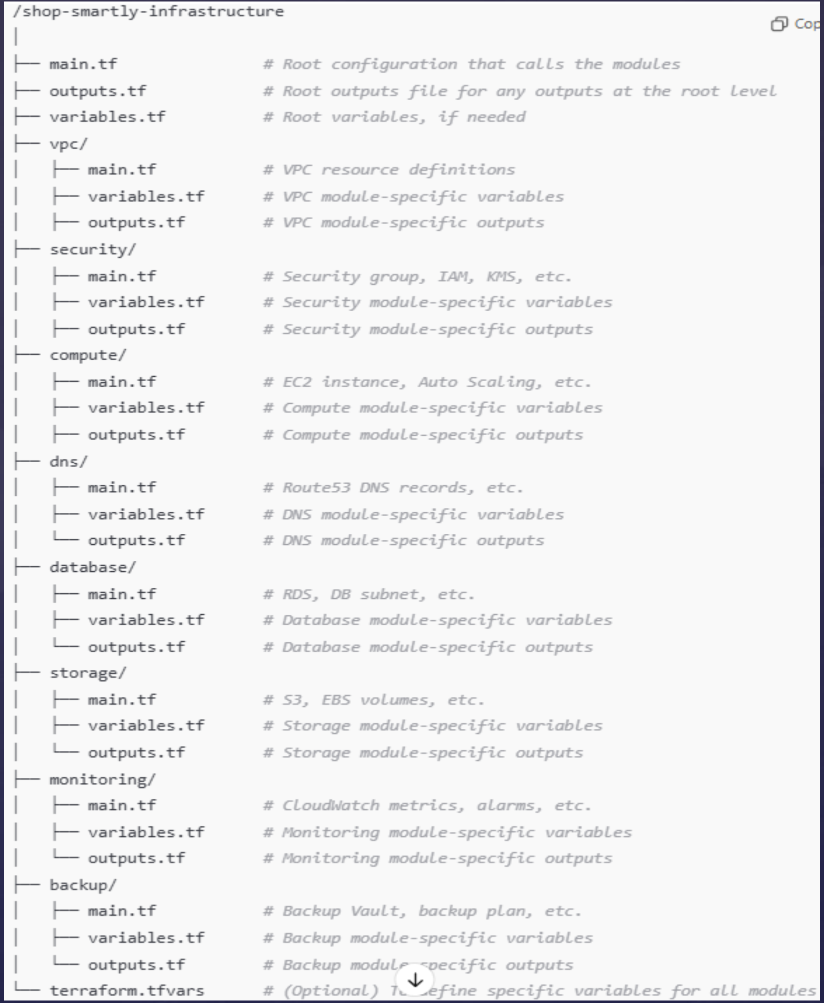
- This modular approach makes each part of the infrastructure reusable and independently manageable, providing clarity and ease of updating specific parts without affecting the entire setup. This structure also promotes scalability and can be extended for larger infrastructures.

❑ Each module is organized into separate subdirectories to improve:

- **readability,**
- **maintainability,**
- **reusability**



```
aws-website-terraform-infrastructure/
├── main.tf                # Root configuration file that sets up providers and calls module
├── variables.tf           # Defines input variables needed for the infrastructure
├── outputs.tf             # Defines output values for resources (e.g., EC2 IPs, VPC IDs)
├── modules/               # Directory for all modules, grouping resources by type
│   ├── vpc/               # VPC module directory
│   │   ├── main.tf        # Defines VPC, subnets, Internet Gateway, and routing resources
│   │   ├── outputs.tf     # Outputs for VPC resources, e.g., VPC ID and Subnet ID
│   │   └── variables.tf # Input variables required for the VPC module
│   ├── security_groups/ # Security groups module directory
│   │   ├── main.tf        # Defines security group configurations and rules
│   │   ├── outputs.tf     # Outputs for security group IDs
│   │   └── variables.tf # Input variables for security groups (e.g., VPC ID, SSH IP)
│   └── ec2/               # EC2 instance module directory
│       ├── main.tf        # Defines the EC2 instance with Nginx setup and configuration
│       ├── outputs.tf     # Outputs the public IP of the EC2 instance
│       └── variables.tf # Input variables for the EC2 instance (e.g., key_name, subnet ID
├── scripts/               # Directory for startup and setup scripts
│   └── userdata.sh        # Bash script to install and configure Nginx on the EC2 instance
└── README.md              # (Optional) Documentation for the infrastructure setup
```

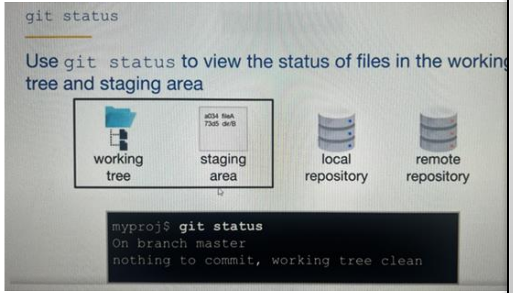


```
/shop-smartly-infrastructure
│
├── main.tf                  # Root configuration that calls the modules
├── outputs.tf               # Root outputs file for any outputs at the root level
├── variables.tf             # Root variables, if needed
├── vpc/
│   ├── main.tf              # VPC resource definitions
│   ├── variables.tf         # VPC module-specific variables
│   ├── outputs.tf           # VPC module-specific outputs
├── security/
│   ├── main.tf              # Security group, IAM, KMS, etc.
│   ├── variables.tf         # Security module-specific variables
│   ├── outputs.tf           # Security module-specific outputs
├── compute/
│   ├── main.tf              # EC2 instance, Auto Scaling, etc.
│   ├── variables.tf         # Compute module-specific variables
│   ├── outputs.tf           # Compute module-specific outputs
├── dns/
│   ├── main.tf              # Route53 DNS records, etc.
│   ├── variables.tf         # DNS module-specific variables
│   └── outputs.tf           # DNS module-specific outputs
├── database/
│   ├── main.tf              # RDS, DB subnet, etc.
│   ├── variables.tf         # Database module-specific variables
│   └── outputs.tf           # Database module-specific outputs
├── storage/
│   ├── main.tf              # S3, EBS volumes, etc.
│   ├── variables.tf         # Storage module-specific variables
│   └── outputs.tf           # Storage module-specific outputs
├── monitoring/
│   ├── main.tf              # CloudWatch metrics, alarms, etc.
│   ├── variables.tf         # Monitoring module-specific variables
│   └── outputs.tf           # Monitoring module-specific outputs
├── backup/
│   ├── main.tf              # Backup Vault, backup plan, etc.
│   ├── variables.tf         # Backup module-specific variables
│   └── outputs.tf           # Backup module-specific outputs
└── terraform.tfvars         # (Optional) To define specific variables for all modules
```

Key takeaway:

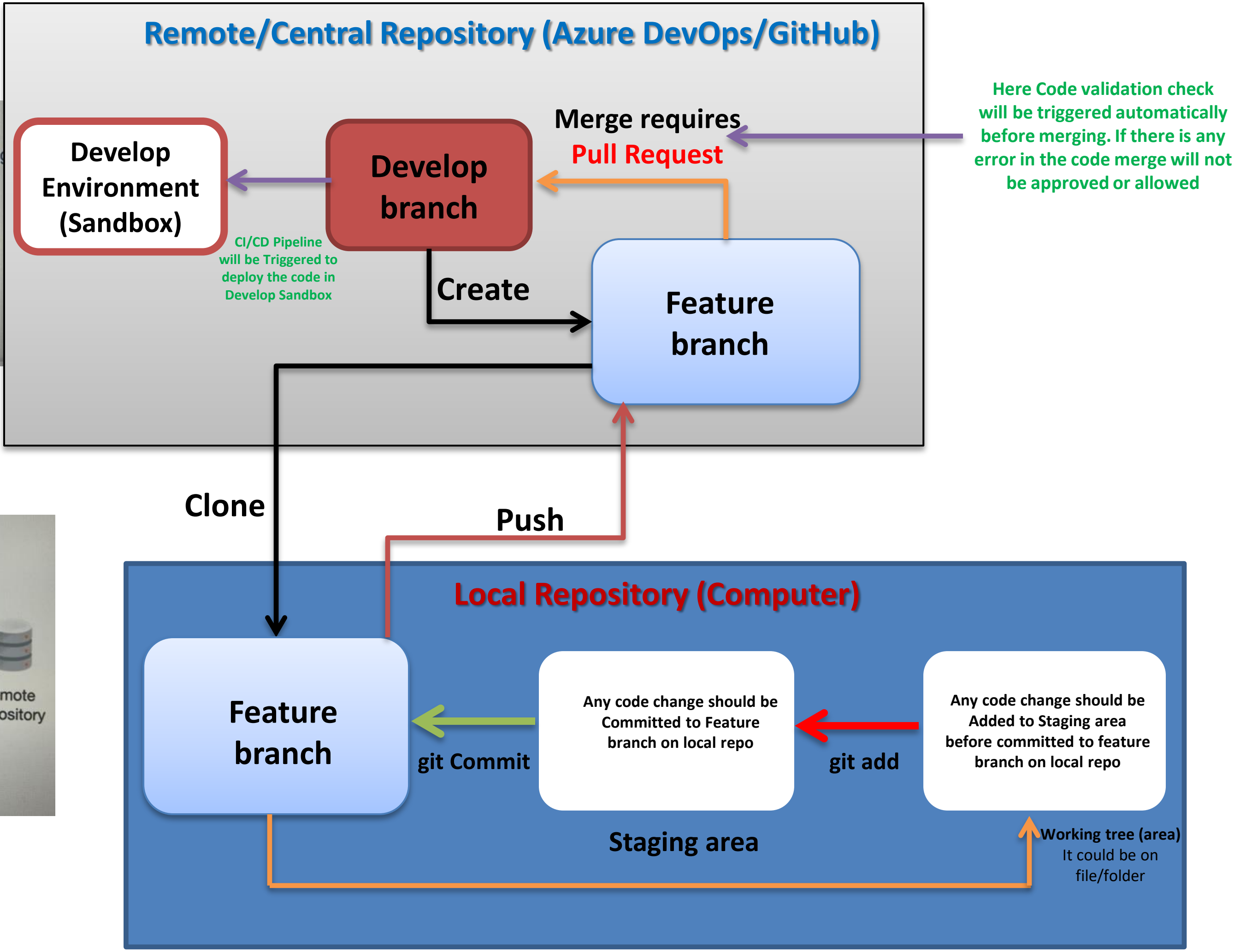
- ✓ **Modular Approach**: The infrastructure is broken down into separate, reusable modules.
- ✓ **Independent Manageability**: Each module can be updated independently, reducing the risk of affecting the entire setup.
- ✓ **Clarity and Ease of Updates**: The structure simplifies updates, making it easy to modify specific parts.
- ✓ **Scalability**: This modular setup can be scaled up for larger infrastructure needs.
- ✓ **Flexibility for Extension**: New modules can be added to expand functionality as requirements grow.

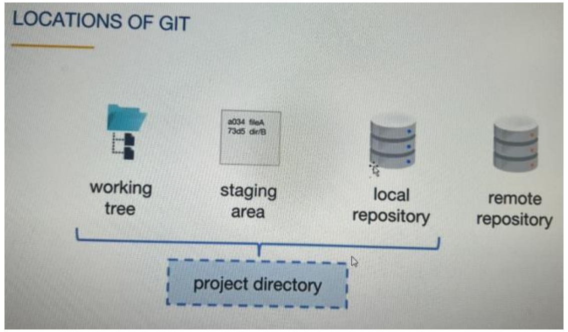
Git



Use `git status` to view the status of files in the working tree and staging area

```
myproj$ git status
On branch master
nothing to commit, working tree clean
```


Remote/Central Repository (Azure DevOps/GitHub)
Develop Environment (Sandbox)
Develop branch
Merge requires Pull Request
Here Code validation check will be triggered automatically before merging. If there is any error in the code merge will not be approved or allowed
CI/CD Pipeline will be Triggered to deploy the code in Develop Sandbox
Create
Feature branch
Clone
Push
Local Repository (Computer)
Feature branch
git Commit
Any code change should be Committed to Feature branch on local repo
git add
Any code change should be Added to Staging area before committed to feature branch on local repo
Staging area
Working tree (area) It could be on file/folder



BASIC GIT COMMANDS

command	description
`git clone` *url [dir]*	copy a Git repository so you can add to it
`git add` ***file***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` *[command]*	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	