Terraform

Key Topics for a DevOps Role Specialized in Terraform:

1. **Infrastructure as Code (IaC):**
   - Concept of IaC and its importance in automating infrastructure.
   - Terraform vs other IaC tools (CloudFormation, Ansible, etc.).
2. **Terraform Basics & Core Concepts:**
   - Providers (AWS, Azure, GCP).
   - Resources, Modules, and Outputs.
   - Data sources and their usage.
   - State management (remote state, terraform.tfstate file).
   - Workspaces and Environments.
3. **Terraform Modules & Best Practices:**
   - Creating reusable modules.
   - Structuring Terraform codebase for scalability.
   - Input variables, locals, and outputs.
4. **State Management:**
   - Locking state files for concurrent access.
   - Remote backend options (S3, Azure Blob, Google Cloud Storage).
   - State file protection and security.
5. **Security & Compliance:**
   - Managing secrets (S3 bucket policies, IAM roles, and encryption with KMS).
   - Handling credentials with AWS Secrets Manager or Parameter Store.
   - Policies for least privilege access in Terraform.
6. **Networking:**
   - Creating VPC, Subnets (public/private), and setting up routing tables.
   - Security groups, NACLs, and VPN connections.
   - Managing load balancers (ALB, NLB).
7. **Scaling & Performance:**
   - Auto Scaling Groups (ASG) for scalability.
   - Elastic Load Balancing (ELB/ALB).

   - Using Terraform to manage Auto Scaling and disaster recovery strategies.

8. **Continuous Integration/Continuous Deployment (CI/CD):**
   - Integrating Terraform with Jenkins, CircleCI, or GitLab pipelines.
   - Running Terraform in a CI/CD pipeline (terraform plan, apply with automated checks).
   - Automated testing with terraform validate and terraform fmt as pre-deployment steps.
9. **Advanced Terraform Concepts:**
   - Dependency management with depends_on.
   - Using count, for_each, and dynamic blocks for scalable infrastructure.
   - Handling multi-cloud or hybrid cloud environments with Terraform.

Vital Terraform Commands (Categorized for Different Scenarios):

1. **Basic Commands:**
   - terraform init: Initialize a working directory containing Terraform configuration files.
   - terraform plan: Create an execution plan.
   - terraform apply: Execute the actions defined in the plan.
   - terraform destroy: Destroy Terraform-managed infrastructure.
2. **State Management:**
   - terraform state list: List resources in the current state.
   - terraform state show <resource>: Show detailed information about a resource in the state file.
   - terraform state mv <resource> <destination>: Move resources within the state.
   - terraform state rm <resource>: Remove a resource from the Terraform state.
   - terraform refresh: Update local state with real-world resources.
3. **Validation & Linting:**
   - terraform validate: Validate the configuration for syntax errors.
   - terraform fmt: Automatically format Terraform code according to style guidelines.
   - terraform graph: Generate a visual representation of the dependency graph.
4. **Module Management:**

- terraform get: Download and update modules.
- terraform output: Show output values from your configuration.
- terraform import: Import existing infrastructure into your Terraform state.

5. **Debugging & Troubleshooting:**
   - terraform taint <resource>: Mark a resource for recreation during the next apply.
   - terraform untaint <resource>: Remove the tainted mark from a resource.
   - TF_LOG=DEBUG terraform apply: Enable detailed logging for debugging.
6. **Workspaces:**

A **Terraform workspace** is an environment within a single Terraform configuration that allows you to manage multiple instances of infrastructure with different states. It helps in managing separate environments like **development**, **staging**, and **production** without duplicating your Terraform code.

- Terraform workspace new <workspace-name>: Create a new workspace.
- terraform workspace select <workspace-name>: Switch to a different workspace.
- terraform workspace delete <workspace-name>: Delete a workspace.

CI/CD Cycle with Terraform:

1. **Development:**
   - Write Terraform configuration files in main.tf, variables.tf, and outputs.tf.
   - Use terraform validate and terraform fmt for pre-deployment checks.
2. **Testing:**
   - Test your infrastructure by running terraform plan in the CI pipeline.
   - Use testing frameworks like terratest for automated infrastructure tests.
3. **Production Deployment:**
   - Use terraform apply to apply changes.
   - Monitor the state using terraform show and ensure that the infrastructure is behaving as expected.
4. **Post-Deployment:**
   - Use terraform destroy for tearing down infrastructure.

Terraform

   - Manage secrets securely by rotating them automatically using AWS Secrets Manager with Lambda.