1. \*\*Complex Infrastructure Provisioning\*\*:

   - \*\*Answer\*\*: To manage complex infrastructure with multiple VPCs, subnets, and security groups, use Terraform modules to encapsulate related resources. Break down the infrastructure into modular components (e.g., networking, application servers, databases) and use module inputs to handle different configurations. Implement output variables to pass information between modules. Leverage Terraform's workspace feature to handle different environments and use `count` or `for\_each` to dynamically manage resources.

1. \*\*State Management\*\*:

   - \*\*Answer\*\*: To recover from a corrupted state file, first, restore the state from the backup if available. If a backup isn’t available, you can manually reconstruct the state using `terraform import` to bring resources into the state file or use the `terraform state` commands to manually edit and fix the state. Ensure you perform a thorough review and testing to confirm that the state accurately reflects the actual infrastructure before proceeding with further changes.

1. \*\*Dependency Management\*\*:

   - \*\*Answer\*\*: Use Terraform's implicit dependency management by referencing output values from one resource in another, ensuring Terraform understands the creation order. For more dynamic dependencies, you can use `depends\_on` to explicitly define dependencies. In complex scenarios, consider breaking down your infrastructure into smaller modules and use data sources to gather information dynamically.

1. \*\*Resource Drift\*\*:

   - \*\*Answer\*\*: To address resource drift, first, run `terraform plan` to identify differences between the actual state and the desired configuration. Use `terraform apply` to reconcile the drift by applying the changes. Implement infrastructure-as-code best practices and restrict manual changes to prevent future drift. Regularly run `terraform plan` to detect and address drift early.

1. \*\*Module Design\*\*:

   - \*\*Answer\*\*: Structure your module to be flexible and reusable by parameterizing region, network configurations, and instance types through module inputs. Use conditional logic and dynamic blocks to handle variations in configurations. Implement best practices for module versioning and maintain a clear interface through input and output variables. Document the module to provide clear guidance on usage and configuration.

1. \*\*Terraform Enterprise/Cloud\*\*:

   - \*\*Answer\*\*: Implement workspace access controls by configuring team permissions and roles within Terraform Cloud or Enterprise. Use policy sets and Sentinel policies to enforce governance and compliance rules. Integrate with version control systems to ensure code reviews and approvals before applying changes. Regularly audit and review policies and permissions to maintain security and compliance.

1. \*\*Performance Optimization\*\*:

   - \*\*Answer\*\*: To optimize Terraform performance, reduce the number of resources in a single plan or apply operation by breaking them into smaller, more manageable configurations. Use `terraform graph` to visualize dependencies and identify any unnecessary dependencies. Leverage parallelism settings (`-parallelism` flag) to speed up resource creation and updates. Review and optimize resource configurations to ensure they are efficient.

1. \*\*Version Control Integration\*\*:

   - \*\*Answer\*\*: Integrate Terraform with CI/CD pipelines by using tools like GitHub Actions, GitLab CI, or Jenkins. Configure the pipeline to run `terraform fmt` and `terraform validate` to ensure code quality before applying changes. Use `terraform plan` to review changes before applying them with `terraform apply`. Implement environment-specific workspaces or configurations in the pipeline to handle different deployment targets.

1. \*\*State Locking and Concurrency\*\*:

   - \*\*Answer\*\*: To address state locking issues, ensure that all team members use a consistent workflow and do not run concurrent Terraform operations on the same state. Use remote state backends with locking mechanisms (e.g., S3 with DynamoDB for locking) to prevent concurrent access. Educate the team on best practices for running Terraform operations and use CI/CD pipelines to manage deployments.

1. \*\*Dynamic Configurations\*\*:

    - \*\*Answer\*\*: Use Terraform’s variable files or environment variables to handle dynamic configurations based on the environment. Implement conditional logic in your Terraform configuration to adjust settings according to the environment. Use `terraform workspace` or separate state files for different environments to manage configurations cleanly and avoid mixing resources.

These answers provide a comprehensive approach to handling complex Terraform scenarios and maintaining robust infrastructure-as-code practices.

**Important questions:**

1. What are all the types of applications you have deployed?

2. How have you injected the secrets in ConfigMaps?

3. How do you find which pod is taking more system resources across nodes using kubectl?

4. How do you know which worker node is consuming more resources across the clusters using kubectl?

5. What are the steps for configuring Prometheus and Grafana for monitoring Kubernetes clusters?

6. If 20 pods are running, how do you visualize the metrics of these pods in Grafana?

7. What is Apache Kafka?

8. How do you set up a Docker Hub private registry and integrate it with a CI/CD pipeline? What is the procedure?

9. What is the difference between a hard link and a soft link?

10. What is the use of the `break` command in shell scripting? In what scenarios have you used it?

11. How do you count the number of "devops" words in 15 HTML files?

12. What is the `terraform taint` command?

13. What are the possible ways to secure a state file in Terraform?

14. If you provision 100 servers and someone deletes 50 VMs manually, what happens if you apply the `terraform apply` command?

15. What is the syntax for `for\_each` in Terraform?

16. What are the advantages and disadvantages of multi-stage builds in Docker?

17. How do you deploy containers on different hosts, not the same host, within a Docker cluster?

18. If you have a Docker Compose setup, how do you deploy the web container on one host and the DB container on another host?

19. What is the difference between bridge networking and host networking in Docker?

20. How do you resolve merge conflicts?

21. What command do you use to change the existing commit message?

22. What is session affinity?

23. What is pod affinity and its use case?

24. What is the difference between pod affinity and pod anti-affinity?

25. What are readiness and liveness probes?

26. Write a simple Groovy pipeline for a Java Spring Boot application that waits for user input for approval to move to the next stage, with stages for checkout, build, push, and deploy?

27. How do you export test reports in Jenkins?

28. If 5 pods are running, how do you scale the number of pods to 10 using the command line in Kubernetes?