RealTime Handson

careerbytecode

10

Terraform

REALTIME REAL TIME SCENARIO BASED QUESTION AND ANSWER

SONALI

https://www.linkedin.com/in/techopsbysonali/

careerbytecode.substack.com

+32 - 471408908

Terraform scenario-based question and Answer:

1. A Terraform apply fails due to a state file lock issue. How would you resolve it?

Steps to resolve:

- **Identify the lock:** Run `terraform plan` or check the Terraform backend to confirm the lock.
- **Check for active sessions:** If using Terraform Cloud/Enterprise or S3 with DynamoDB, verify if another process is holding the lock.
- **Manually unlock (if needed):** Use `terraform force-unlock <LOCK_ID>` cautiously. Ensure no active Terraform processes are running before forcing an unlock.
- **Use a backend with locking support:** If using S3 as a backend, enable DynamoDB for state locking.
- **Prevent future locks:** Use `terraform apply -lock-timeout=5m` to wait before failing due to locking issues.

---

2. Your Terraform deployment created resources in AWS, but some are not being deleted when destroying the infrastructure. What went wrong?

Steps to troubleshoot:

- **Check resource dependencies:** Run `terraform graph | dot -Tpng > graph.png` to visualize dependencies.
- **Examine lifecycle rules:** If `prevent_destroy` is set in `lifecycle` block, Terraform will not delete those resources.
- **Check manual modifications:** If resources were manually changed outside Terraform, they might be ignored. Run `terraform refresh` or import them with `terraform import`.
- **Verify remote state:** If the state file is outdated, run `terraform refresh` to sync it.
- **Review provider-specific settings:** Some AWS resources require explicit deletion settings (e.g., S3 versioned buckets, RDS final snapshot).

---

3. You need to deploy infrastructure across multiple AWS accounts using Terraform. What's the best approach?

Best practices:

Use multiple provider configurations: Define multiple `provider "aws"` blocks with `alias`. Example:

```
provider "aws" {
  alias  = "prod"
  region = "us-east-1"
  profile = "prod-account"
}
```

- **Use workspaces:** `terraform workspace new prod` helps manage state separately per environment.
- **Use `for_each` or modules:** Deploy resources dynamically across accounts.
- **Use remote state:** Store state in an S3 backend with DynamoDB for locking to prevent conflicts.
- **Use automation tools:** Tools like Terraform Cloud, Terragrunt, or Atlantis help manage multiple accounts efficiently.

---

4. A junior engineer accidentally deleted the Terraform state file. How would you recover?

Recovery steps:

- **Check backend storage:** If using an S3 backend, check versioning to restore an older state file.
- **Use Terraform Cloud:** Terraform Cloud provides state snapshots that can be restored.
- **Recreate the state file:** If no backup exists, use `terraform import` to manually add existing resources back into state.
- **Reinitialize Terraform:** Run `terraform init` and use `terraform refresh` to sync state with actual resources.

---

5. Your Terraform plan shows unexpected changes even when no code modifications were made. How would you troubleshoot?

Troubleshooting steps:

- **Check Terraform version:** Run `terraform version` to ensure consistency across team members.

- **Check state drift:** Run `terraform refresh` to update the state with actual infrastructure.
- **Examine provider changes:** Cloud providers may have changed API responses; upgrade/downgrade provider versions if needed.
- **Check variable values:** Ensure correct variables are used with `terraform output`.
- **Inspect dependencies:** Use `terraform graph` to find implicit dependencies causing unexpected changes.

---

6. The Terraform execution is slow due to too many dependent modules. How can you optimize it?

Optimization strategies:

- **Use `depends_on` cautiously:** Minimize unnecessary dependencies.
- **Parallel execution:** Run `terraform apply -parallelism=10` to speed up execution.
- **Optimize modules:** Use lightweight modules with minimal dependencies.
- **Use data sources effectively:** Instead of creating redundant resources, use `data` blocks to fetch existing resources.
- **Cache provider plugins:** Run `terraform providers mirror` to avoid downloading providers repeatedly.

---

7. Your Terraform code is failing intermittently due to API rate limits from a cloud provider. How would you handle this?

Solutions:

Use `retryable_errors` in provider configuration:

```
provider "aws" {
  retryable_errors = ["Throttling", "RequestLimitExceeded"]
}
```

-
- **Reduce parallelism:** Run `terraform apply -parallelism=5` to limit concurrent API requests.
- **Use exponential backoff:** Wrap API calls with retries in a script.
- **Check service quotas:** Increase API quotas in AWS, Azure, or GCP if applicable.

---

8. You need to enforce strict security policies for infrastructure provisioning with Terraform. How would you achieve this?

Security best practices:

- **Use Sentinel or OPA (Open Policy Agent):** Define policy-as-code to enforce security rules.
- **Enforce RBAC:** Restrict permissions using IAM policies or Terraform Cloud workspaces.
- **Enable state file encryption:** If using S3 backend, enable server-side encryption.
- **Scan Terraform code:** Use security tools like `tfsec`, `checkov`, or `terrascan`.
- **Use least privilege principle:** Define IAM roles with minimal required permissions.

---

9. Your Terraform deployment in a multi-cloud environment is facing latency issues. How would you optimize it?

Optimization strategies:

- **Reduce API calls:** Use `data` sources instead of creating duplicate resources.
- **Run Terraform in the same region:** Deploy Terraform execution closer to cloud resources.
- **Parallel execution:** Increase `-parallelism` for faster deployment.
- **Use Terraform Cloud:** Terraform Cloud runs operations in a distributed manner, reducing local latency.
- **Optimize remote state access:** If using S3, enable Transfer Acceleration for faster state updates.

---

10. A Terraform apply failed midway, leaving some resources partially created. How do you fix this while ensuring consistency?

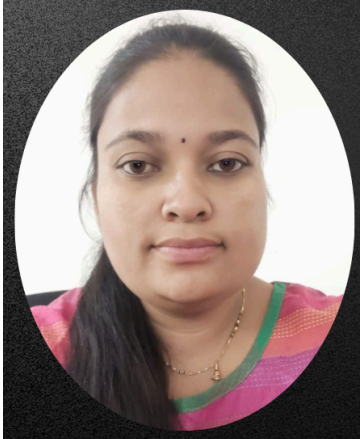
Steps to recover:

- **Identify failed resources:** Run `terraform state list` to check affected resources.
- **Manually verify existing resources:** Check cloud provider console to see which resources exist.
- **Refresh state:** Run `terraform refresh` to update state with existing infrastructure.
- **Retry Terraform apply:** Run `terraform apply` again, ensuring state is consistent.
- **Manually remove broken resources (if needed):** Delete partial resources manually and run `terraform apply` again.

GET IN TOUCH

Let's Work Together

TECHOPSBYSONALI



sonali.k593@gmail.com

CareerByteCode
Learning Made simple

SONALI KURADE

LETS CONNECT