DevOps — Terraform (25 Questions)

Q1: Terraform apply in prod fails with `Error locking state` because another apply is in progress, but that process crashed.

Answer:
If using an S3 backend with DynamoDB lock table, manually remove the stale lock item in DynamoDB **only afterconfirmingnootherappliesarerunning** . Then re-run. Consider -lock-timeout to wait for locks in future.

SamplePoints:

- Never delete lock unless sure it's stale.
- DynamoDB item removal unblocks state.
- Use -lock-timeout for busy teams.

Example Code:

```
aws dynamodb delete-item --table-name tf-lock --key '{"LockID": {"S": "prod/terraform.tfstate-md5"}}'
```

Q2: You must migrate manually created AWS resources into Terraform without downtime.

Answer:
Use terraform import to bring existing resources into state, then run terraform plan to ensure config matches reality. Avoid changing properties that would force replacement unless planned.

Sample Points:

- Import keeps IDs; no recreation.
- Align config with live settings before apply.

- Plan first to detect drifts.
  Example Code:

```
terraform import aws_s3_bucket.mybucket my-bucket-name
```

Q3: Remote backend state on S3 shows drift after manual console changes.

Answer:
Run `terraform plan` to see differences. If drift is intended, update `.tf` files; if not, apply to revert. Use `terraform state rm` for resources no longer managed.
SamplePoints:

- Plan → decide to update config or infra.
- State rm for unmanaged resources.
- Avoid console edits in IaC environments.
  Example Code:

```
terraform state rm aws_security_group.unmanaged
```

Q4: Terraform plan for a new VPC fails due to CIDR block overlap.

Answer:
Adjust CIDRs to non-overlapping ranges or use cidrsubnet() function for consistent derivation. In multi-env setups, store base CIDR in a shared tfvars file.
Sample Points:

- Overlap blocks VPC creation.
- Derive subnets to avoid manual mistakes.
- Centralize network config.
  Example Code:

```
cidrsubnet(var.vpc_cidr, 4, count.index)
```

Q5: Applying a module update forces recreation of critical RDS instance.

Answer:
Check if immutable properties (e.g., `allocated_storage` in some engines) changed. Use `lifecycle{prevent_destroy = true }` and perform updates with in-place changes only.

Sample Points:

- Prevent destroy for prod DBs.
- Review module changelog before apply.
- Change params in maintenance window.

Example Code:

```
lifecycle {
  prevent_destroy = true
}
```

Q6: Terraform workspace `prod` accidentally applied dev resources.

Answer:
Separate state files per environment instead of relying solely on workspaces, or use different backend key paths. Enforce var.environment as a required var.

Sample Points:

- Workspaces not isolation alone.
- Different backend keys safer.
- Always pass environment explicitly.

Example Code:

```
backend "s3"{
```

```
 key = "prod/terraform.tfstate"
}
```

Q7: `terraform destroy` in staging deleted a shared VPC used by prod.

Answer:

Add prevent_destroy lifecycle on shared resources, and separate shared infra into its own state. Enforce tagging and IAM policy to block deletes in shared projects.

Sample Points:

- Lifecycle protect critical shared assets.
- Separate state files for shared vs env-specific.
- IAM denies for delete API calls in shared.

Example Code:

```
lifecycle { prevent_destroy = true}
```

Q8: Applying infra in CI/CD fails due to missing provider creds.

Answer:

Inject creds via environment variables or use workload identity federation (OIDC) for ephemeral credentials in CI. Avoid hardcoding in tfvars.

Sample Points:

- CI pipeline must set creds securely.
- Prefer OIDC over long-lived keys.
- Use TF_VAR_ prefix for vars.

Example Code:

```
export AWS_ROLE_ARN=arn:aws:iam::123:role/tf-role
terraform apply
```

Q9: Terraform state file contains secrets.

Answer:
State stores all computed values. Use sensitive = true in variables, enable state encryption (S3 SSE or Vault backend), and limit state access via IAM.

Sample Points:

- Mark vars sensitive.
- Encrypt state at rest.
- Restrict read access.

Example Code:

```
variable "db_password"{
 type       = string
 sensitive = true
}
```

Q10: Resource creation fails due to provider version mismatch in team members' local setups.

Answer:

Pin provider versions in `required_providers` and commit `.terraform.lock.hcl` to repo.

Sample Points:

- Pin provider version for consistency.
- Lock file ensures deterministic builds.
- Upgrade via `terraforminit-upgrade`.

Example Code:

```
terraform {
  required_providers {
    aws = { source = "hashicorp/aws", version = "~> 5.0"}
 }
}
```

Q11: Terraform plan is slow due to thousands of resources.

Answer:
Use -target for scoped applies when safe, split resources into multiple states/modules, and enable provider-side filtering.

Sample Points:

- Target only changed modules.
- Split state for scale.
- Provider filters speed up queries.

Example Code:

```
terraform plan -target=module.network
```

Q12: Need to roll back to a previous infrastructure version after failed deployment.

Answer:
Check state history in backend (e.g., S3 versioning), download older state, and `terraform apply` with matching config.

Sample Points:

- Keep state versioning enabled.
- Rollback means re-applying older config.
- Document rollback procedure.

Example Code:

```
aws s3 cp s3://bucket/prod.tfstate version-id=xyz ./terraform.tfstate
```

Q13: Developers must create S3 buckets only via Terraform, never manually.

Answer:
Use AWS Config or Cloud Custodian to detect manual resources, and enforce creation via CI pipelines with Terraform. Block manual create API via IAM condition on aws:ViaAWSService.

Sample Points:

- Guardrails + detection for manual drift.
- CI as only entrypoint for IaC changes.
- IAM denies outside Terraform role.

Example Code:

```
"Condition":{"StringNotEquals":{"aws:CalledVia":"cloudformation.amazonaws.com"}}
```

Q14: Team often forgets to run `terraform fmt` before committing.

Answer:
Add pre-commit hook to run `terraform fmt -recursive` and fail commit on diff.

Sample Points:

- Pre-commit enforces formatting.
- CI can also check.
- Consistent style reduces churn.

Example Code:

```
#!/bin/sh
terraform fmt -recursive -check || exit1
```

Q15: Need to generate unique names for resources across environments.

Answer:
Use `format()`with `var.environment` and random provider (`random_id`) to suffix names.

Sample Points:

- Combine env + random for uniqueness.
- Avoid collisions in global namespaces.
- Keep naming convention consistent.

Example Code:

```
resource "aws_s3_bucket" "b" {
  bucket = "${var.environment}-${random_id.suffix.hex}"
}
```

Q16: Provider credentials expire mid-apply for long deployments.

Answer:
Use short modules with smaller applies or refresh creds via token renewal during apply. For AWS, use session durations > apply time.

Sample Points:

- Split large applies.
- Extend token lifetime.
- Orchestrate apply steps.

Example Code:

```
aws sts assume-role --duration-seconds 3600
```

Q17: Need to avoid recreating immutable resource IDs when only tags change.

Answer:
Use `ignore_changes = [tags]` in lifecycle.

Sample Points:

- Ignore tag changes for immutable IDs.
- Helps for vendor-managed resources.

- Still apply tags manually if needed.
  Example Code:

```
lifecycle {
  ignore_changes = [tags]
}
```

Q18: Must run a script after resource creation but before marking apply complete.

Answer:
Use `local-exec` provisioner with `when = create`. Keep idempotent.
Sample Points:

- Provisioners last resort.
- Idempotent scripts avoid drift.
- Prefer user_data/cloud-init when possible.
  Example Code:

```
provisioner "local-exec" {
  when    = create
  command = "echo ${self.id} >> created.log"
}
```

Q19: Two resources depend on each other cyclically in config.

Answer:
Break the cycle by creating one with minimal config, outputting an ID, and updating the other in asecond apply. Or use depends_on to force order.

Sample Points:

- Break cycles into stages.
- Use depends_on explicitly.

- Avoid mutual references.
  Example Code:

```
depends_on = [aws_security_group.sg]
```

Q20: Terraform in CI/CD must plan without exposing secrets in logs.

Answer:
Mark variables as sensitive and use `terraform plan -out=planfile` without showing values.
Sample Points:

- Sensitive vars mask in logs.
- Use planfile to hide secrets.
- Avoid -var with secrets inline.
  Example Code:

```
terraform plan -out=planfile
```

Q21: Must ensure all resources are tagged with `Owner` and `Environment`.

Answer:
Use `default_tags` in provider config (AWS >=3.38), or a tagging module. Validate via `terraform validate` + custom rules.
Sample Points:

- Default tags enforce globally.
- Validation prevents missing tags.
- Use TF Cloud policy sets.
  Example Code:

```
provider "aws"{
default_tags{
```

```
    tags = { Owner = var.owner, Environment = var.environment}
 }
}
```

Q22: Multi-cloud project requires different providers in same root module.

Answer:
Alias providers (`provider "aws" { alias = "east" }`) and pass into resources/modules.
Sample Points:

- Aliases for multi-provider configs.
- Pass providers explicitly to modules.
- Keep creds separate per alias.
  Example Code:

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

Q23: Remote state backend migration from local to S3.

Answer:

Run `terraform init -migrate-state` after defining new backend block.
Sample Points:

- Migrate state without losing resources.
- Backend config in .tf file.
- Plan after migration to verify.
  Example Code:

```
terraform init -migrate-state
```

Q24: Must enforce no `terraform apply` in prod without PR approval.

Answer:
Use TF Cloud/Terragrunt with run tasks, or CI pipeline with approval stage before `apply` in prod workspace.
Sample Points:

- Manual gate before prod apply.
- PR review ensures compliance.
- Use TF Cloud policy-as-code.
  Example Code:

```
- stage: Approval
jobs:
    - manual: true
```

Q25: Need to refactor large root module into reusable modules without downtime.

Answer:
Refactor incrementally:

1. Extract a resource to new module.
2. Use terraform state mv to match new address.
3. Plan/apply with no changes. Repeat.
   Sample Points:

- State mv avoids recreation.
- One resource/module at a time.
- Validate after each step.
  Example Code:

```
terraform state mv aws_s3_bucket.old module.new.aws_s3_bucket.old
```