






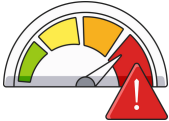






10 Terraform Scenarios

Every DevOps Engineer Must Know

This document covers key scenarios every DevOps engineer should know when using Terraform. It highlights common pitfalls and provides guidance to avoid infrastructure issues and ensure smoother deployments.

Terraform Scenarios for DevOps

				
Backend Access Loss	State File Deletion	Service Quotas	Resource Removal	Provider API Changes
				
API Rate Limits	Failed Applies	Concurrent Applies	Manual Changes	Circular Dependencies



1. What if your Terraform state file is deleted?

Disaster mode on! Terraform loses track of its deployments and tries to rebuild everything from scratch—risking conflicts and duplicate resources. Always back up your state file!



HashiCorp

Terraform

Terraform State File Deletion Consequences





2. What if two engineers run terraform apply at the same time?

State file clash! Without proper locking, Terraform risks state corruption, leading to unpredictable infrastructure drift. Use remote state with locking!

How to manage Terraform state files to prevent corruption?



Use Remote State with Locking

Ensures state file integrity and prevents corruption during simultaneous applies.



Use Local State without Locking

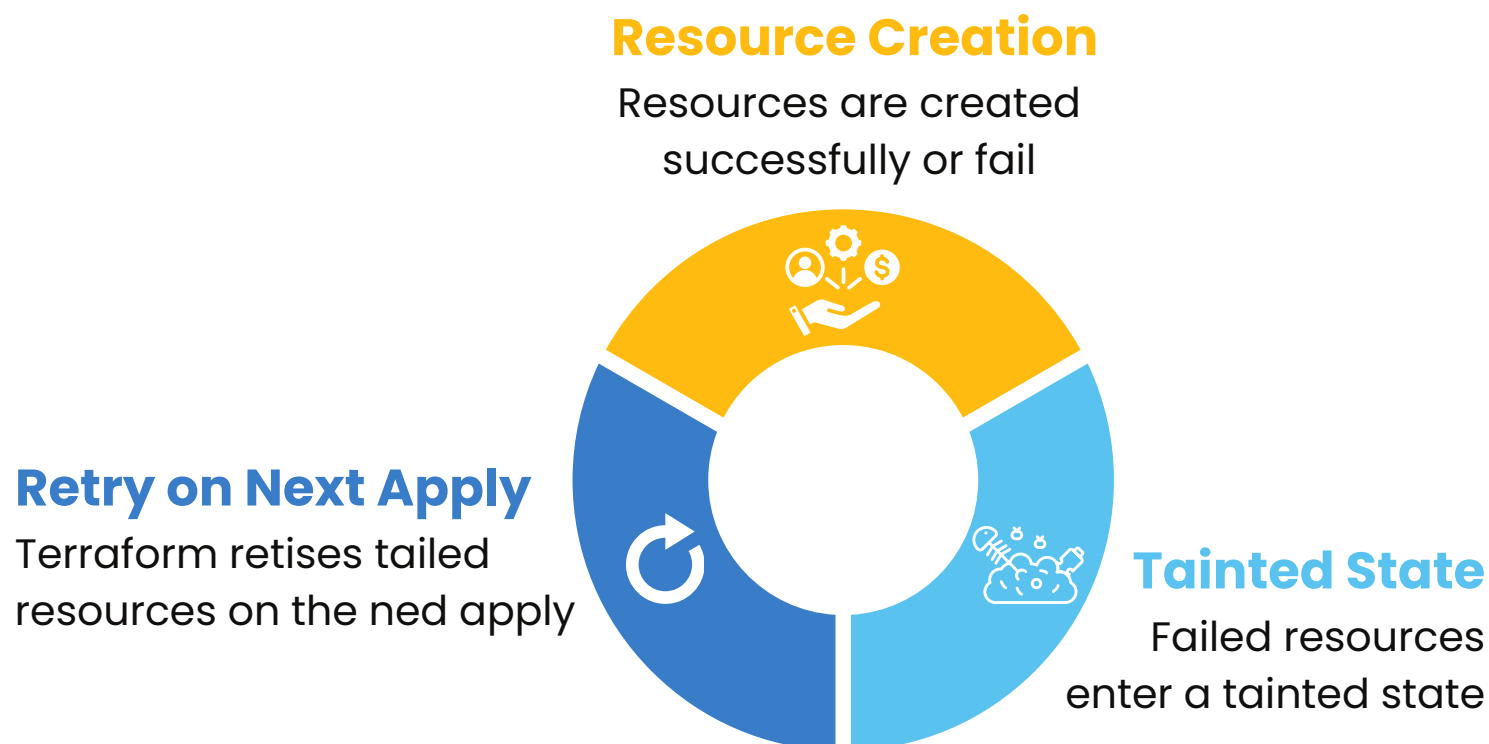
Risk of state corruption and infrastructure drift due to simultaneous applies.



3. What if a Terraform apply fails halfway?

Partially deployed chaos! Successfully created resources stay live, but failed ones remain in a tainted state. Terraform will retry the failed ones on the next apply.
Know how to handle tainted resources!

Handling Tainted Resources in Terraform

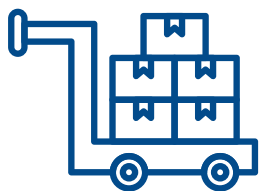




4. *What if AWS API rate limits are hit during deployment?*

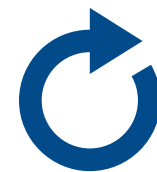
Throttling nightmare! Terraform retries a few times but then fails. Resources already created stay live, leaving you with a half-baked deployment. Optimize batch sizes and handle retries smartly!

How to handle AWS API rate limits during Terraform deployment?



Optimize Batch Sizes

Reduces the likelihood of hitting API rate limits by spreading requests more evenly



Smart Retry Handling

Ensures that retries are managed effectively to minimize deployment disruptions.



5. What if Terraform plan shows no changes, but someone manually modified infra?

Invisible drift detected! Terraform won't notice external changes unless you run terraform refresh. Future applies might overwrite manual updates without warning. Always detect drift before applying!

Terraform Drift Detection Process



Detect Invisible Drift

Identify external changes. not recognized by Terraform



Run Terraform Refresh

Execute command to update Terraform's state



Update Terraform State

Modify state to reflect current infrastructure



Apply Changes

implement updates to the infrastructure



Prevent Overwrite

Ensure manual changes are not lost





6. What if you remove a resource from your Terraform config?

Terraform sees it as unwanted! On the next apply. Terraform will destroy it unless you manually remove it from the state [terraform state rm). Use prevent destroy to avoid accidental deletions!

Terraform Drift Detection Process



Manual Infra Change

An infrastructure component is modified manually



Terraform Apply Detection

Terraform detects the manual change during the apply process



State Update

Terraform updates its state to reflect the current infrastructure



Plan Execution

Terraform executes the plan to apply changes



Resource Marked for Destruction

The manually changed resource is marked for destruction



Prevent Destruction Applied

The prevent, destroy argument is applied to safeguard the resource



Resource Preservation

The resource is preserved and not destroyed.

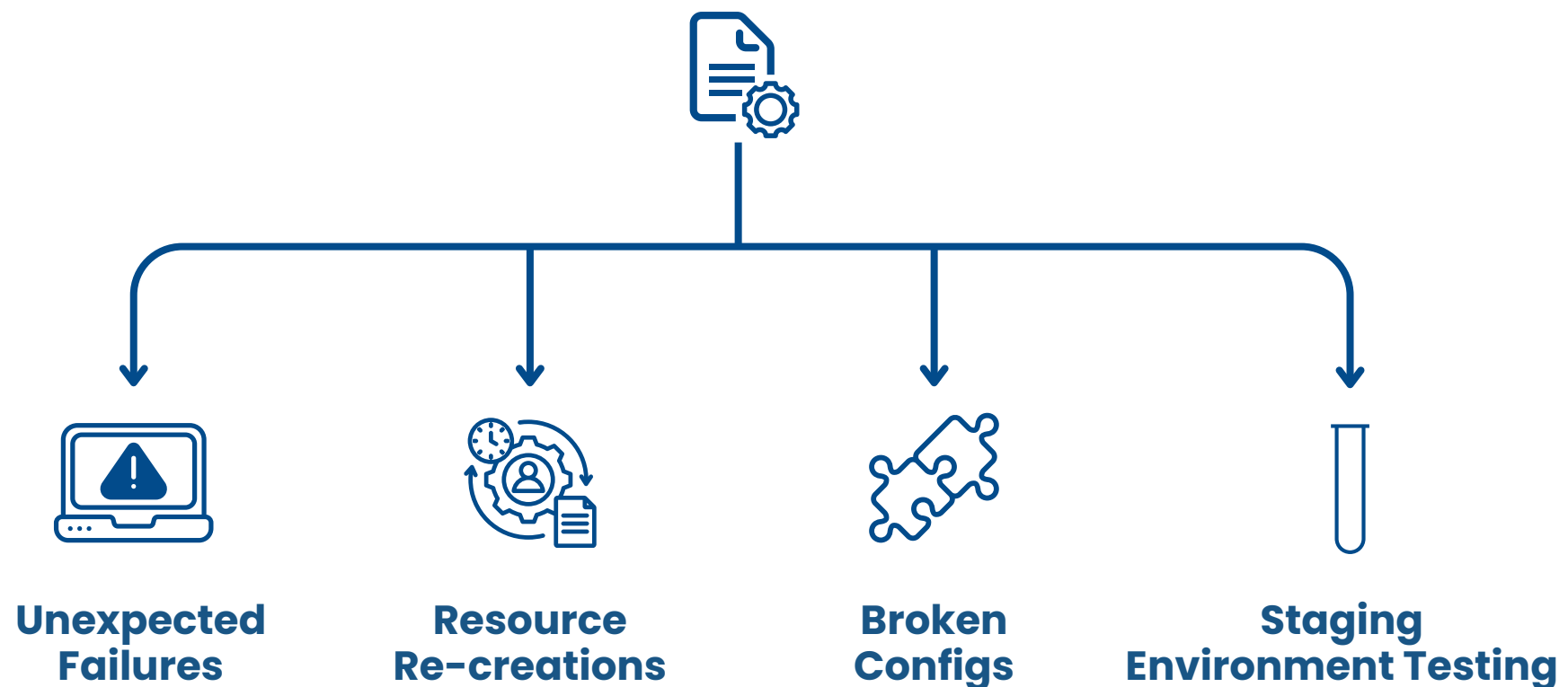




7. *What if a provider API changes between Terraform versions?*

Version mismatch headaches! Unexpected failures, resource re-creations, or broken configs might occur. Always test updates in a staging environment first!

Managing Terraform Version Mismatches

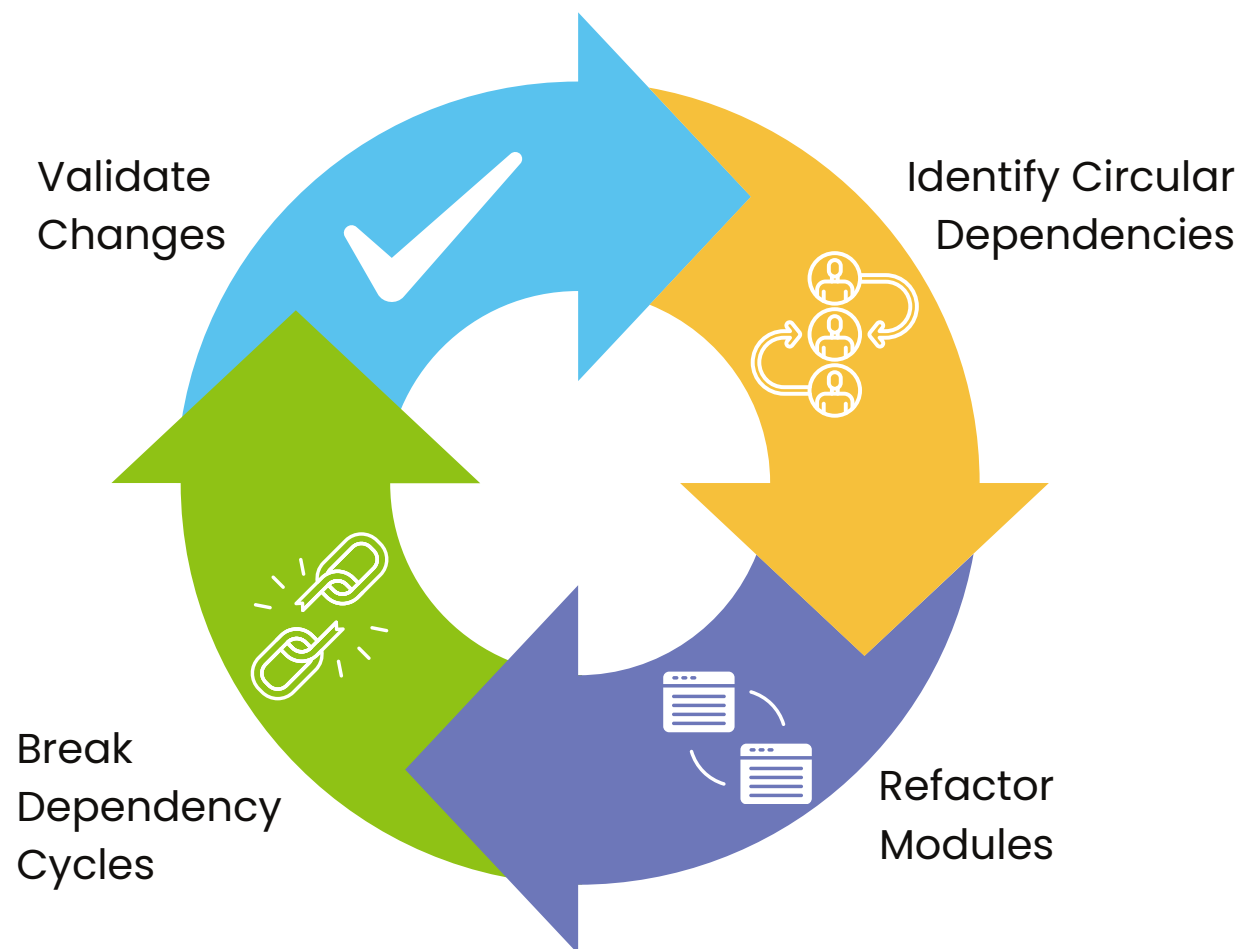




8. *What if your Terraform modules have circular dependencies?*

Infinite loop of doom! Terraform refuses to proceed, throwing dependency cycle errors. Refactor your modules and break cycles!

Breaking Terraform Dependency Cycles





9. What if you exceed AWS service quotas during deployment?

Quota exceeded! Terraform marks resources as failed, leaving your deployment incomplete. Pre-check quotas before large deployments!

How to handle exceeding AWS service quotas during Terraform deployment?



Pre Check Quotas

Ensures quotas are not exceeded, preventing deployment issues

Request Quotas Increase

Seeks approval to increase quotas, allowing larger deployments

Adjust Deployment

Modifies the deployment to stay within quota limits, ensuring completion



10. What if you lose access to the remote backend storing your Terraform state?

Loss of access can lead to a complete inability to manage your infrastructure. Ensure you have proper access controls and backup strategies in place to avoid this scenario.

Mitigating Terraform State Access Loss

