Module 6: Import and State Surgery

Managing Terraform State Without Recreating Infrastructure

MODULE 6

Import and State Surgery (Terraform)

This module covers the essential techniques for managing Terraform state when working with existing infrastructure, renaming resources, and transferring ownership–all without disrupting your running systems.

Module 6 Overview

Import and State Surgery

Managing Terraform State Without Recreating Infrastructure

What this module covers:

- Bringing existing resources into Terraform
- Renaming resources safely
- Removing resources from Terraform without deleting them
- Best practices for working with Terraform state

Learning Objectives

By the end of this module, you will be able to:

Use import blocks

Bring existing resources under Terraform

Use moved blocks

Rename or reorganize resources

Use removed blocks

Stop managing resources safely

✓ Identify when each state operation should be used

! Understand risks and trade-offs of state surgery

The Challenge

Why Do We Need State Surgery?

Real-world scenario:

You join a team where infrastructure was created:

- Manually in AWS Console
- Using lost scripts
- By an old tool

Now your org wants **Terraform adoption** without downtime.

💡 Recreating everything is risky and time-consuming.

Common Challenges

Common Problems

Problem 1 – Existing Infrastructure

Terraform tries to **create duplicates** because it doesn't know resources already exist.

Problem 2 – Renaming Resources

Changing a resource name causes Terraform to **destroy and recreate** it.

Problem 3 – Transferring Ownership

You need to **stop managing** a resource but keep it running.

Solutions

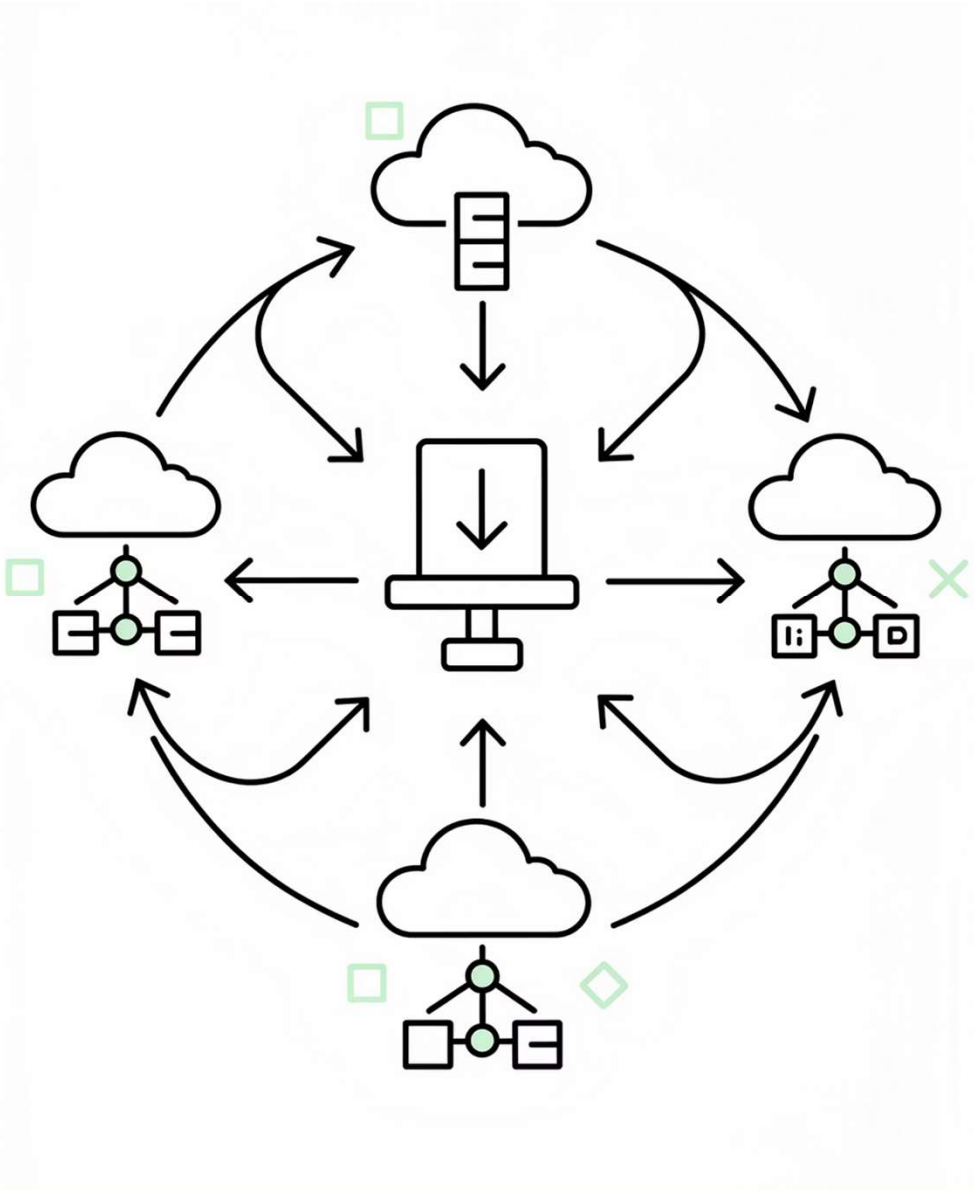
The Solutions

Problem	Terraform Solution
Existing resources	Import Block
Renaming / Refactoring	Moved Block
Stop managing but keep resource	Removed Block

These features let you change **state safely without touching real infrastructure.**

SECTION 1

IMPORT BLOCKS SECTION



Import Blocks

What is an Import Block?

Introduced in **Terraform 1.5**

A **declarative way** to bring existing resources into Terraform state.

```
import {
 to = resource_type.resource_name
 id = "real-world-resource-id"
}
```

Comparison

Why Import Blocks (Instead of CLI command)?

Legacy terraform import	Import Block
One-time CLI command	Declarative in code
No preview	Works with terraform plan
Not version controlled	Can be reviewed in PR
Manual per resource	Supports batch imports

Example Walkthrough

Example: Import an S3 Bucket (Step 1)

Write matching resource config first

Before importing, you need to define the resource configuration in your Terraform code that matches the existing infrastructure.

```
resource "aws_s3_bucket" "logs" {
 bucket = "dev-logs-bucket"

 tags = {
 Name = "dev-logs-bucket"
 Environment = "dev"
 Purpose = "logs"
 }
}
```

Example Walkthrough

Example: Import Block (Step 2)

After defining your resource configuration, create an import block that references the resource and provides its real-world ID.

```
import {
 to = aws_s3_bucket.logs
 id = "dev-logs-bucket"
}
```

Example Walkthrough

Preview Import (Step 3)

Run:

```
terraform plan
```

Safe preview before touching state.

You will see:

```
aws_s3_bucket.logs will be imported

Plan: 1 to import, 0 to add,
 0 to change, 0 to destroy.
```

Example Walkthrough

Apply & Cleanup (Step 4–5)

Apply Changes

```
terraform apply
```

After success

🧹 Remove import block (one-time use)

```
rm imports.tf
```

Post-Import

Configuration Drift

After import:

```
terraform plan
```

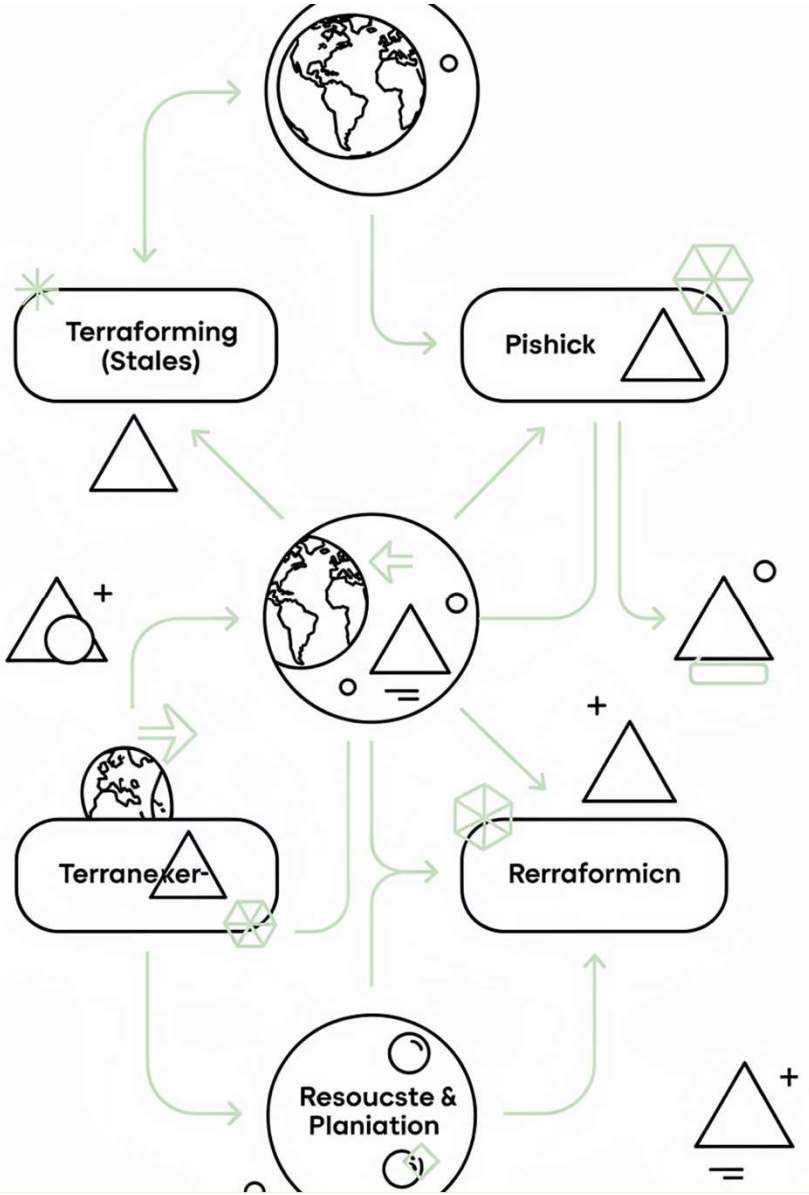
If Terraform wants to update something → drift exists

You must decide:

- ✓ Update Terraform code OR
- ✓ Update real resource via apply

SECTION 2

MOVED BLOCKS SECTION



Moved Blocks

What is a Moved Block?

Used when **renaming or reorganizing resources**

```
moved {
 from = aws_s3_bucket.old_name
 to = aws_s3_bucket.new_name
}
```

Benefits

Why Use Moved Blocks?

❌ *Without moved block:*

- Terraform destroys old resource
- Creates new resource

✓ *With moved block:*

- Terraform **only updates state**
- Infrastructure remains untouched

Example

Example: Renaming Resource

1

Old:

```
resource "aws_s3_bucket" "legacy_logs" {
  ...
}
```

2

New:

```
resource "aws_s3_bucket" "logs" {
 ...
}
```

Example Continued

Add the Moved Block

```
moved {
  from = aws_s3_bucket.legacy_logs
  to   = aws_s3_bucket.logs
}
```

Key Point: The moved block tells Terraform to update its state file to reflect the new name, without making any changes to the actual infrastructure.

Then delete the old resource block.