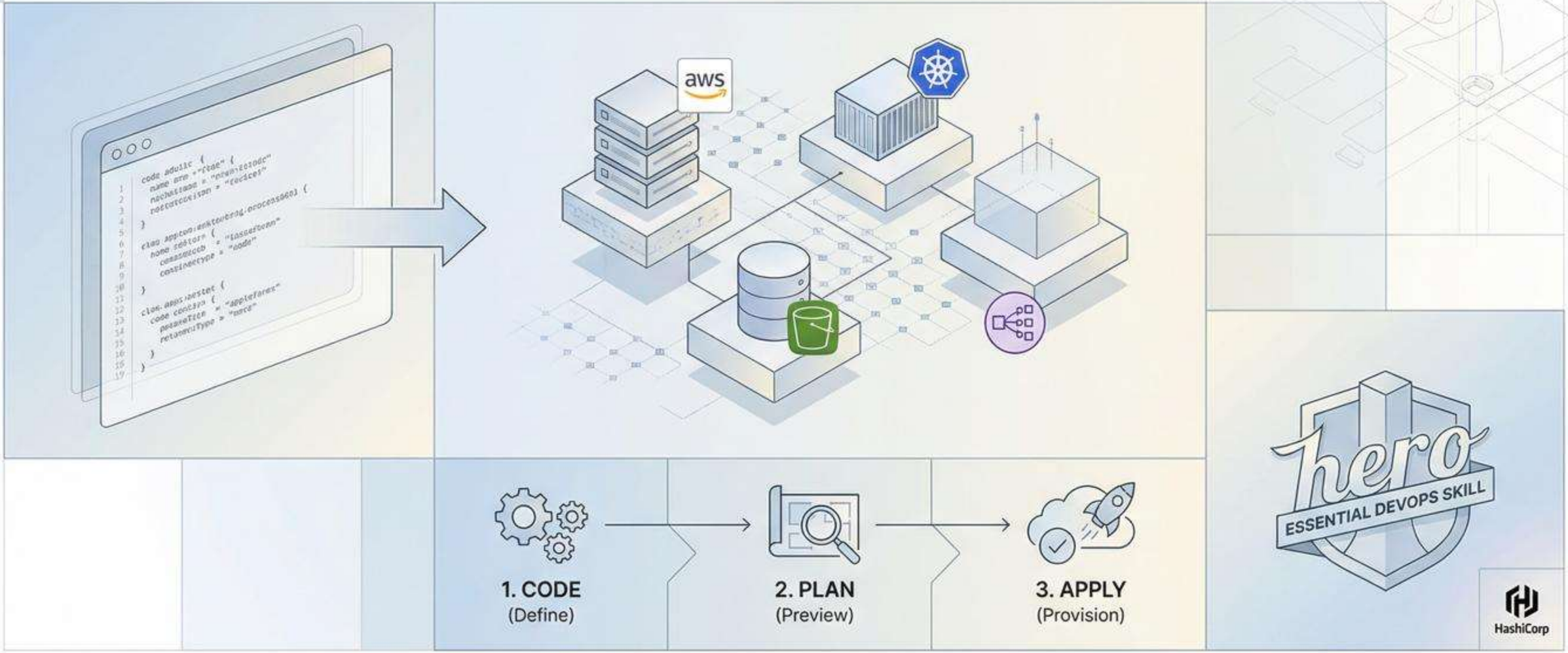
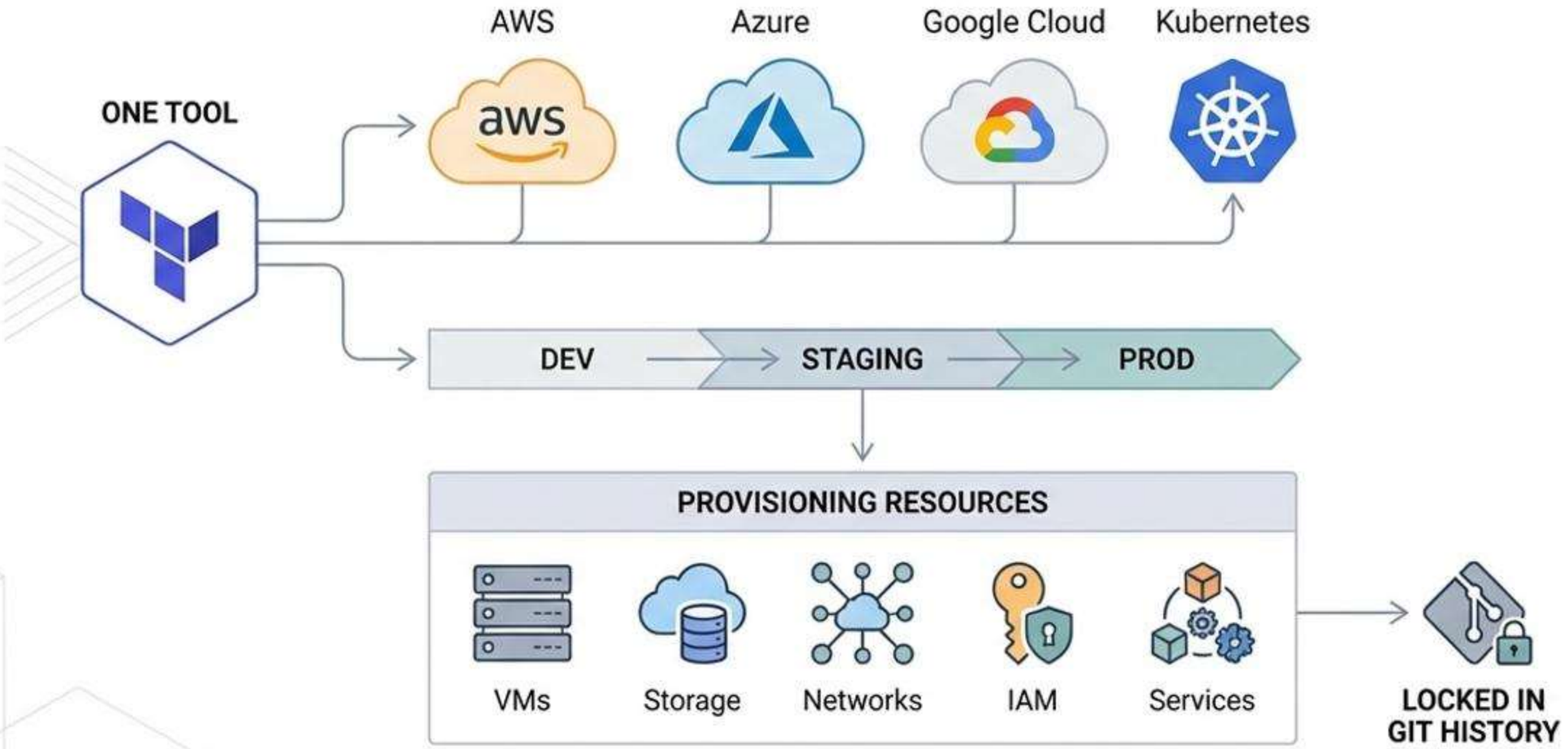
TERRAFORM – ZERO TO HERO

Infrastructure as Code: Automating Cloud Deployments with Precision



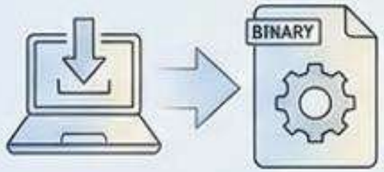
TERRAFORM: AUTOMATION AT SCALE & UNIFIED WORKFLOW

Enforces Identical Environments from Dev to Prod, Supports AWS, Azure, GCP & Kubernetes



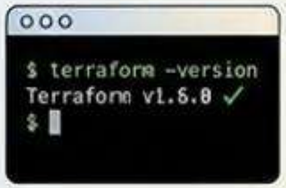
TERRAFORM & AWS: SETUP & AUTHENTICATION

LOCAL TERRAFORM SETUP



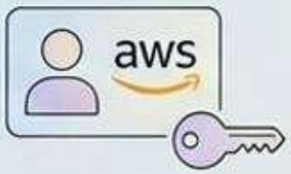
1. Download Binary
Download the single binary

2. Add to PATH
Add it to system PATH



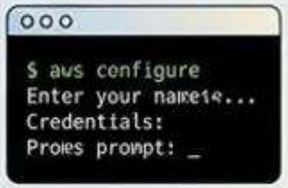
3. Verify Installation
Verify with
`terraform -version`

AWS SETUP FOR TERRAFORM



1. Create IAM User
Create an IAM user with programmatic access

2. Access Keys
Obtain Access Key & Secret Key



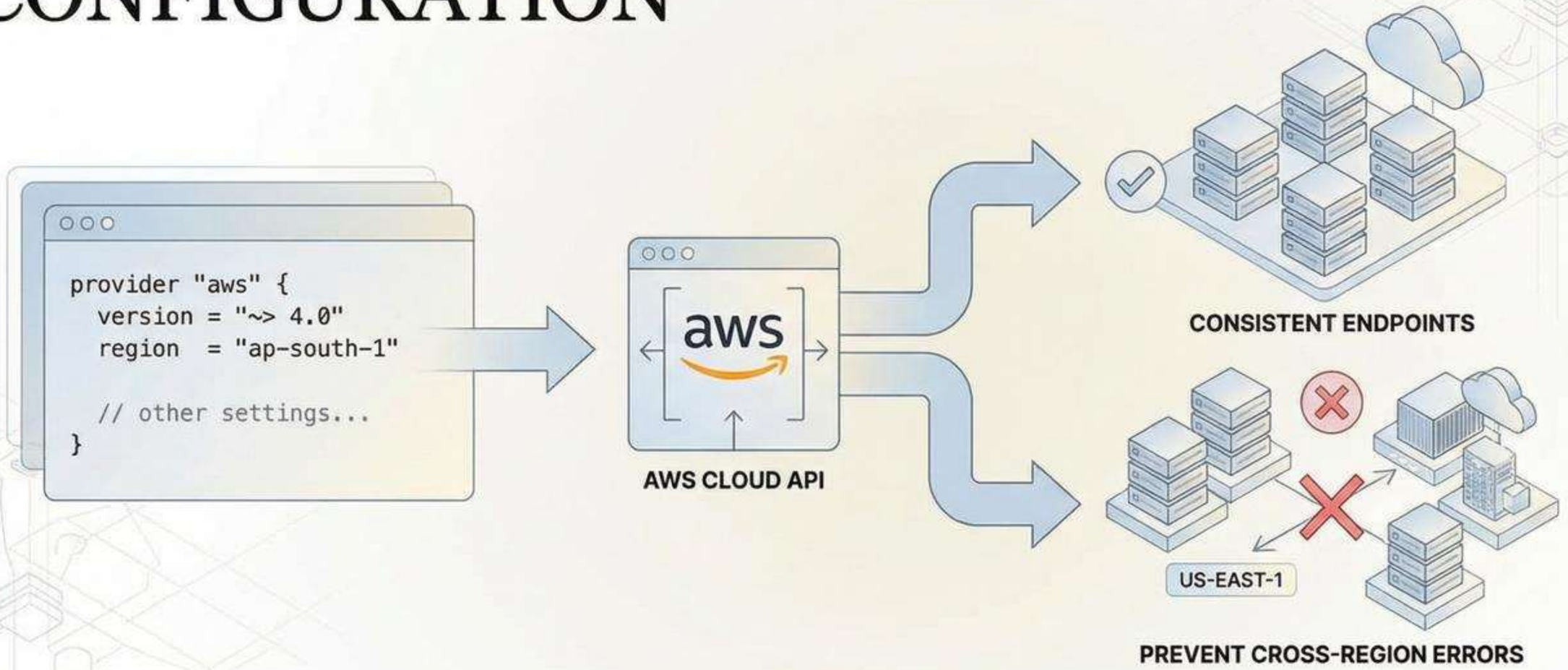
3. Configure AWS CLI
Run `aws configure` to store credentials

4. Ensure Access
Ensure CLI can list regions for Terraform authentication

Terraform Authenticates & Creates Resources on AWS

HashiCorp

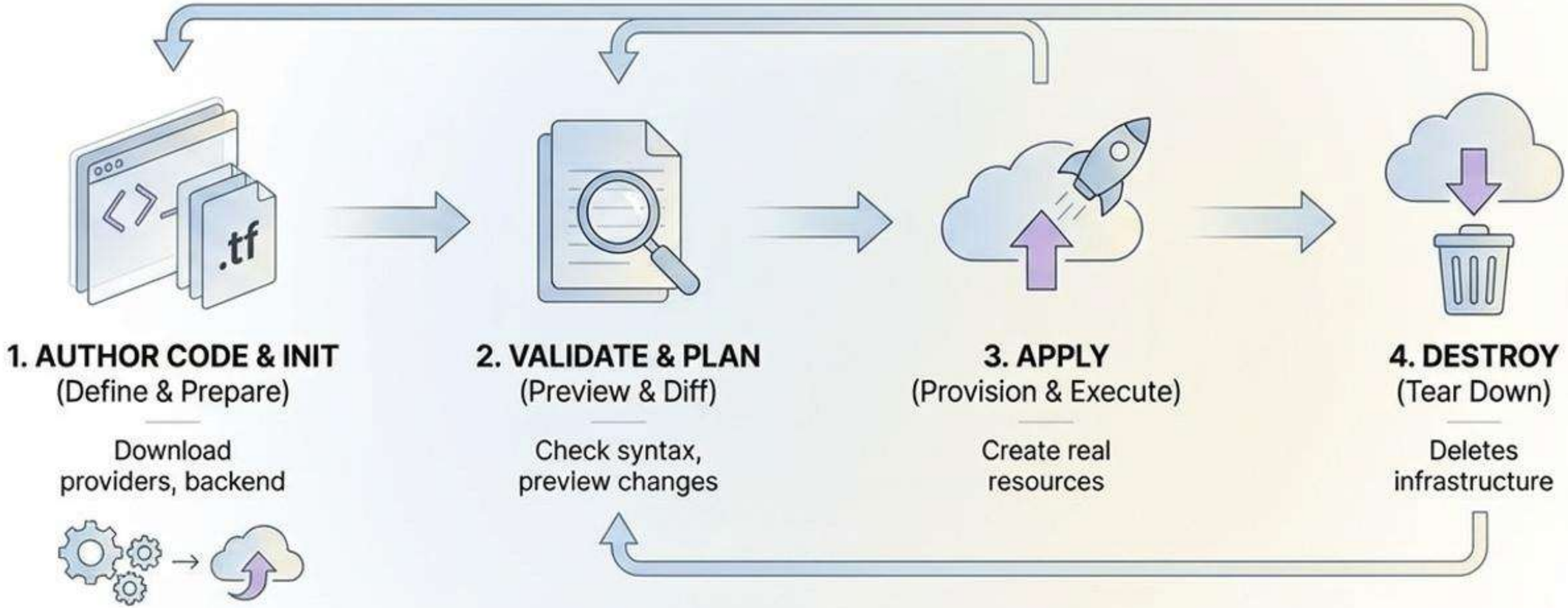
TERRAFORM AWS PROVIDER CONFIGURATION



The **provider block** acts as the central interface to the cloud API, pinning versions and setting default regions for reliable, uniform deployments across all resources.

Terraform Deployment Loop:

The Heartbeat of Safe Infrastructure



This four-step loop is the heartbeat of every safe Terraform deployment, ensuring predictable and managed infrastructure changes.

HashiCorp

CORE TERRAFORM COMMANDS:
Mastering Control of Any Environment

init
Installs providers and modules.

validate
Checks HCL syntax.

plan
Shows a diff of changes.

apply
Executes changes with approval.

destroy
Removes all resources.

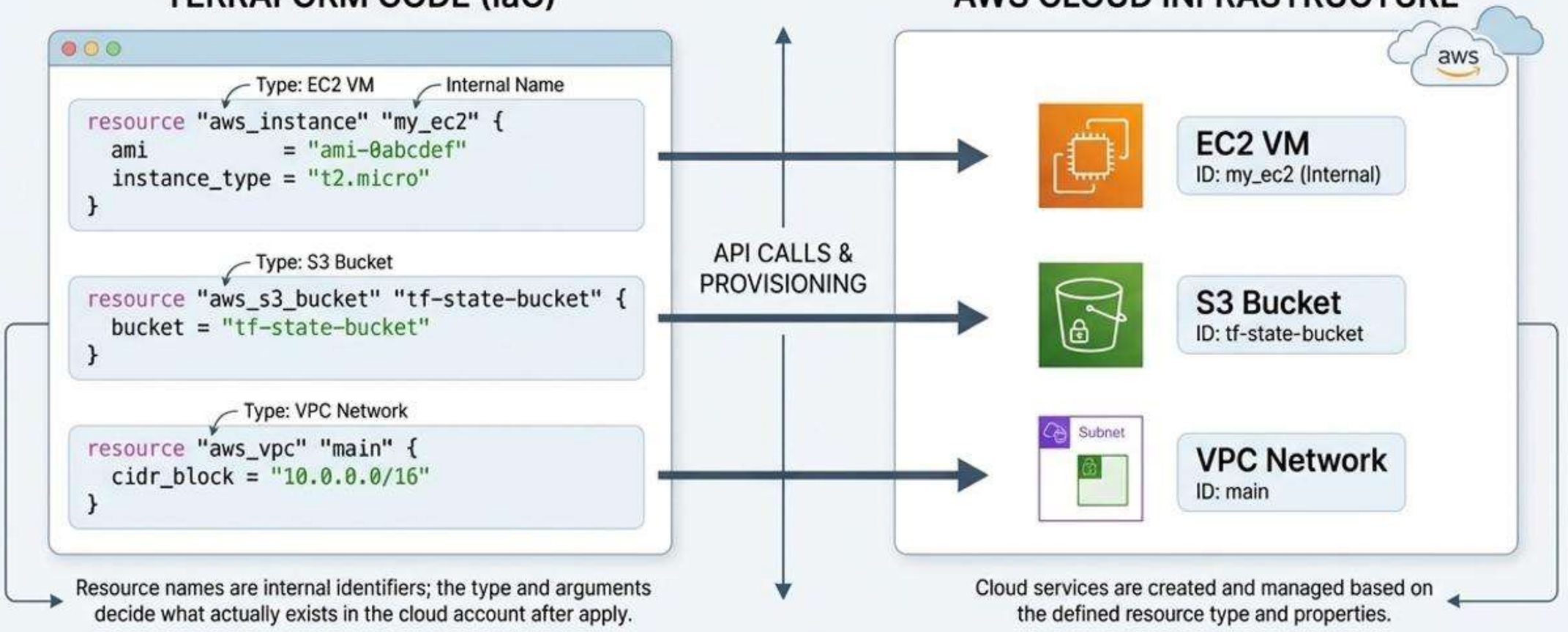
fmt
Rewrites style for consistency.

state list
Locates resources in state.

Master these seven commands to control any environment.

TERRAFORM RESOURCE MAPPING: CODE TO CLOUD

Each resource block becomes one API call, translating Terraform definitions into actual AWS services.

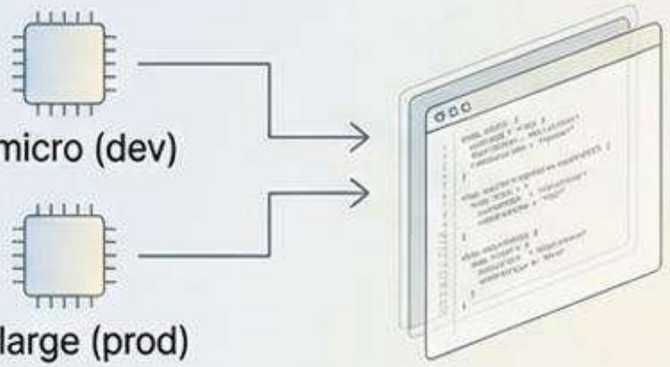


Terraform's resource abstraction simplifies cloud management by treating infrastructure as code, enabling precise and reproducible deployments.

TERRAFORM – ZERO TO HERO: VARIABLES & OUTPUTS

VARIABLES (Input)

Parameterize Templates:
Same code for t2.micro or m5.large



- Supply defaults
- Enforce types (string, number, list, map)
- Add descriptions

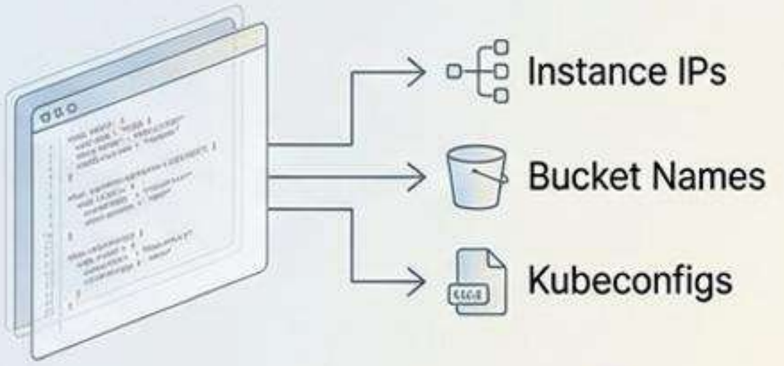
```
variable "instance_type" {
  type        = string
  default     = "t2.micro"
  description = "The type of EC2 instance to launch"
}
```

Terraform Code & Apply

Data Flow:
Configure
→ Deploy
→ Consume

OUTPUTS (Output)

Return Values:
For downstream consumption



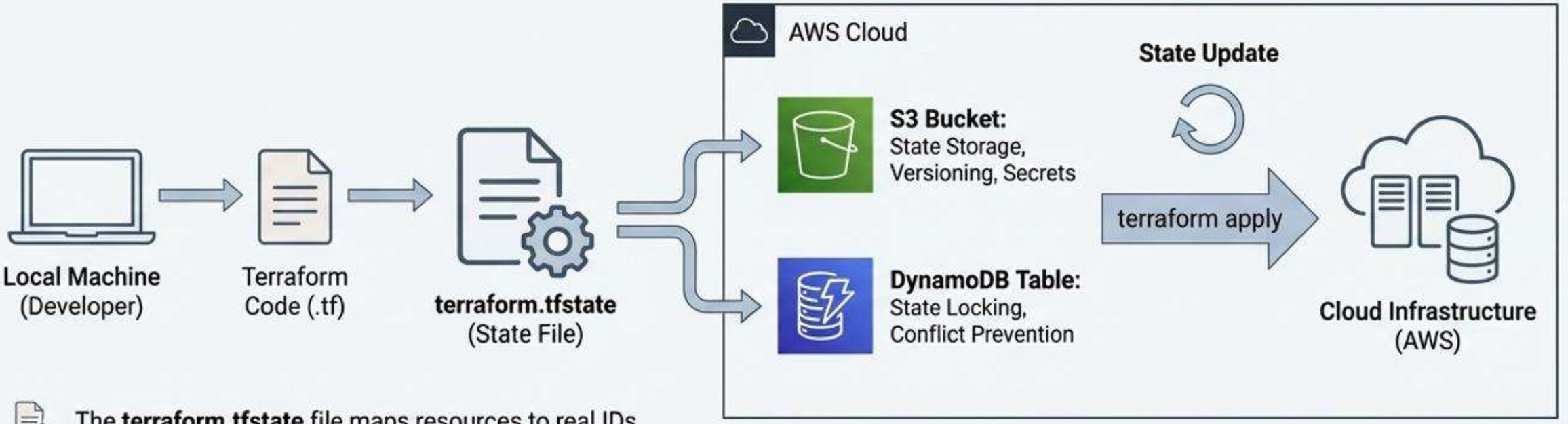
- Consume immediately
- Use in CI/CD pipelines
- Share across modules


```
output "public_ip" {
  value = aws_instance.app_server.public_ip
}
```

HashiCorp

TERRAFORM REMOTE STATE MANAGEMENT

Secure, Collaborative, and Reliable Infrastructure State



- The **terraform.tfstate** file maps resources to real IDs.
- Storing it in an S3 bucket with versioning and **DynamoDB table locking** prevents team conflicts, enables rollback, and keeps secrets off laptops.
- Every **apply** updates the remote state, ensuring a single source of truth.

KEY BENEFITS

COLLABORATION	SECURITY	RELIABILITY
PREVENTS CONFLICTS & ENABLES TEAMWORK	SECRETS OFF LAPTOPS & SECURE STORAGE	VERSIONING, ROLLBACK & SINGLE SOURCE OF TRUTH

HashiCorp

Terraform + AWS: EC2 & Security Group Example

Infrastructure as Code: Declarative Provisioning with One Command

1. CODE DECLARATION (.tf files)

aws_instance

```
resource "aws_instance" "web_server" {
  ami                    = "ami-0c55b159cbfafe1f0"
  instance_type          = "t2.micro"
  vpc_security_group_ids =
    [aws_security_group.ssh_access.id]
  tags = {
    Name = "Terraform-Web-Server"
  }
}
```

aws_security_group

```
resource "aws_security_group" "ssh_access" {
  name        = "allow_ssh"
  description = "Allow SSH inbound traffic"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

2. WORKFLOW & OUTPUT

terraform apply ▶ → AWS Cloud

OUTPUT VARIABLES

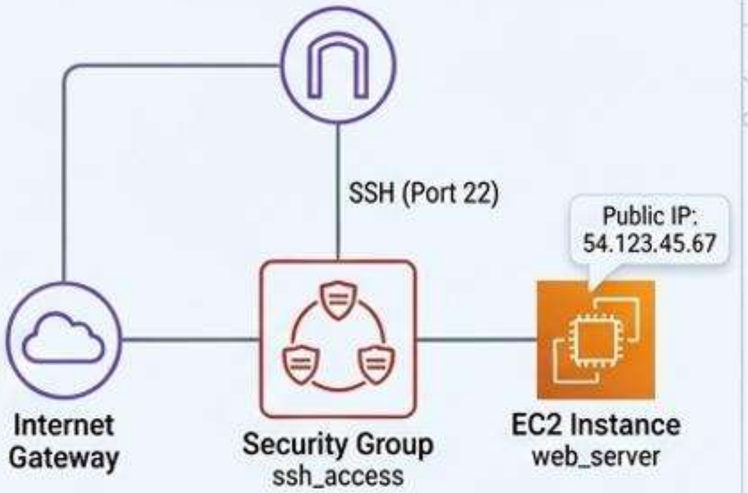
```
output "public_ip" {
  value       = aws_instance.web_server.public_ip
  description = "The public IP of the web server"
}
```

```
>_ public_ip = "54.123.45.67"
```

terraform destroy 🗑 →

One command creates all resources.
One command removes everything.

3. ARCHITECTURE & SCALABILITY



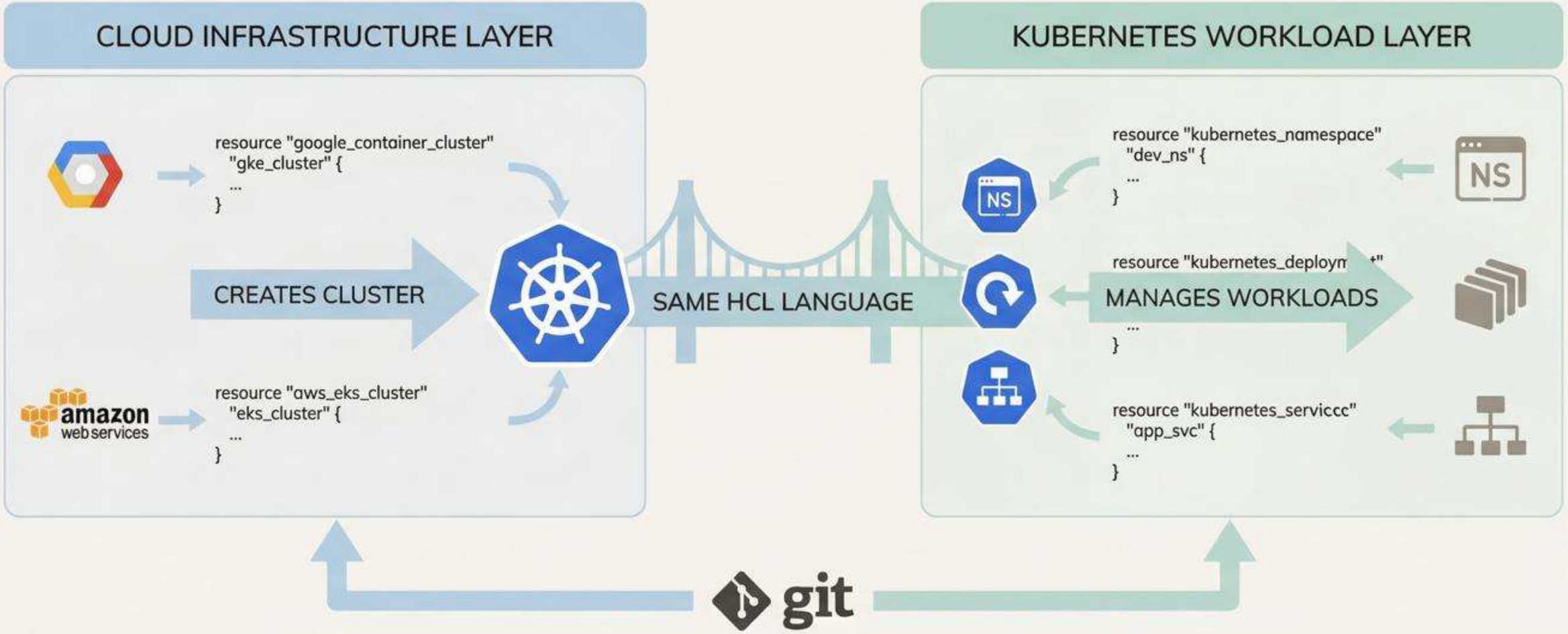
SCALABILITY & BENEFITS

- Scales to multi-tier architectures.
- Automated, reproducible deployments.
- Eliminates manual console errors.
- Version-controlled infrastructure state.

HashiCorp

TERRAFORM + KUBERNETES: UNIFIED WORKLOAD MANAGEMENT

One Tool for Cloud Clusters and Kubernetes Objects, Synchronized in Git.

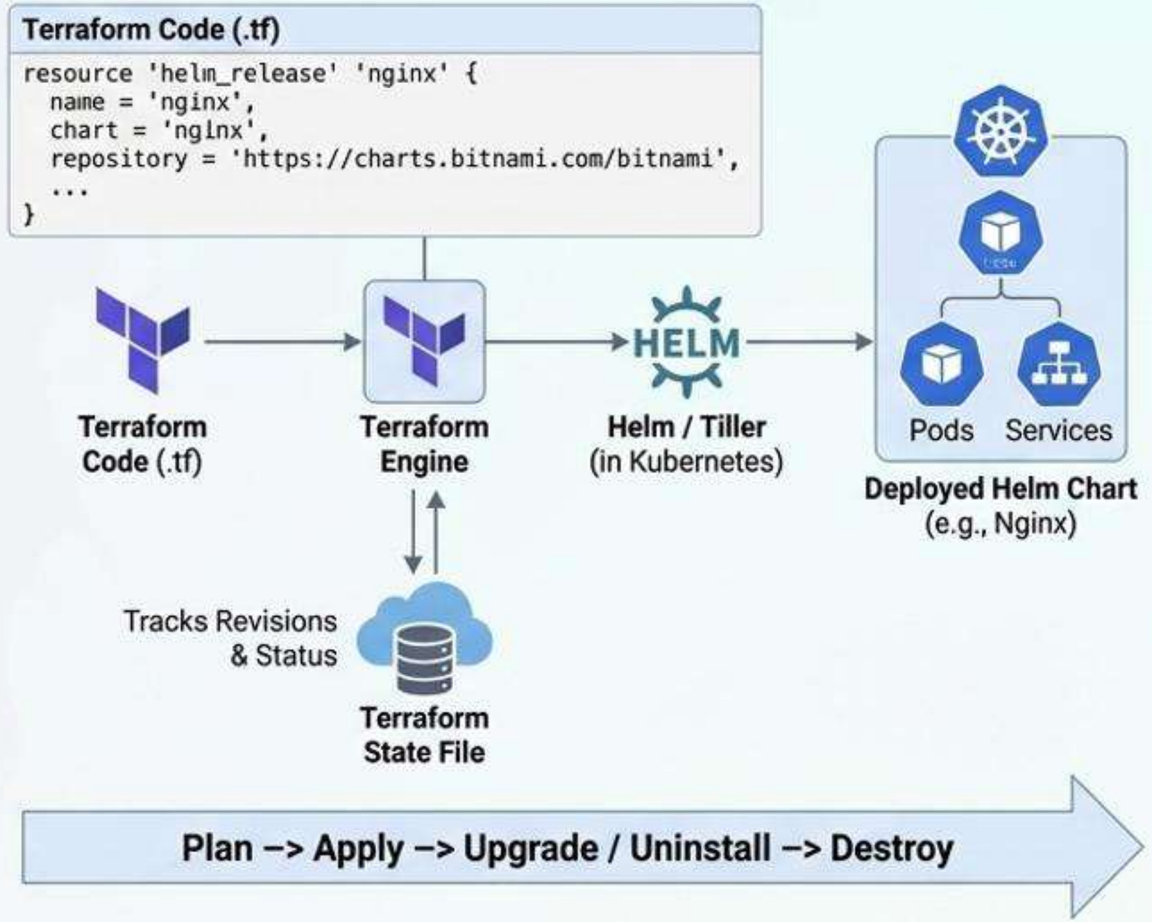


TERRAFORM + HELM: AUTOMATING KUBERNETES DEPLOYMENTS

Managing Helm Releases with Infrastructure as Code

`helm_release` Resource Overview

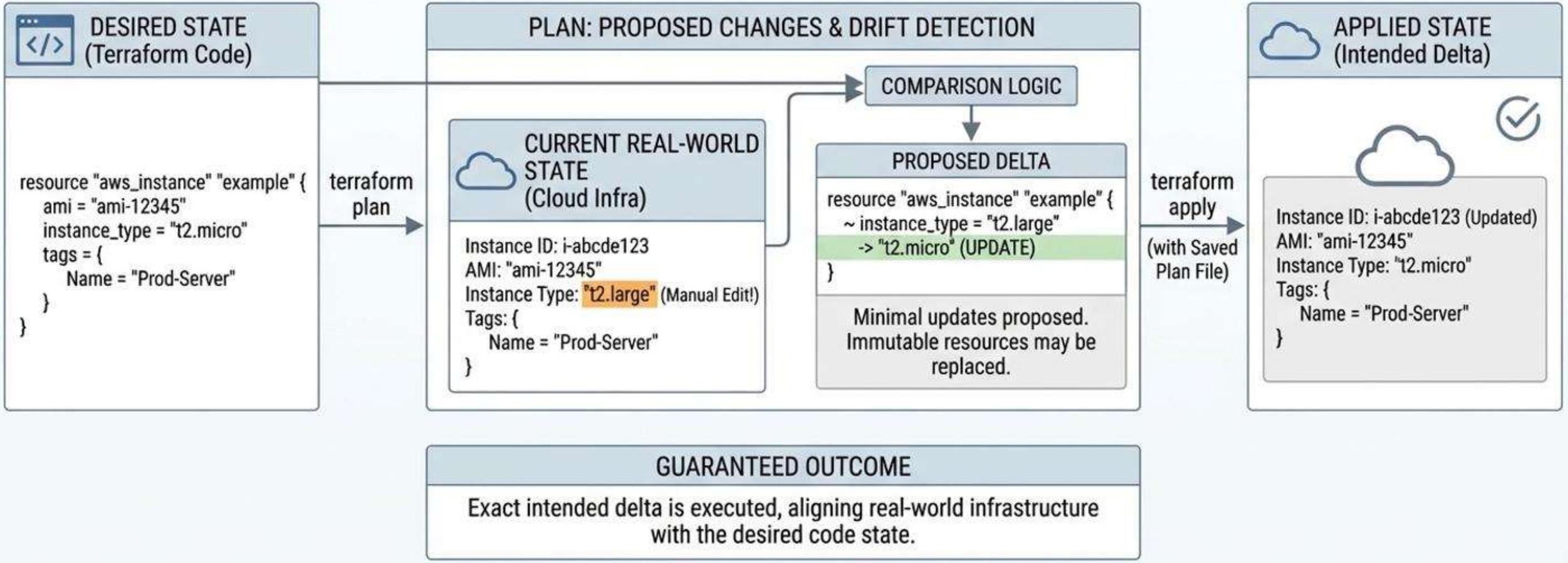
- Installs community or custom Helm charts directly from Terraform.
- Specifies **repository** URL, chart name, version, and custom values file.
- Terraform drives Helm and Tiller, removing manual “helm install” steps.
- Tracks release revisions and configuration in the state file.



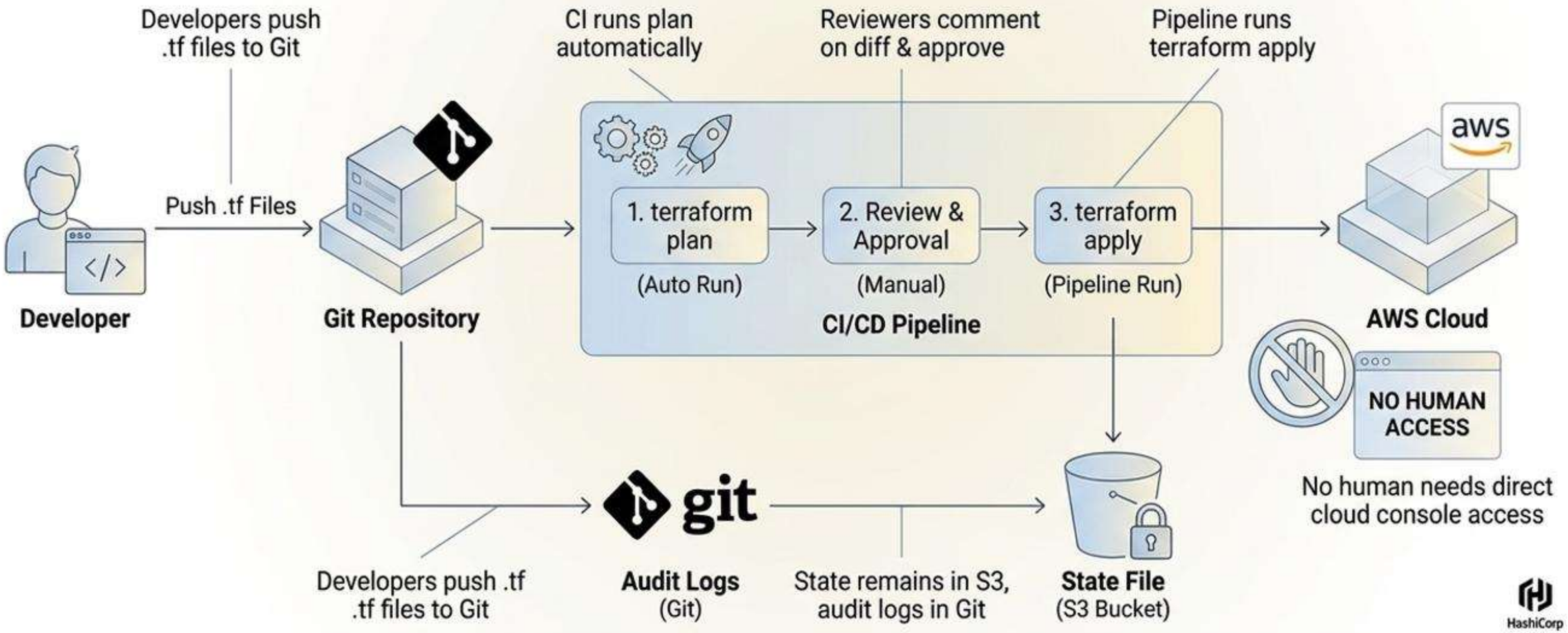
HashiCorp

CHANGE MANAGEMENT IN TERRAFORM

Detecting Drift, Proposing Updates, and Guaranteeing Intended Delta



MODERN DEVOPS TERRAFORM WORKFLOW: AUTOMATION & SECURITY



TERRAFORM INTERVIEW QUESTIONS & ANSWERS

Infrastructure as Code (IaC)

IaC: Infrastructure as Code is the practice of managing and provisioning infrastructure through code instead of manual processes. This approach enables version control, automation, and consistent, repeatable deployments across environments.

State File Purpose

State File: The state file is a JSON record that maps your configuration to real-world resources, tracking their current state. It is essential for Terraform to know what exists and determine the necessary changes during a plan or apply.

Plan vs. Apply

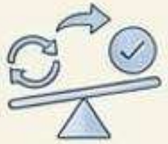
Plan vs. Apply: **`terraform plan`** is a dry-run that shows a preview of the changes Terraform will make, without modifying any resources. **`terraform apply`** executes the proposed plan, making the actual changes to your infrastructure to reach the desired state.

Provider Architecture

Provider: A provider is a plugin that translates Terraform's generic syntax into the specific API calls of a cloud platform or service. It acts as an intermediary, allowing Terraform to manage a wide variety of resources.

Remote Backend Benefits

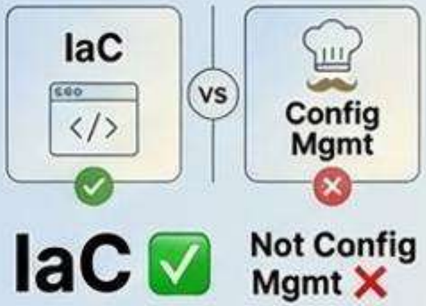
Remote Backend: A remote backend stores the state file in a shared, secure location like S3, rather than locally on a machine. This is crucial for team collaboration, providing state locking to prevent conflicts and ensuring the state is not lost.



Proving Idempotency

Idempotency: Idempotency means that applying the same configuration multiple times results in the same final state, with no additional changes if the state is already correct. Terraform achieves this by comparing the current state with the desired state before taking any action.

TERRAFORM KNOWLEDGE RECAP: FIVE KEY CONCEPTS



1. IaC vs. CONFIG MANAGEMENT

Terraform is **Infrastructure as Code**, automating infrastructure provisioning, not software configuration.

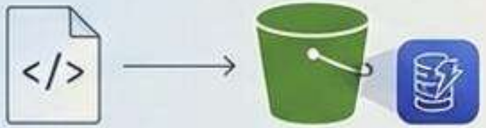
2. PREVIEW WITH 'PLAN'

Use **"terraform plan"** to preview changes before execution. **"terraform apply"** executes them.

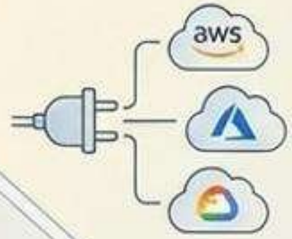
RAPID RECAP & EXAM FOCUS

3. S3 FOR STATE LOCKING & SAFETY

Use a remote backend like S3 with DynamoDB for state file storage and locking to prevent conflicts.



REMOTE STATE & LOCKING



4. PROVIDERS: CLOUD API PLUGINS

Providers are plugins that translate Terraform code into API calls for specific cloud platforms.

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

→ AWS API

→ Azure API

5. 'FMT' FOR STYLE & CONSISTENCY

Use **"terraform fmt"** to automatically format Terraform configurations to the canonical style.

Before

```
code "aaCt" {
    naxe = "us-east-1"
    nawe = "unde"
}
```

→

```
code "adult" {
  region = "us-east-1"
  update = "unde"
}
```

`terraform fmt`

After

HashiCorp

TERRAFORM – ZERO TO HERO

📌 20. Terraform Cheat Sheet

CORE COMMANDS

`terraform init`

Purpose terraform inits to mentieo & operators

`terraform validate`

Purpose validate the sheet to ainn confix:

`terraform plan`

Purpose plan penuanation for blue print plan

`terraform apply`

Purpose comorrate cloud to terraform apply

`terraform destroy`

Destroy bin for trash and terraform destroy

KEY CONCEPTS & SYNTAX

Syntax Essentials

```
variable "instance_type" {
  ...
}

output "ec2_ip" {
  ...
}
```

Provider Configuration

```
provider "aws" {
  region = "..."
}
```

Backend Configuration (S3)

```
backend "s3" {
  bucket = "..."
  ...
}
```

Resource Meta-Arguments

count – Sheot in a sennpectert on
- coxdoa porine ts fuf project

for_each – Conother tortraction requires to a specixfoum of local games

hero
ESSENTIAL DEVOPS SKILL

Print or bookmark this sheet for interview whiteboard sessions.

HashiCorp

MASTER TERRAFORM:
AUTOMATION, GOVERNANCE, & CERTIFICATION

AUTOMATION & GOVERNANCE

API-Driven Resources

Git-Based Governance

Multi-Cloud DevOps

MASTER TERRAFORM

CORE CONCEPTS & WORKFLOW

Init → Plan → Apply

Workflow

State Management

Input / Output

Variables

Modules

VALIDATE SKILLS

HashiCorp Terraform Associate Certification

Pursue Certification to Validate Your Skills & Become an Industry Standard.