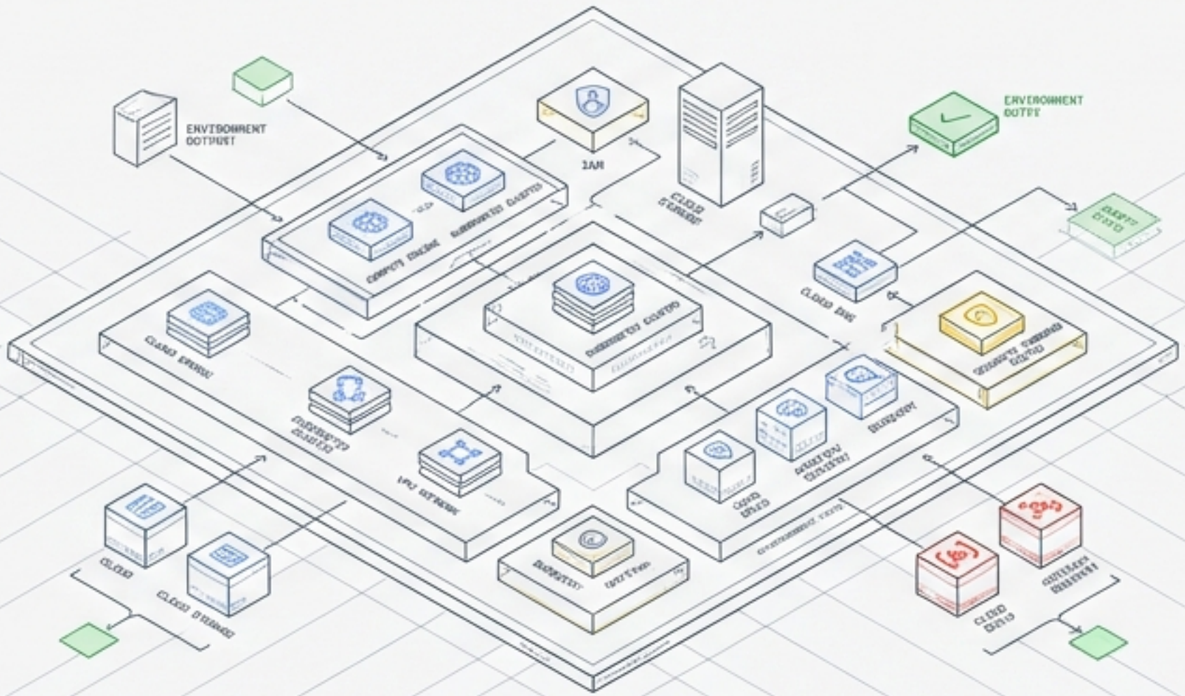
Bootstrapping the Google Cloud

pde_section_1_architecture_guide.md

A visual schematic for designing, provisioning, and securing a scalable cloud organization.

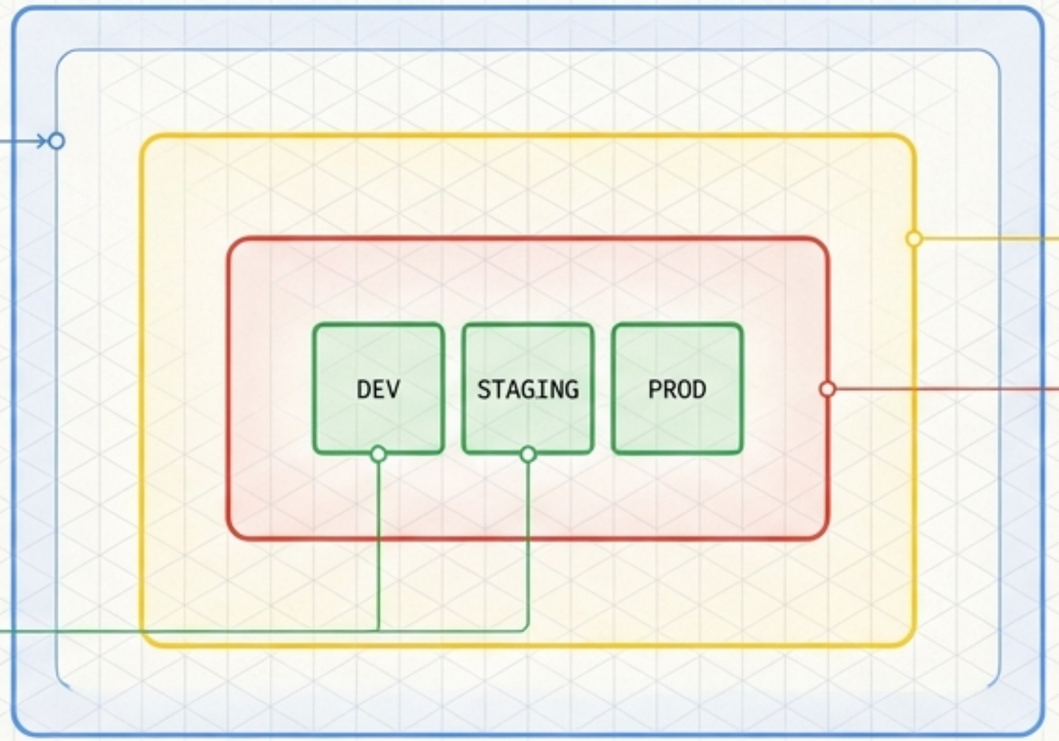


The Blueprint Architecture

Organization Hierarchy

Establishing boundaries, billing, and governance.

`gcp_organization | hierarchy_root`



Infrastructure Engine

Declarative provisioning via Terraform.

`terraform_state | policy_as_code`

Isolated Environments

Maintaining parity across the application lifecycle.

`dev_environment | staging_environment | prod_environment`

Delivery Pipeline

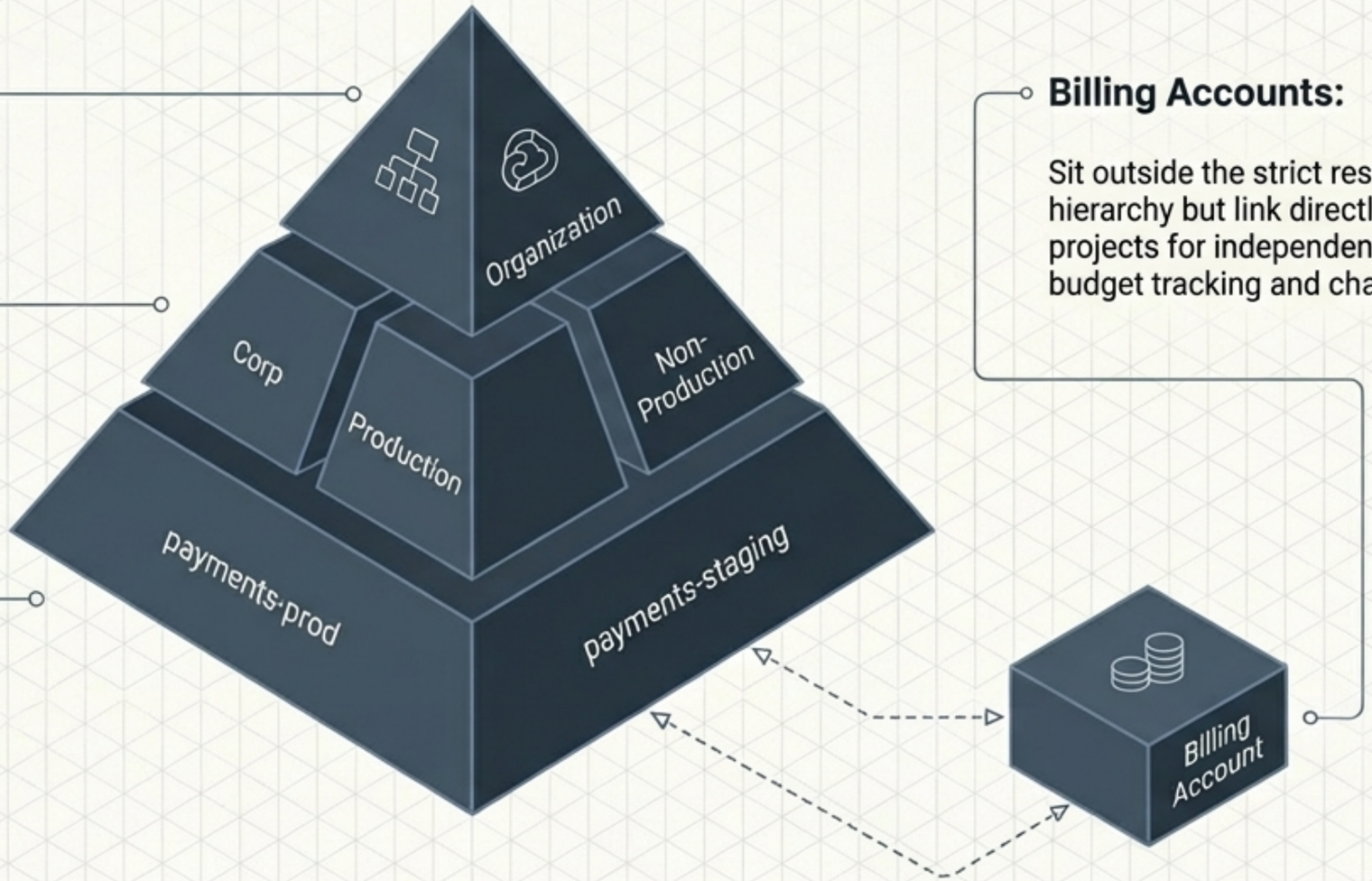
Secure, automated code-to-cloud pathways.

`ci_cd_pipeline | artifact_registry`

The Resource Hierarchy Pyramid

Folders:
Isolate environments and map to business units.

Projects:
The strict boundary where APIs are enabled, and resources are provisioned.



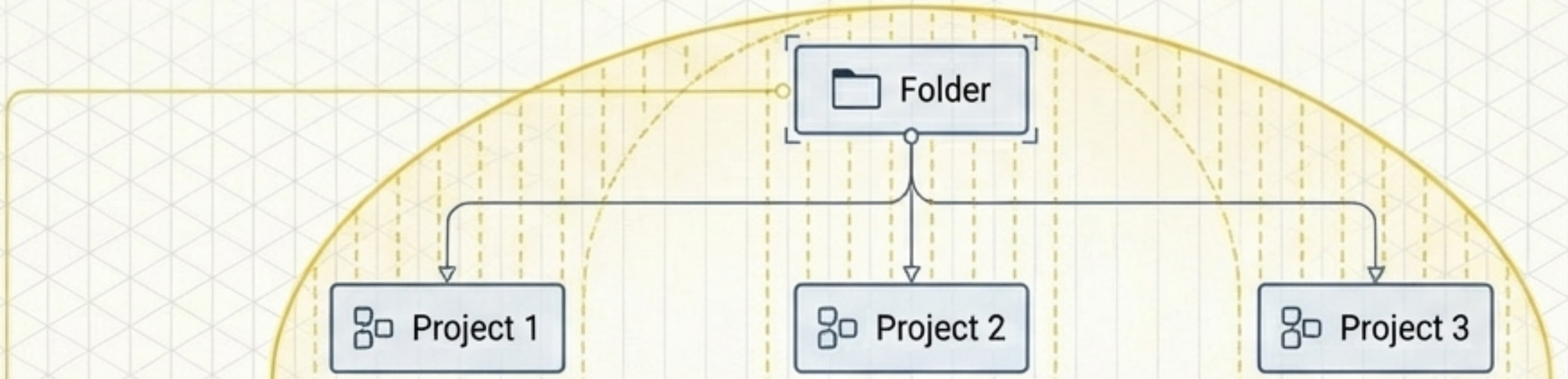
Billing Accounts:

Sit outside the strict resource hierarchy but link directly to projects for independent budget tracking and chargeback.

Flowing Policies and Governance

`google_org_policy_policy`

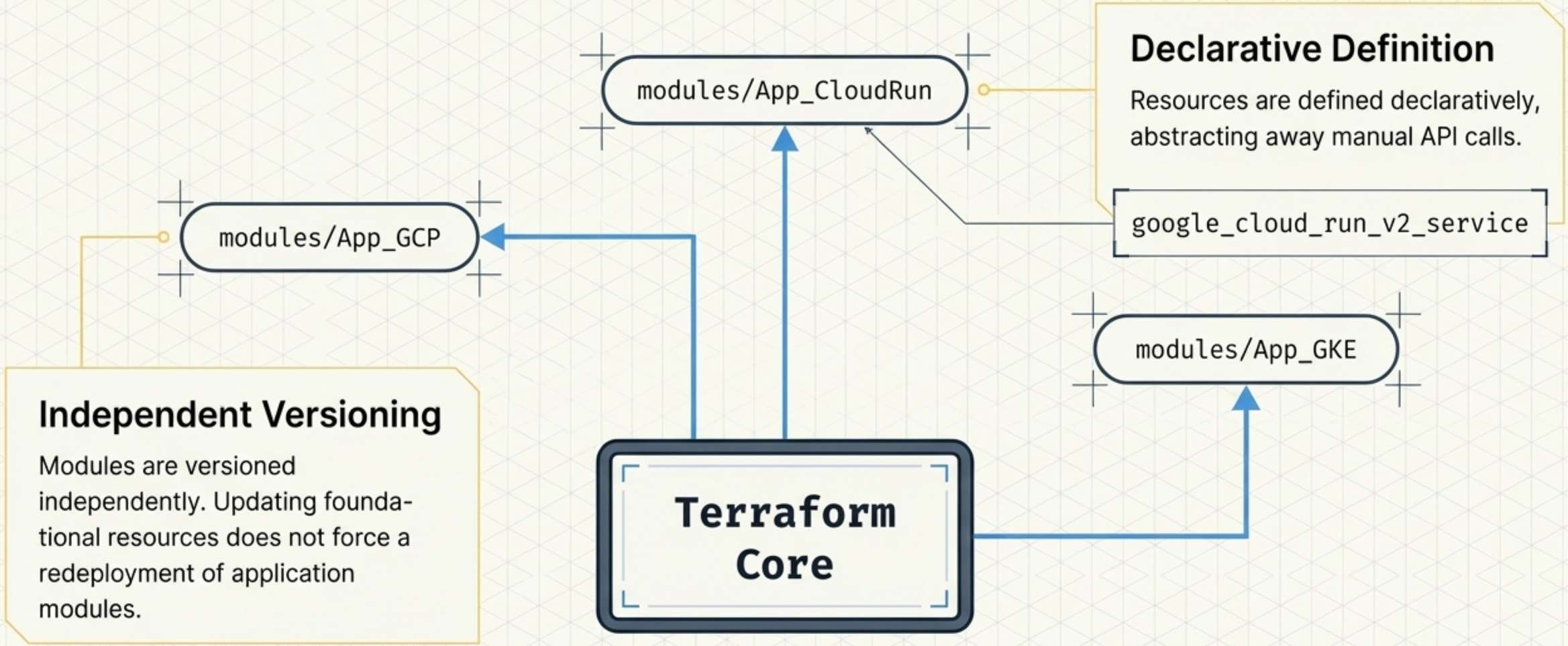
Key Insight: Policies defined at the folder level flow down automatically. Zero per-project configuration is required.



`constraints/compute.restrictCloudRunRegion` — Forces all child projects to deploy only to `us-central1`, keeping experimentation costs low.

`constraints/iam.disableServiceAccountKeyCreation` — Prevents JSON key files from being generated in any underlying production projects.

The Infrastructure Engine



State Synchronization and Drift

Declared State

main.tf

```
resource
  "google_cloud_
  run_service"
  "app" {
  ...
}
```

Remote State

Storing state in Cloud Storage prevents team conflicts and enables concurrent collaboration.

Terraform State File

Actual State

Cloud Run

Compares the Declared configuration against the Actual cloud reality. Manual changes made in the Console are instantly flagged as drift.

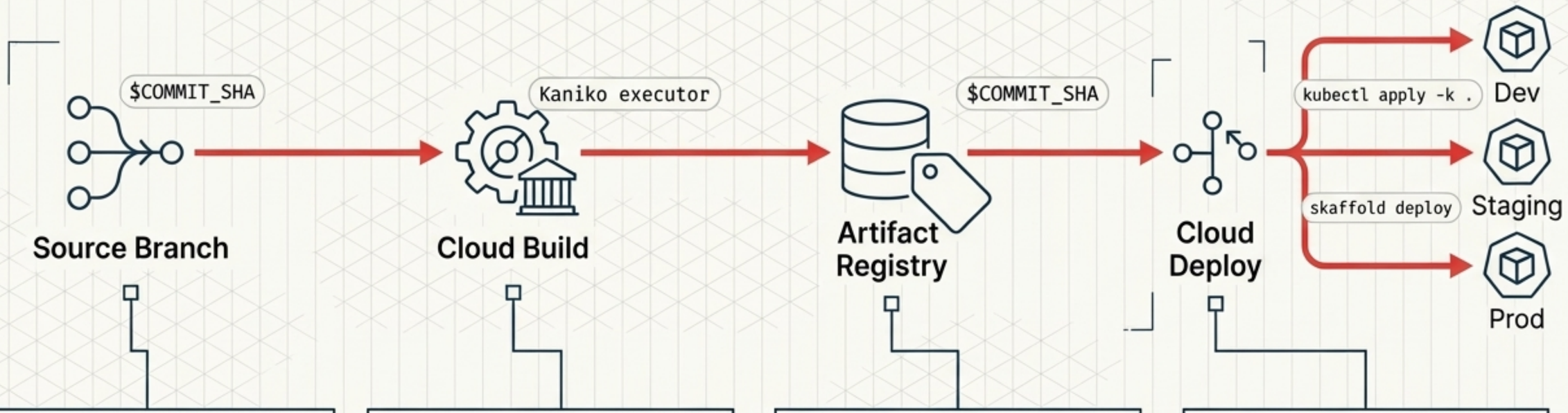
Drift Detection

Compares the Declared configuration against the Actual cloud reality. Manual changes made in the Console are instantly flagged as drift.

IaC Tool Selection Matrix

	Terraform	Config Connector
Core Philosophy	Terraform: Infrastructure-focused, cloud-agnostic platform.	Config Connector: Application-centric, Kubernetes-native workflow.
State Management	Terraform: Managed via external state files in Cloud Storage.	Config Connector: Managed natively via Kubernetes etcd.
Execution Method	Terraform: CLI and CI/CD pipelines.	Config Connector: Kubernetes manifests applied via kubectl.
Best Used When	Terraform: Bootstrapping organizational landing zones and broad GCP services.	Config Connector: Teams deeply embedded in a GKE-centric workflow managing GCP resources alongside pods.

The Code-to-Cloud Assembly Line



Source

The pipeline originates from a single, immutable Git commit SHA, establishing the foundation for absolute traceability throughout the assembly line.

Cloud Build

Uses a daemonless, rootless `Kaniko executor` for secure image building without Docker daemon privileges, ensuring a hardened and secure environment.

Artifact Registry

Images are pushed with an immutable tag derived strictly from the Git commit SHA, ensuring absolute traceability and preventing version conflicts.

Cloud Deploy

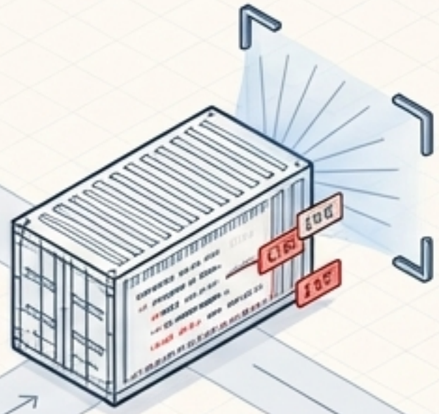
`Skaffold` manages the rendering of Kubernetes manifests and progressive promotion across target stages, from Dev through Staging to Production.

The Immutability Passport

SLSA Provenance

Cloud Build generates SLSA Level 3 provenance—a cryptographically signed attestation of what was built, from what source, and by what build system.

Artifact Analysis



Automatically scans images for known CVEs. Builds fail instantly if CRITICAL vulnerabilities are found.

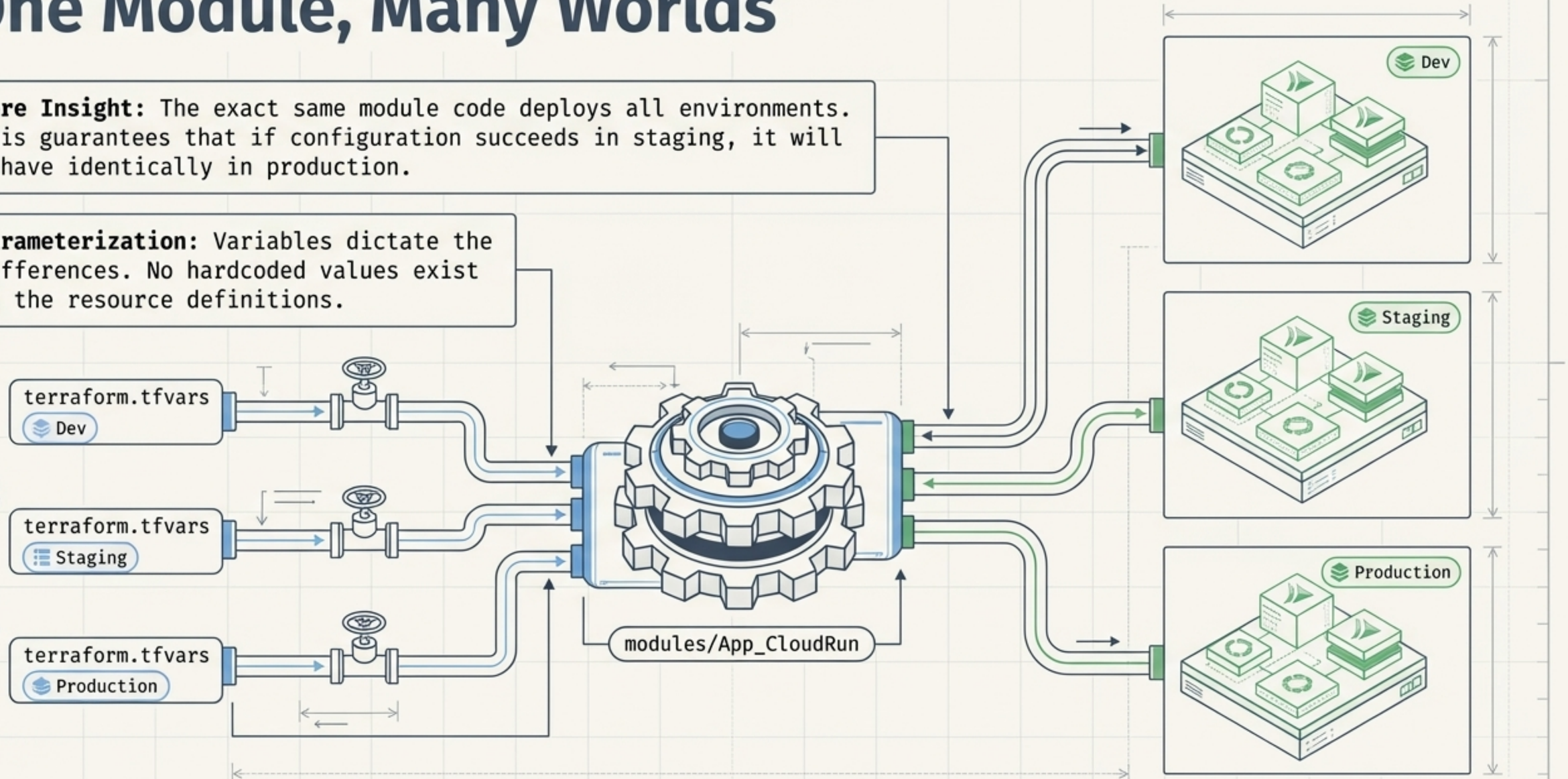
Binary Authorization

The ultimate deployment gate. Enforces that only attested, policy-compliant images can reach production, blocking arbitrary manual pushes.

One Module, Many Worlds

Core Insight: The exact same module code deploys all environments. This guarantees that if configuration succeeds in staging, it will behave identically in production.

Parameterization: Variables dictate the differences. No hardcoded values exist in the resource definitions.



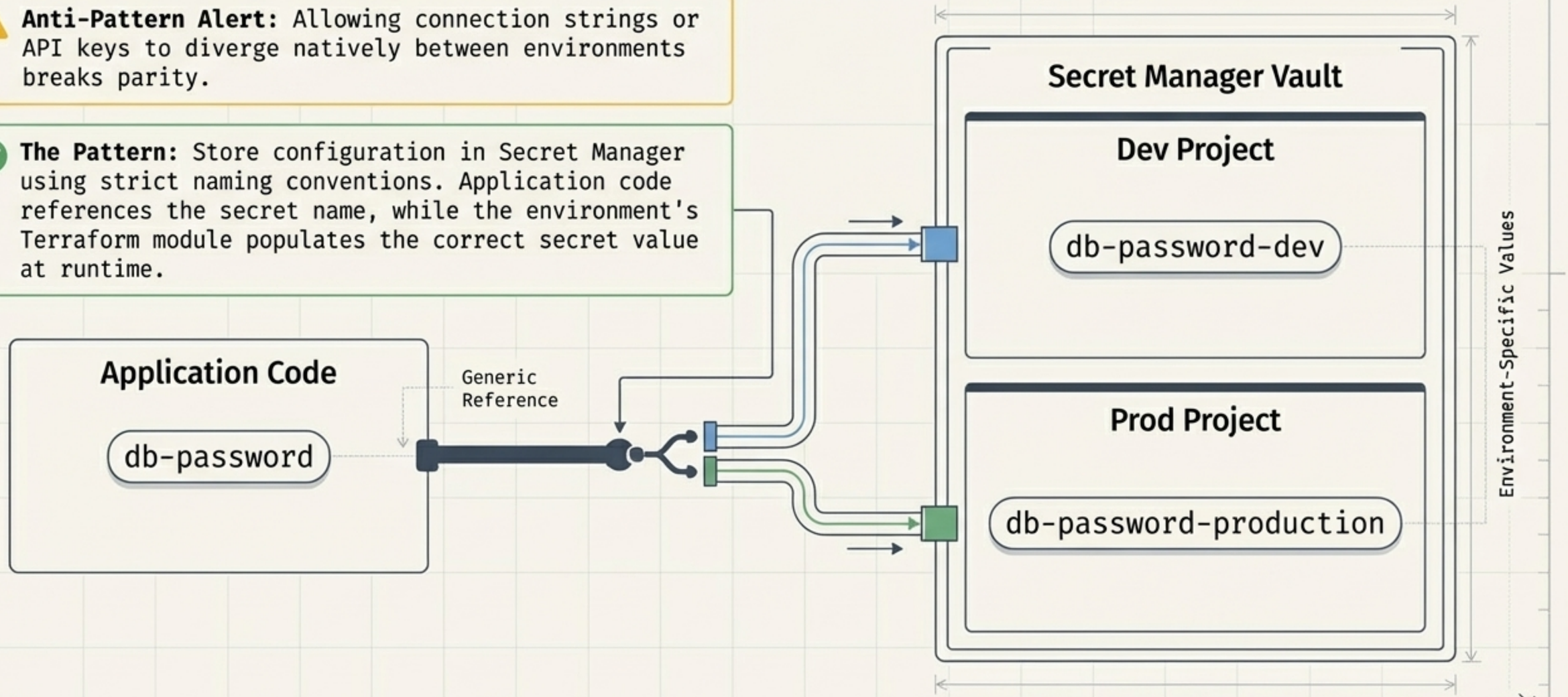
Tuning the Parity Matrix

Variable	Dev Environment	Staging Environment	Production Environment
`min_instance_count`	`0` (Scale to zero, idle cost savings)	`1` (Standby)	`3` (Always-warm, no cold starts)
IAM Developer Access	`roles/editor`	`roles/viewer`	`roles/viewer` (Read-only observation)
Promotion Trigger	`Automatic on commit`	`Manual Console Approval`	`Dual-approval required`
Feature Flags	`True`	`True`	`False` (Gated for A/B rollout)

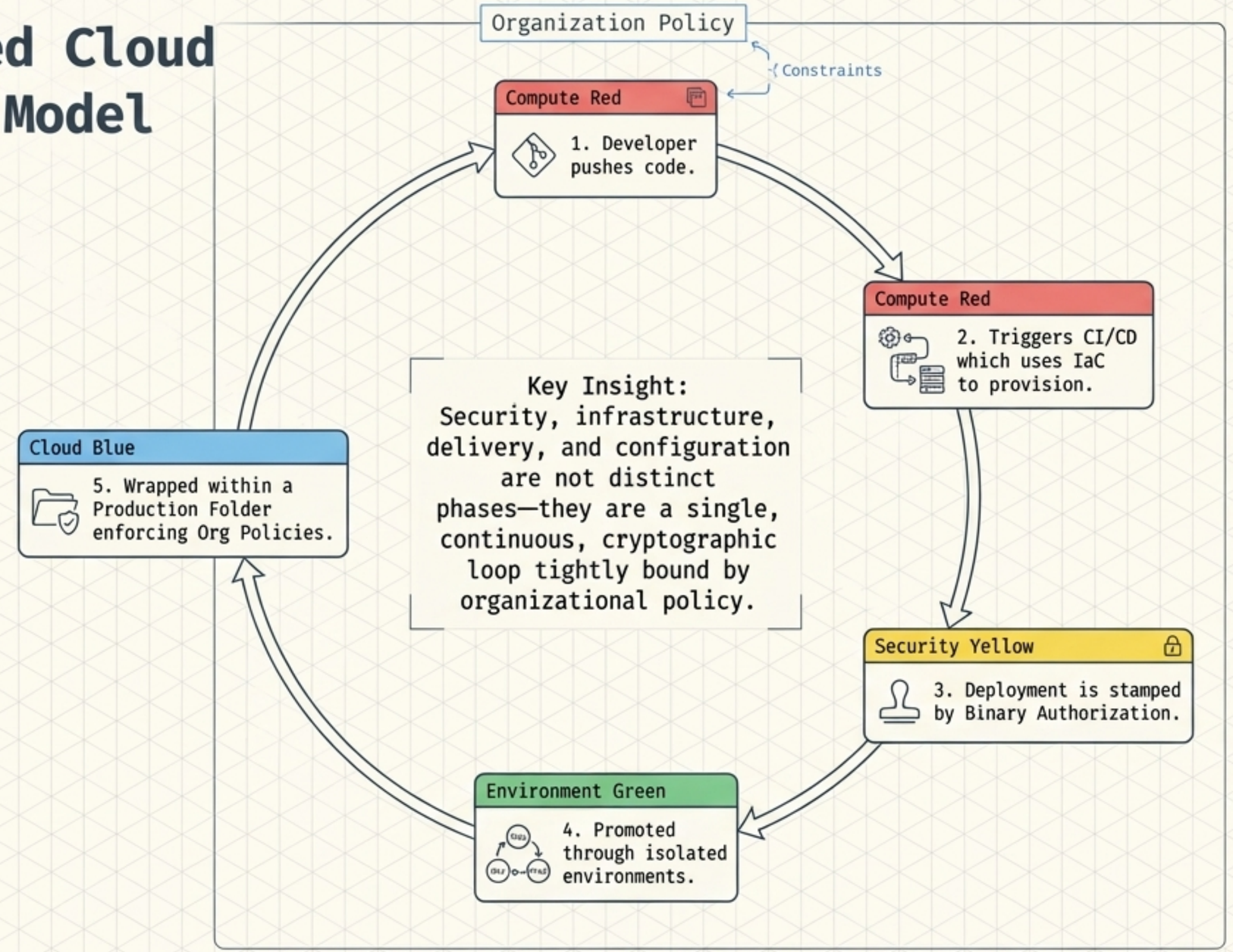
Promoting Configuration Securely

Anti-Pattern Alert: Allowing connection strings or API keys to diverge natively between environments breaks parity.

The Pattern: Store configuration in Secret Manager using strict naming conventions. Application code references the secret name, while the environment's Terraform module populates the correct secret value at runtime.



The Unified Cloud Operating Model



Mastery Beyond the Tools

“The Professional Cloud DevOps Engineer does not simply write Terraform or configure pipelines. They architect a synchronized ecosystem where infrastructure is immutable, security is cryptographically enforced, and governance flows automatically from code to cloud.”

End of Section_1_Exploration // Execute plan to continue.