Scope & Audience
This guideline is intended for developers, AI engineers, DevOps engineers, architects, and AI coding assistants working on GCP-based projects. It prescribes Infrastructure as Code (IaC) and Continuous Integration / Continuous Deployment (CI/CD) best practices to enable code components to autonomously provision their required cloud infrastructure.

---

1. Layered Architecture

Define clear separation of concerns across five layers. Each layer consumes services from the layer beneath and exposes well-documented interfaces to layers above.

1.1 Pipeline Layer

- **Data / Knowledge Base Pipeline**: Automated ingestion, transformation, and versioning of data.
- **Code / Deployment Pipeline**: IaC templates (Terraform, Deployment Manager) stored in Git, triggered by commits.
- **Cloud Operations Pipeline**: Monitoring, logging, alerting, and automated rollbacks via Cloud Build and Cloud Functions.

1.2 Tools Layer

- **Internal Tools**: Custom-made, GCP tenant-scoped service catalog entries.
- **External Tools**: Third-party APIs/MCPs, managed connectors.

1.3 Models Layer

- Packaging and versioning of AI/ML models.
- Artifact storage in Container Registry / Artifact Registry.

1.4 Agents Layer

- Deployment of agent runtimes (Cloud Run, Cloud Functions).
- Definition of agent permissions via IAM roles and service accounts.

1.5 Workflows Layer

- Orchestration via Workflows, Composer, or Cloud Tasks.
- YAML-driven workflow definitions stored alongside code.

---

2. Simplified Environment Transitions

Minimize friction when moving code and IaC between dev, test, and pre-prod.

1. **Parameterized Configuration**
    - Store environment-specific values (project IDs, region, service account emails) in Terraform variables or Cloud Build substitution maps.
    - Use separate tfvars files: dev.tfvars, test.tfvars, preprod.tfvars.
2. **Service Accounts & API Keys**
    - Define one service account per environment; rotate keys via Secret Manager.
    - Automate Secret Manager replication across environments using Terraform.
    - Reference secrets in pipelines with gcloud secrets versions access latest in Cloud Build.
3. **GitOps Branching Strategy**
    - main → pre-prod
    - develop → test
    - feature/* → dev
    - Merge rules trigger automated environments via Cloud Build triggers.
4. **Shared Modules & Blueprints**
    - Publish reusable Terraform modules in a private Artifact Registry.
    - Enforce module versioning to ensure stability across environments.

---

3. IaC & CI/CD Principles

Enable code-driven infrastructure provisioning with automated quality gates.

1. **Declarative IaC**
    - Use Terraform (recommended) or Deployment Manager for all infra.
    - Enforce style and policy with tflint, terraform fmt, and Conftest (OPA).
2. **Pipeline Automation**

    - Cloud Build pipelines defined in cloudbuild.yaml.
    - Steps: terraform init → terraform plan → policy checks → manual approval (test/preprod) → terraform apply.
3. **Immutable Infrastructure**
    - Avoid in-place resource mutation; use resource replacement and canary deployments.
    - For compute workloads, use container images with versioned tags.
4. **Automated Testing**
    - Unit tests for Terraform modules via terraform-compliance.
    - Integration tests using Test Kitchen or Molecule against deployed stacks.
5. **Rollbacks and Roll-forwards**
    - Leverage native Terraform state and versioning.
    - Cloud Build rollback on failed smoke tests; alerting via Cloud Monitoring.

---

4. Maintainability & Continuous Improvement

Focus on simplicity, reusability, and evolutionary enhancements.

- **Minimal Deployments**: Combine related resources in modules to reduce deployment time.
- **Layer Reusability**: Publish shared modules for networking, IAM, storage, and AI services.
- **Strategic Use of Layers**: Keep pipelines, tools, models, agents, and workflows decoupled; evolve each layer independently.
- **Documentation as Code**: Maintain pipeline, module, and API docs in Markdown alongside code.
- **Versioning & Changelog**: Tag every release; publish changelogs automatically via CI/CD pipelines.

---

5. Compliance, Budget Management & Scalability

Ensure governance, cost control, and the ability to serve up to 10,000 users.

5.1 Compliance Controls

- Enforce organization policies with Org Policy Service (e.g., allowed regions, machine types).
- Enable Audit Logging on all projects.
- Periodic configuration scans via Forseti / Config Connector.

5.2 Budget Management

- Define budgets and alerts in Billing Budget API.
- Set cost ceilings per environment; automate notifications via Cloud Functions.
- Leverage Quota Management to cap API usage.

5.3 Scalability

- Compute: Use Cloud Run or GKE with Horizontal Pod Autoscaler.
- Storage: Choose multi-regional Cloud Storage with lifecycle rules.
- Database: Use managed services (Cloud SQL with read replicas, Bigtable for high throughput).
- Networking: Auto-scale load balancers with global backend services.

---

6. Actionable Recommendations & Best Practices

- **Adopt GitOps**: Treat all infra changes as pull requests with peer reviews.
- **Enforce Policy as Code**: Embed security/compliance rules in CI using OPA/Gatekeeper.
- **Automate Secrets Management**: Rotate service credentials automatically; never store creds in code.
- **Use Feature Toggles**: Deploy infra early; gate new services via runtime flags.
- **Regularly Review Costs**: Schedule automated budget reports; optimize idle resources.
- **Monitor & Alert**: Define SLOs/SLIs; set up dashboards and alerts for key metrics.
- **Perform Chaos Testing**: Use Fault Injection frameworks to validate resilience.
- **Continuous Learning**: Hold quarterly IaC and pipeline reviews; adopt community modules and patterns.


Workflow Builder Layer
Non-Technical Users
Technical Users
Models Layer
AI Assistance
AI Agents
implementation Frameworks
Tools Layer
Internal Tools: Custom-made, GCP tenant-scoped service catalog entries.
External Tools: Third-party APIs/MCPs, managed connectors.
Data / Knowledge Base
Data Layer
Prompts
Pipeline Layer
Data / Knowledge Base Pipeline: Automated ingestion, transformation, and versioning of data.
Code / Deployment Pipeline: IaC templates (Terraform, Deployment Manager) stored in Git, triggered by commits.
Cloud Operations Pipeline: Monitoring, logging, alerting and automated rollbacks via Cloud Build and Cloud Functions.