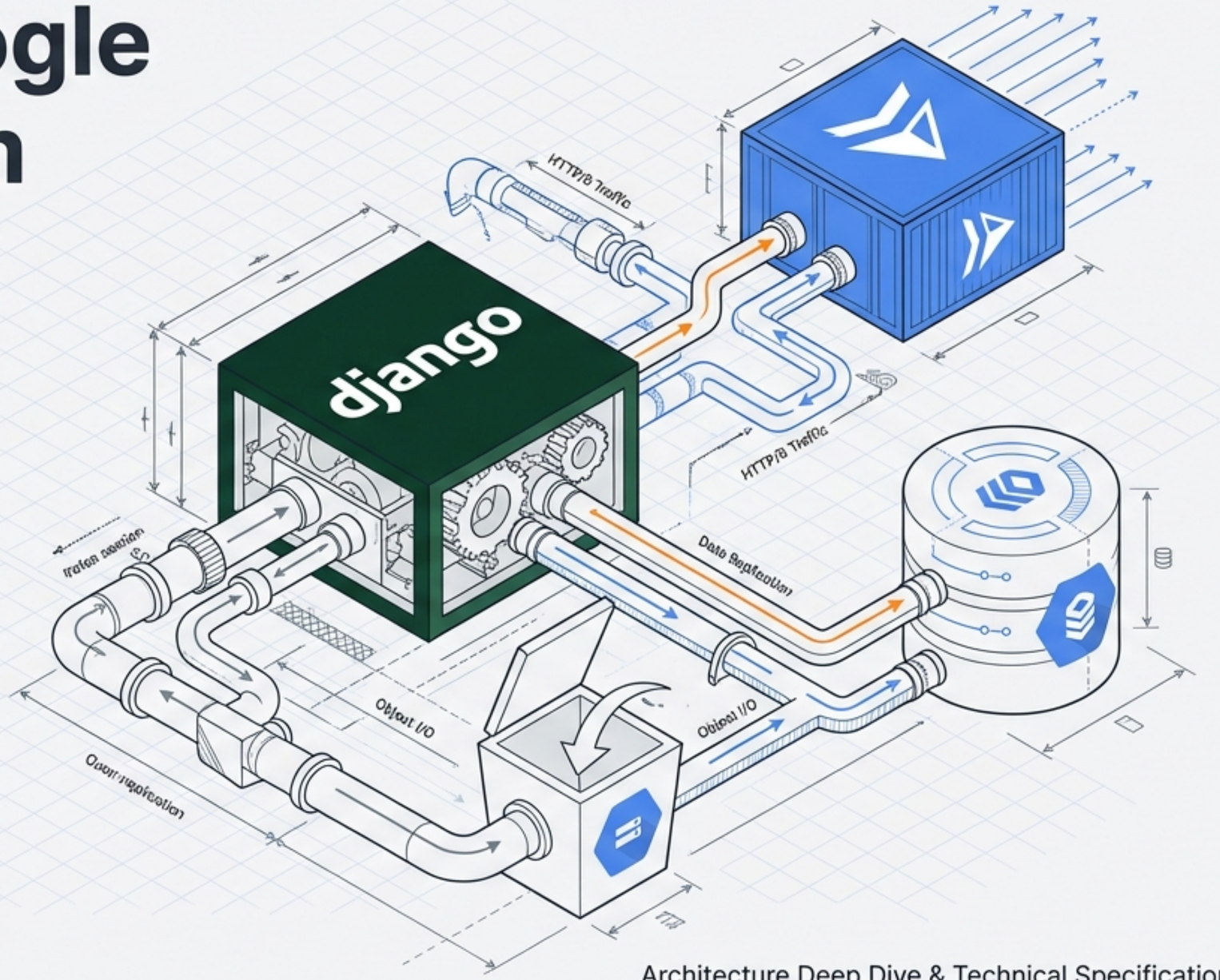
Django on Google Cloud Platform

The RAD Platform Architecture Pattern



MODULE:	modules/Django
VERSION:	1.0.0
DOC_TYPE:	ARCHITECTURE_SPEC

Architecture Deep Dive & Technical Specification

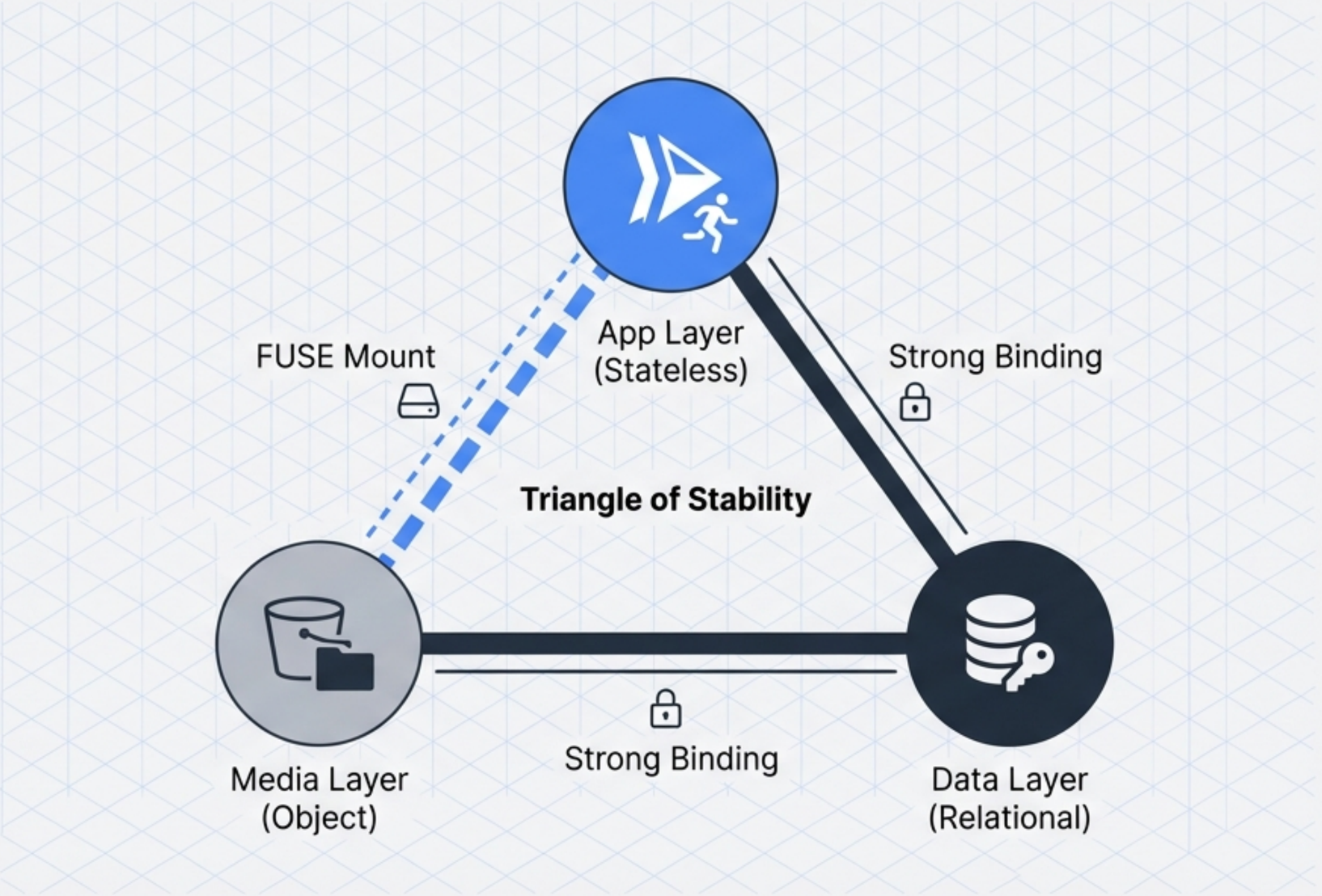
Stateless Compute Meets Managed Persistence

The Philosophy

The ‘modules/Django’ wrapper transforms the standard Django deployment into a production-ready, serverless workload. By strictly application logic from data and file storage, the system achieves true horizontal scalability.

Core Strategy

The application server can scale to zero to minimize costs or burst to max capacity during traffic spikes without any risk of data loss or state inconsistency.



The Application Engine: Cloud Run Configuration

Build Specifications

Base Image: `python:3.11-slim`
Source: `modules/Django/scripts/django/Dockerfile`
Naming Convention: `${tenant_deployment_id}-${app_name}`

Resource Allocation

CPU Allocation: 1 **vCPU** (Default)
Memory Allocation: **1Gi** (Default)

Auto-Scaling Logic

Min Instances: **0** (Cost Optimization)
Max Instances: **3** (Cost Control)

Health & Probes

Startup Probe: **GET /health/** (90s initial delay)
Liveness Probe: **GET /health/** (Continuous check)
Request Timeout: 300s (5 minutes)

Persistence Layer: Cloud SQL & PostgreSQL 15

Connection Mechanics

- Mount Point: `/cloudsql`
- Injection: django.tf injects DB_HOST
- Backend: `django.db.backends.postgresql`

Required Extensions

- pg_trgm (Trigram)
- unaccent (Text Search)
- hstore (Key/Value)
- citext (Case-insensitive)

Automation

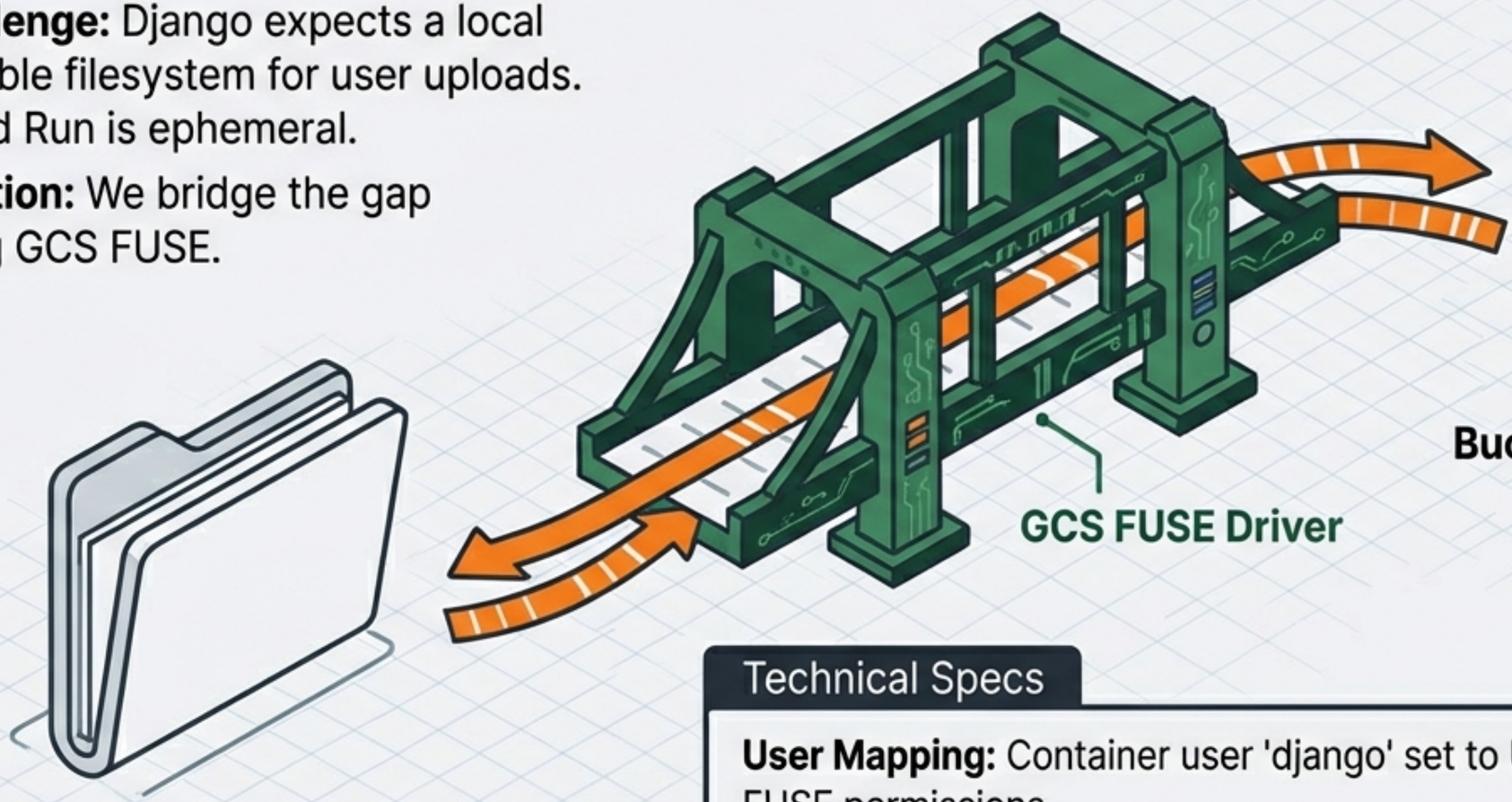
The “db-init” Job

- A specialized Cloud Run Job executing on Terraform apply.
- **Workflow:** Check DB Exists ➔ Create Role ➔ Create DB ➔ Grant Privileges.

Solving the Media Problem: Cloud Storage & FUSE

Challenge: Django expects a local writable filesystem for user uploads. Cloud Run is ephemeral.

Solution: We bridge the gap using GCS FUSE.



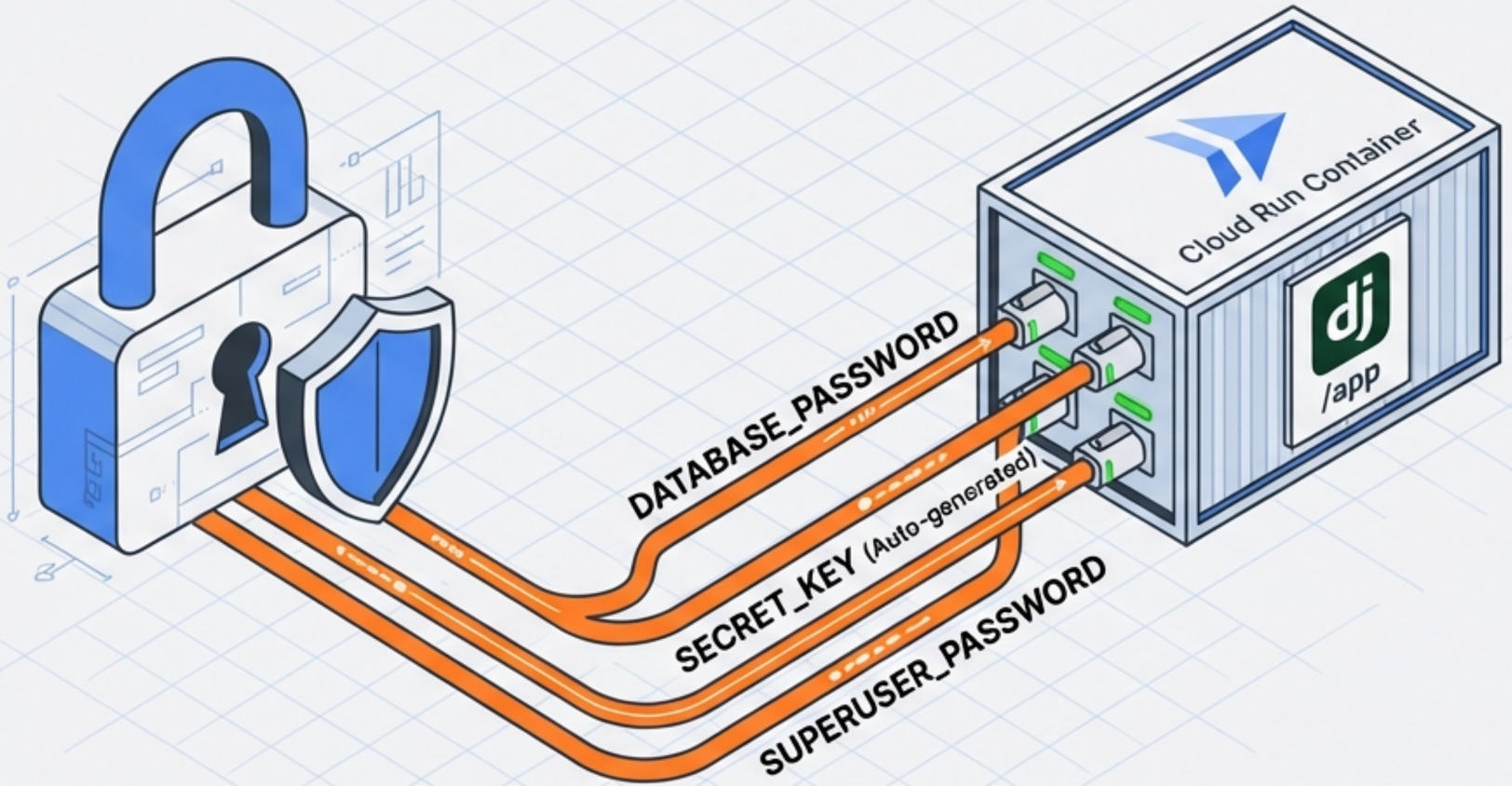
Bucket: *-django-media

Local Path: /app/media

Technical Specs

User Mapping: Container user 'django' set to UID 2000 to match GCS FUSE permissions.
Mount Flags: implicit-dirs (Simulates directory structure in object store).
Volume Mount: bucket mounted directly to /app/media.

Secrets Management & Injection



No Hardcoding

Secrets are never stored in the repo. They are referenced by the Service Definition and injected as environment variables at runtime.

Auto-Generation

The Django Secret Key is automatically created via the 'random_password' Terraform resource during the initial infrastructure build.

Identity & Access Management (IAM) Strategy

Runtime Identity (Cloud Run SA)

The identity of the active application.

`roles/secretmanager.secretAccessor`
(Read Keys)

`roles/storage.objectAdmin`
(Read/Write Media)

`roles/storage.legacyBucketReader`
(FUSE Metadata)

Build Identity (Cloud Build SA)

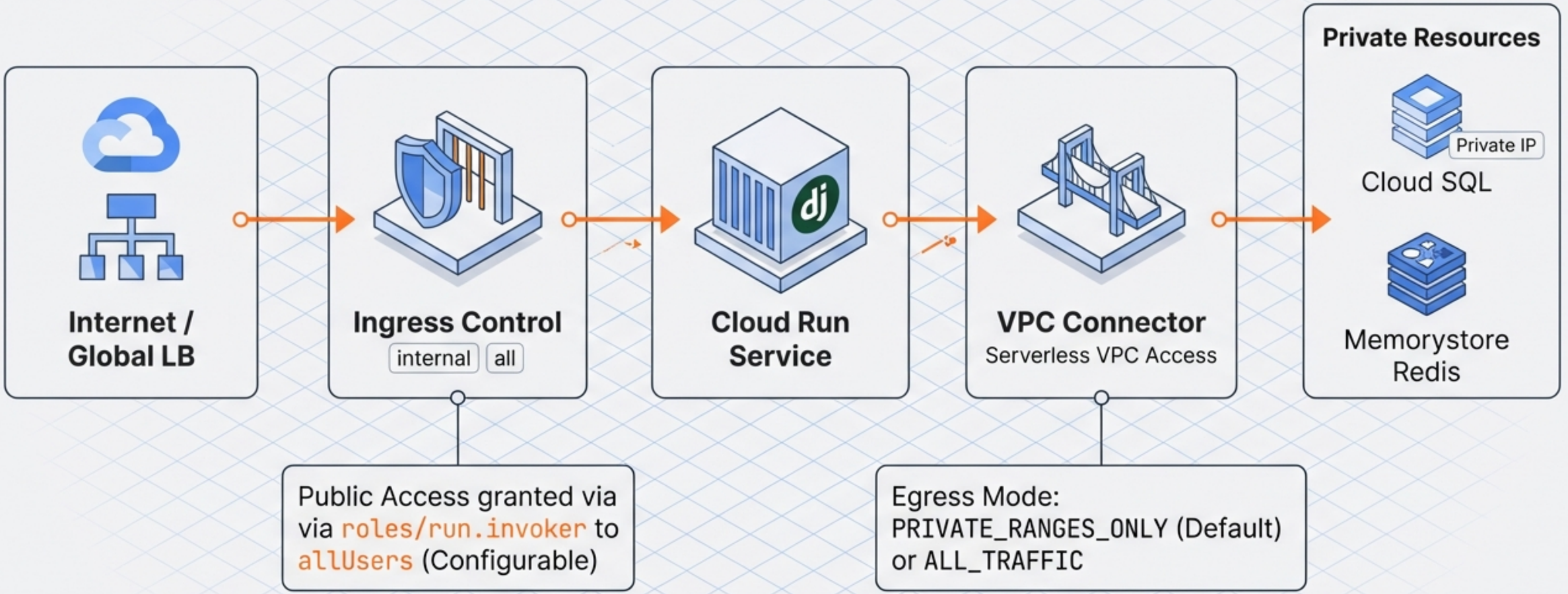
The identity performing the deployment.

`roles/run.developer`
(Deploy Revisions)

`roles/iam.serviceAccountUser`
(Act as Runtime SA)

Principle of Least Privilege enforced at all layers.

Networking, Ingress, and VPC Connectivity



Automated Operations & Deployment Lifecycle

Build & Init

Cloud Build triggers. "db-init" job verifies DB existence & permissions.

Deploy

Cloud Run Service updated with new container image.

Post-Deploy Automation

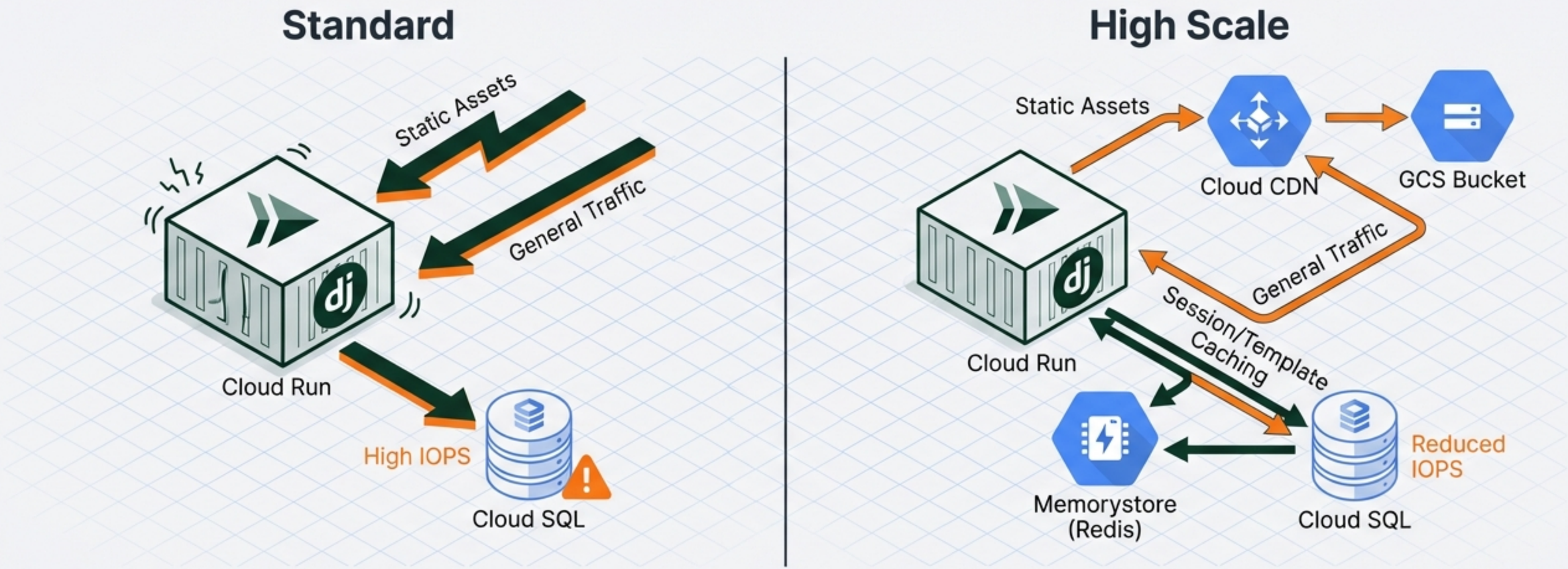
Migration Job: `python manage.py migrate`

Static Files: `collectstatic`

Superuser: `entrypoint.sh` checks & creates admin.

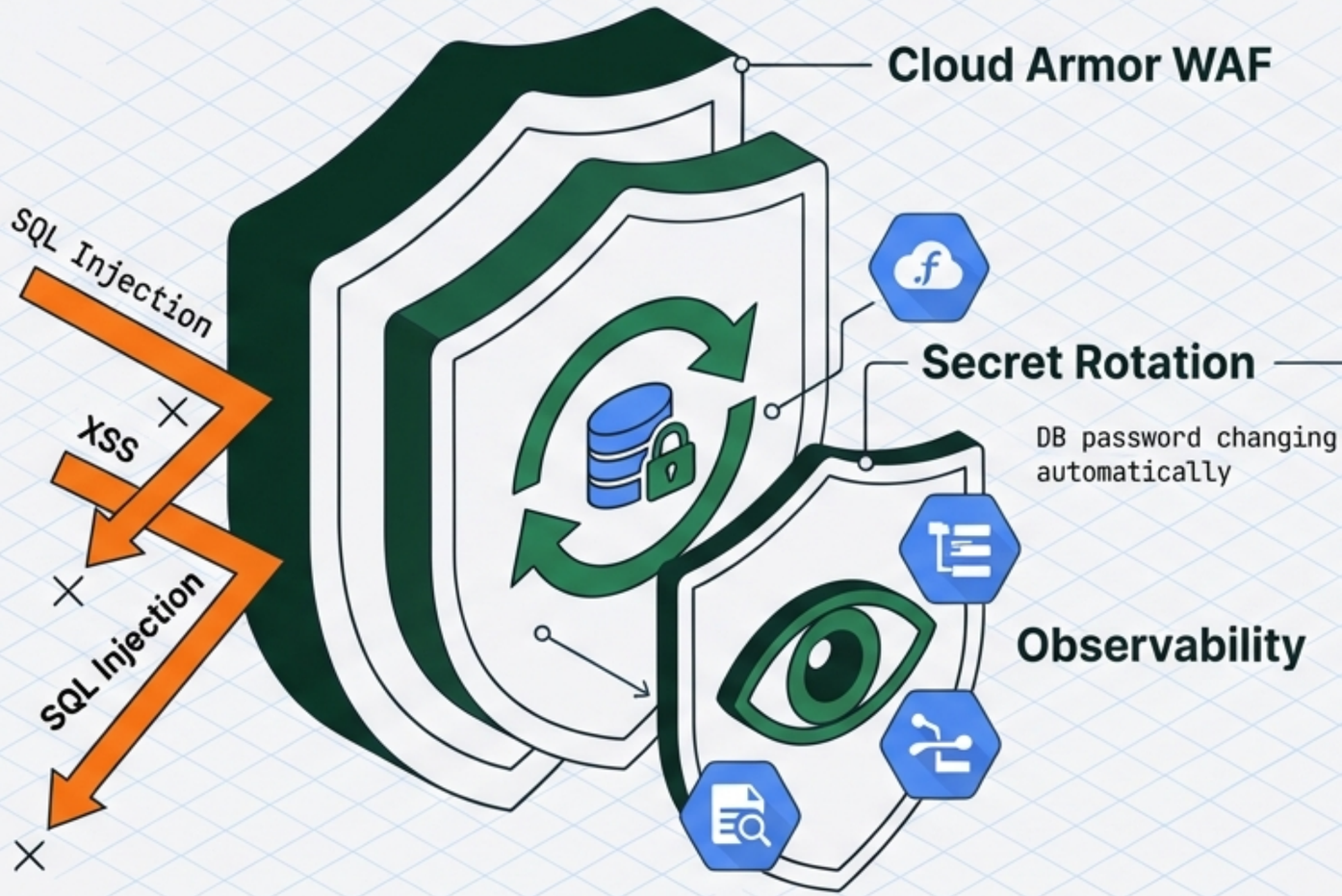
Runtime Hardening: ⚠ Non-root execution (UID 2000), ⚠ DEBUG=False, ⚠ Configurable ALLOWED_HOSTS

Architecture Roadmap: Performance & Caching



- **CDN Integration:** Offload static file serving to Cloud CDN using `'django-storages'`. Reduces container CPU/Network load.
- **Caching Strategy:** Implement `Memorystore (Redis)` for session/template caching to reduce Cloud SQL read IOPS.

Architecture Roadmap: Security & Observability

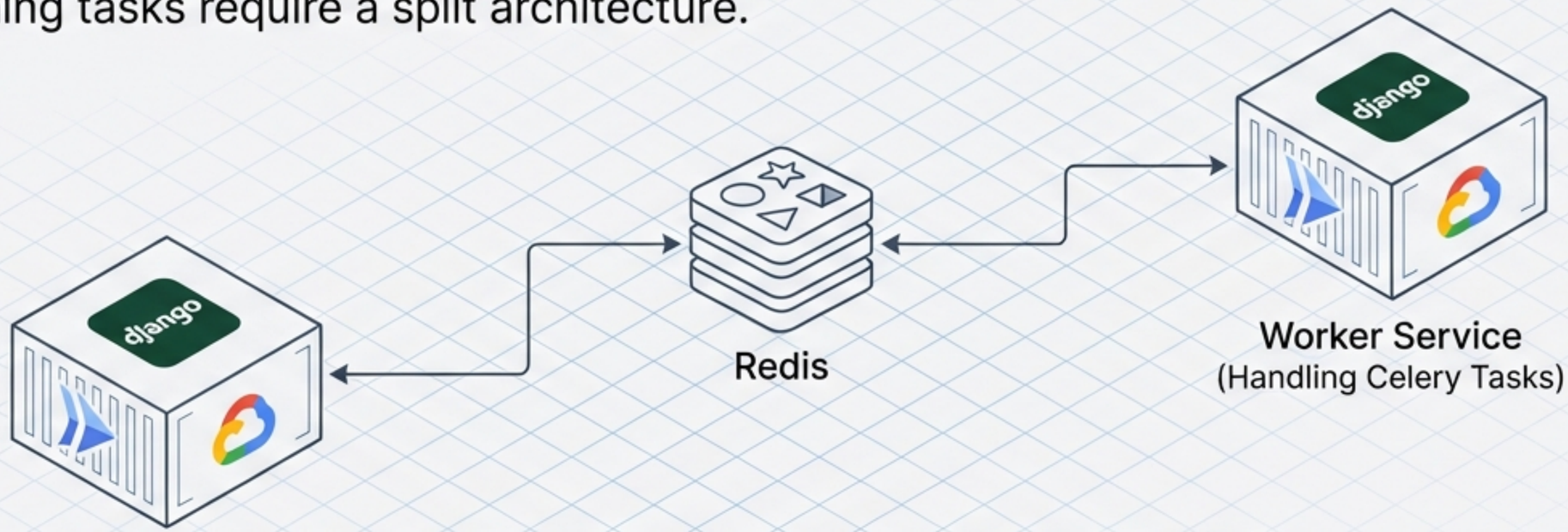


Security & Observability Details

- Security: **Strict Host Checking**: Dynamic discovery of Cloud Run URLs.
- Logging: **Structured Logging**: JSON output via '`structlog`' for severity filtering.
- Tracing: **OpenTelemetry**: Full request tracing across Cloud Run and Cloud SQL.

Architecture Roadmap: Async Workers & Scheduling

Cloud Run is optimized for request/response.
Long-running tasks require a split architecture.



Celery Workers

Deployment of a secondary service (or sidecar) triggered by “enable_worker = true”.

Cloud Scheduler

Replacement of internal Cron with managed Cloud Scheduler jobs to trigger management commands.

Summary: A Production-Ready Standard

SECURE

Identity-based access (IAM), Secret Manager integration, and non-root execution.

SCALABLE

Zero-to-Max auto-scaling with Cloud Run; managed persistence with Cloud SQL.

AUTOMATED

'Batteries-included' operations with db-init, auto-migrations, and superuser handling.

The RAD Platform Django module bridges the gap between complex infrastructure requirements and developer ease-of-use.