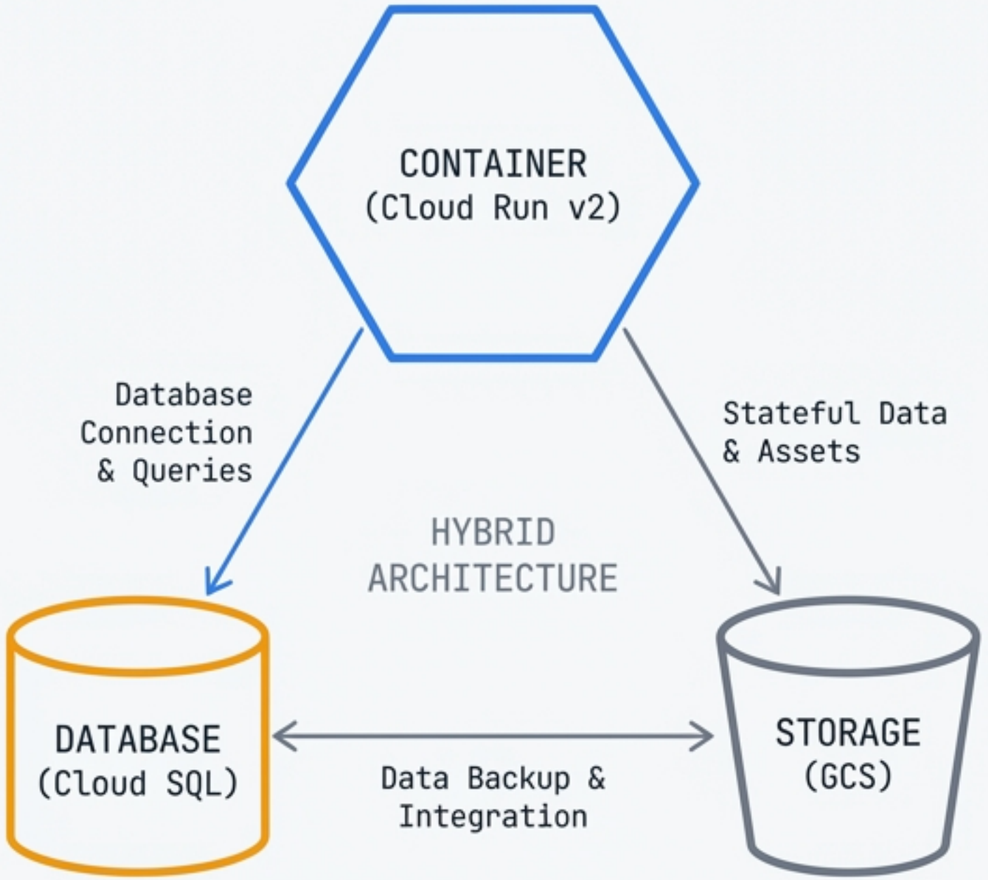
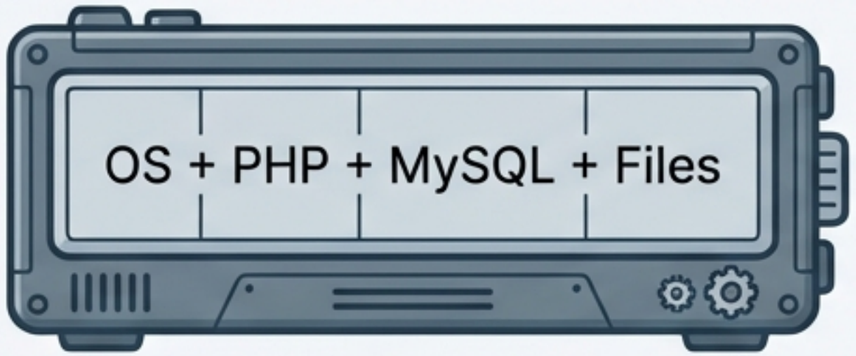
WordPress on RAD Platform: Architecture & Implementation Guide

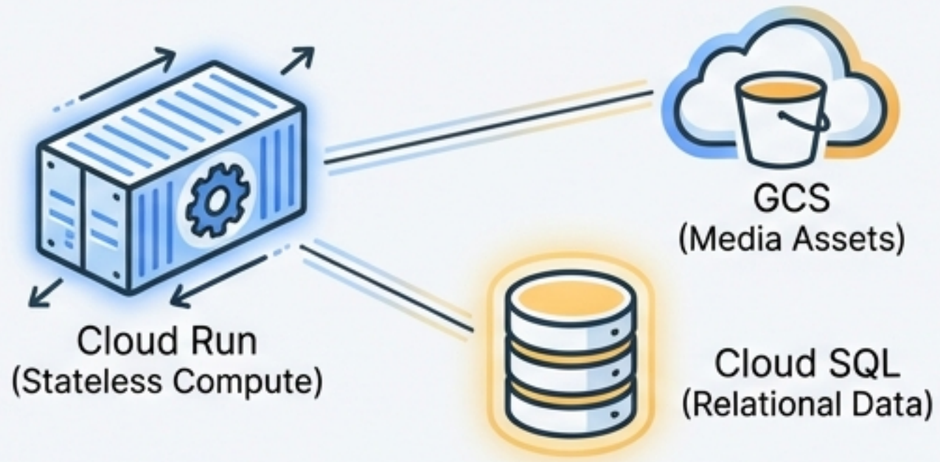
A technical deep dive into deploying containerized, stateful workloads on Cloud Run v2.



The Serverless Hybrid: WordPress on Cloud Run



Traditional Monolith



RAD Module (Decoupled)

Core Concept

This module deploys a scalable, containerized WordPress application on Google Cloud Run (v2), backed by Cloud SQL (MySQL 8.0) and Cloud Storage (GCS).

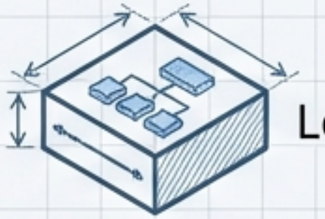
Key Distinction

Implements a wrapper around `modules/CloudRunApp` to inherit standardized infrastructure while injecting app-specific logic like custom builds and initialization jobs.

The Hybrid Approach

- **Stateless Compute**: Cloud Run provides ephemeral containers that scale to zero.
- **Stateful Data**: Cloud SQL handles relational data, while GCS persists media assets via FUSE.

High-Level Architecture



Load Balancer

Cloud Build

CI/CD Triggers
via GitHub

Cloud Run Service (WordPress)

PHP Container

Cloud SQL
(MySQL 8.0)

Unix Socket Connection

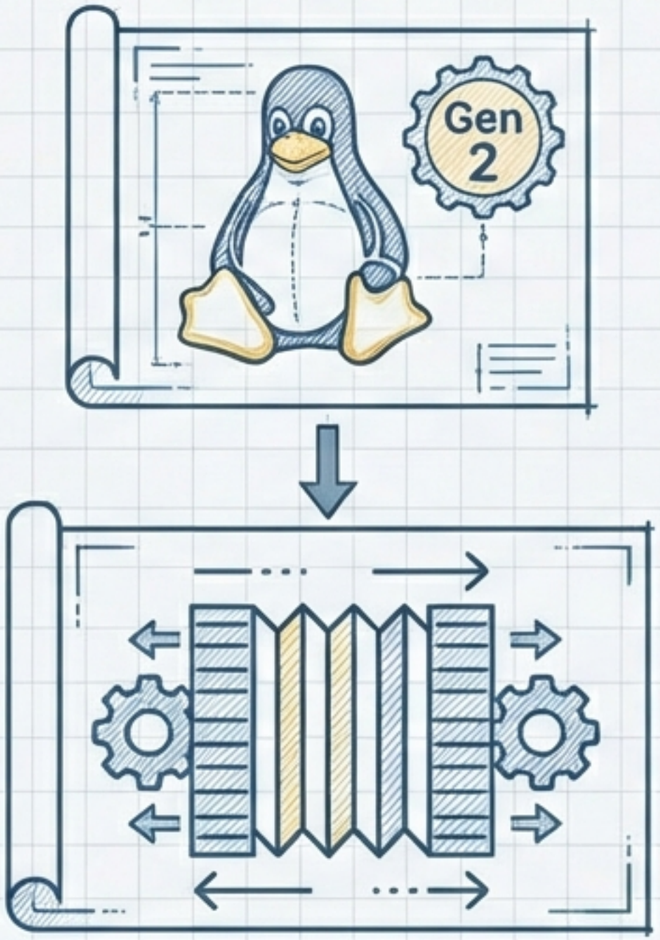
GCS FUSE Mount

GCS Bucket
(wp-uploads)

VPC Network

Compute Layer: Cloud Run Gen 2

VISUAL CONCEPTS



ELASTICITY

SPECIFICATIONS

Execution Environment

- **Type**: Gen 2 (Full Linux Compatibility)
- **Service Model**: Stateless v2 Service

Scaling Parameters

- **Instances**: User-configurable `min_instances` and `max_instances`.
- **Default**: 1–3 instances.
- **Behavior**: Scales to zero (unless min > 0) based on concurrent request volume.

Ingress Options

- **Exposure**: Configurable for Public `allUsers`, Internal-only, or Load-balanced.
- **Security Default**: `public_access = false` restricts direct invoker access.

The Storage Strategy: Decoupling State

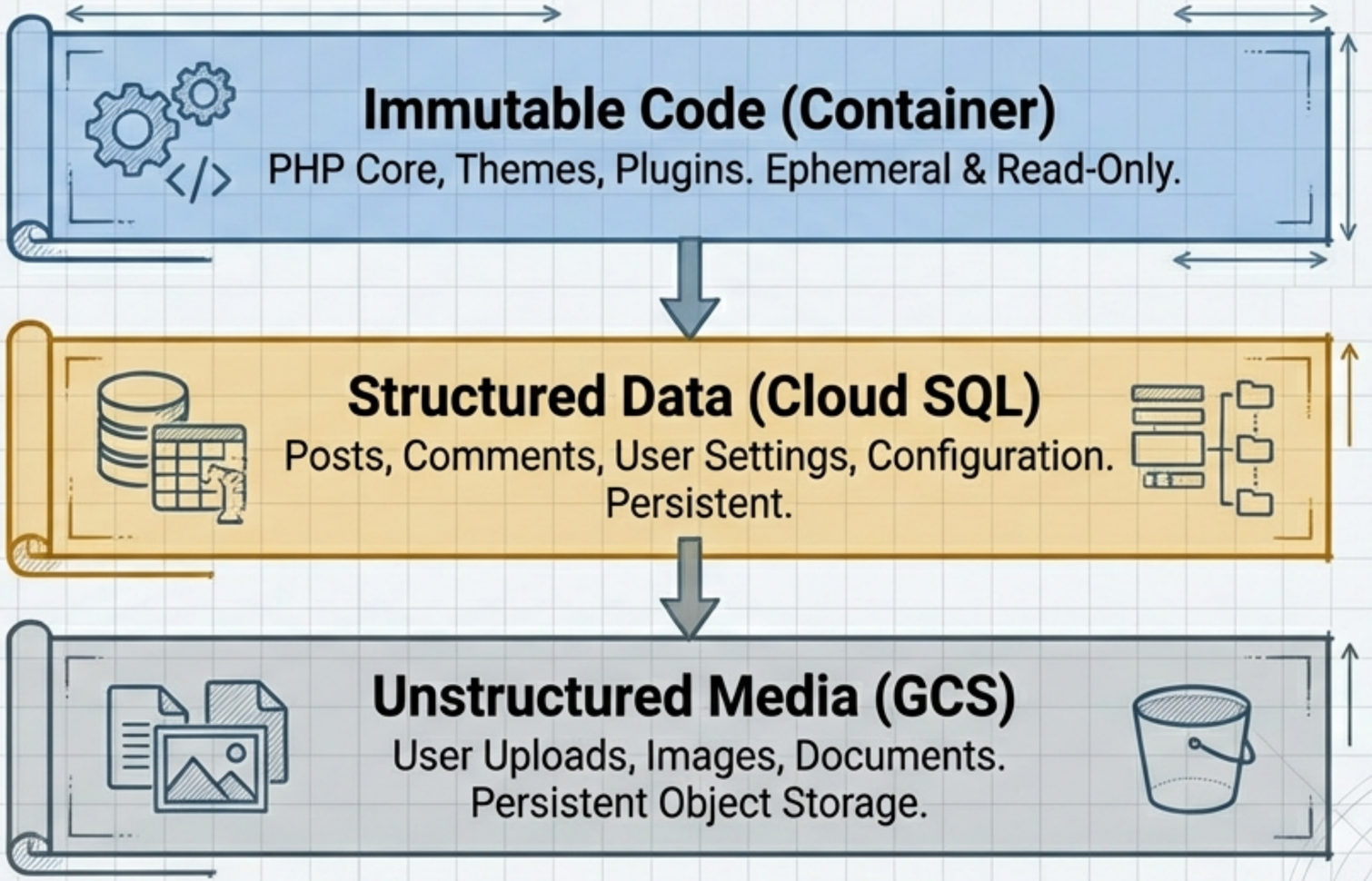
The Challenge

WordPress expects a local writable filesystem, but Cloud Run containers are ephemeral.

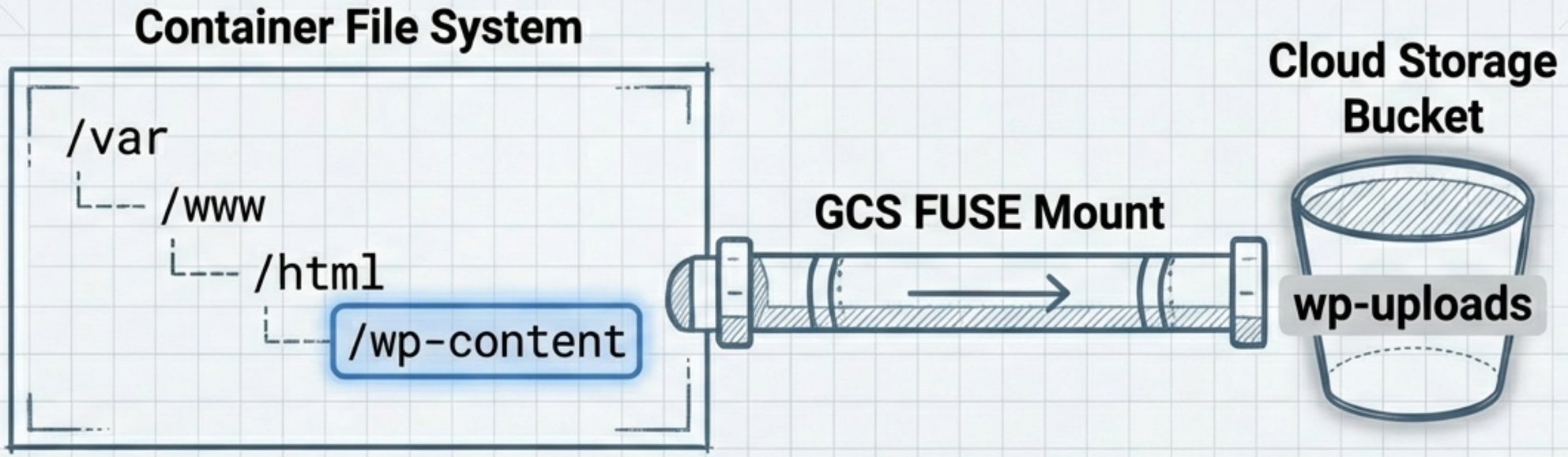
The Solution

Strict separation of concerns allows the application to be stateless while data remains stateful.

Layer Cake

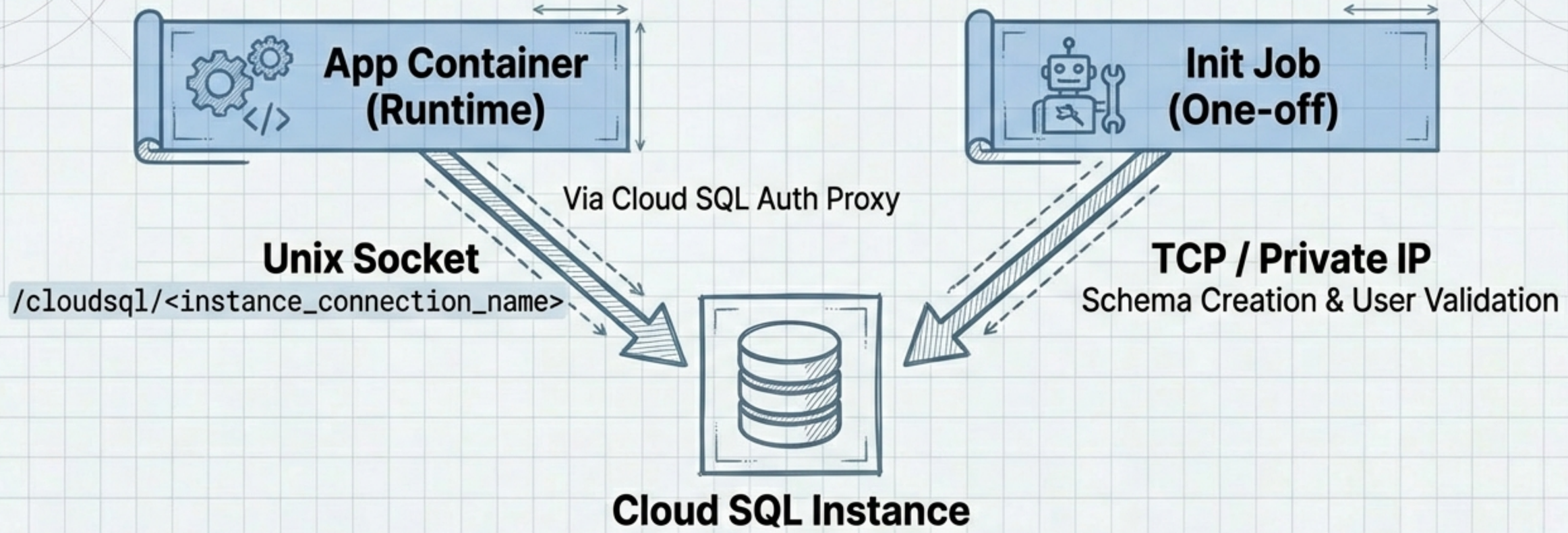


Media Persistence via GCS FUSE



- **The Mechanism:** A Google Cloud Storage bucket is mounted directly to `/var/www/html/wp-content` inside the container.
- **Why it Matters:** Ensures uploads persist across restarts and allows multiple instances to share the same media library.
- **Initialization:** Entrypoint script checks if bucket is empty; if so, populates it with default theme/plugins to prevent broken UI.

Database Connectivity Patterns

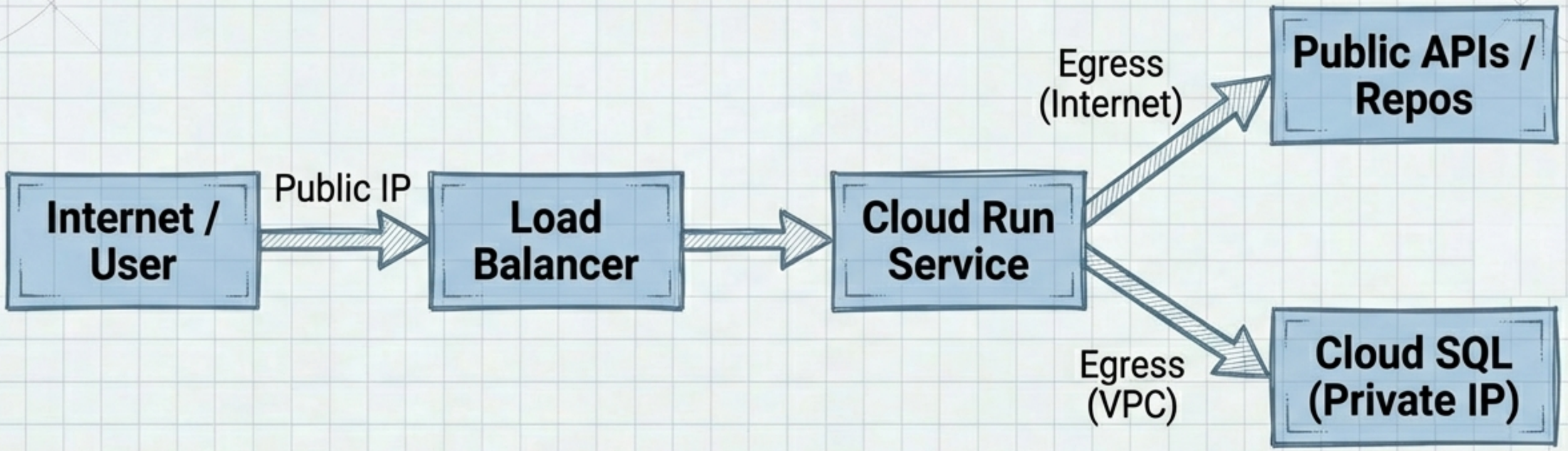


- **Provisioning:** Module expects an existing Cloud SQL instance (validated via `sql.tf`).
- **Application Connection:** Uses native Unix Socket mount for low latency.
- **Init Job Connection:** Uses Private IP over VPC for setup tasks.

IAM & Access Control

- **Strategy:** Separation of duties between Runtime and Build identities.

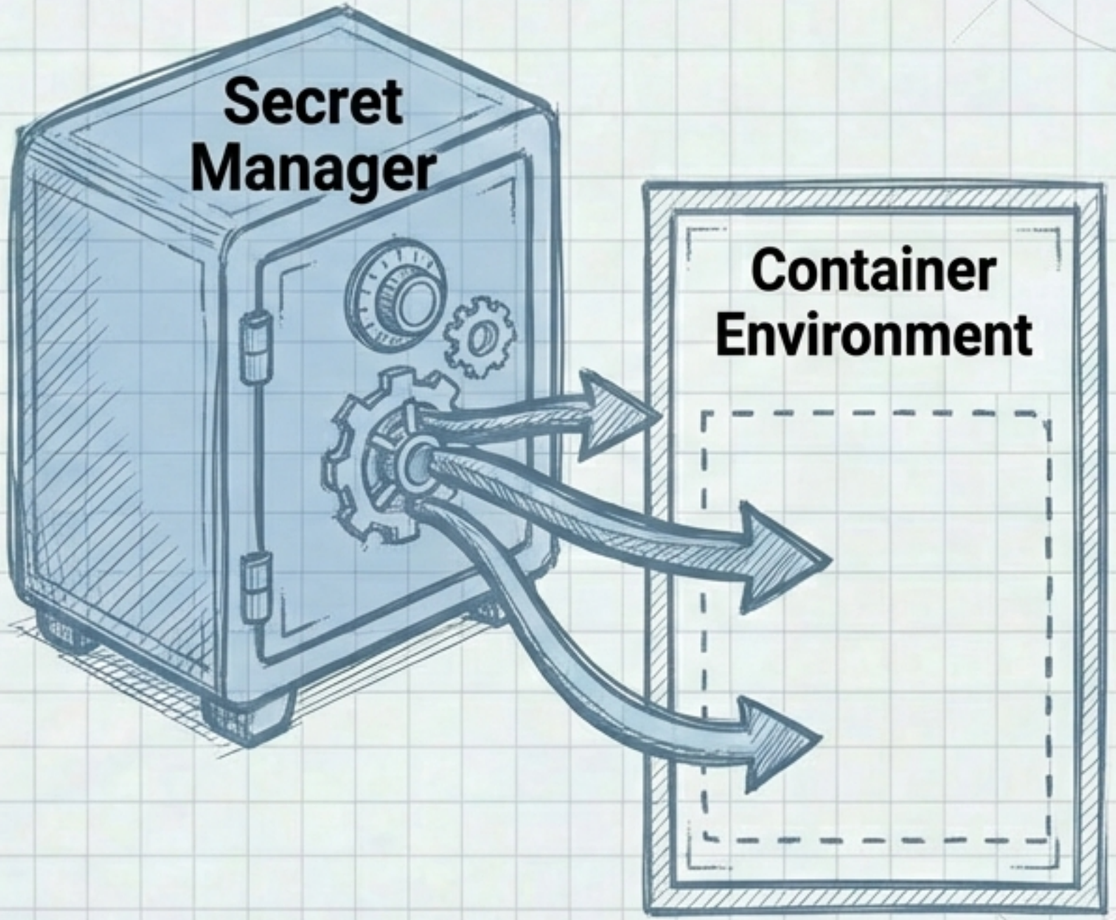
Network Security & Traffic Flow



- **VPC Access**: Direct VPC Egress enabled. `PRIVATE_RANGES_ONLY` ensures internal traffic stays private.
- **Access Control**: Default `roles/run.invoker` to `allUsers`. Restrict via `public_access = false`.
- **Logging**: Apache `mod_remoteip` configured to log true client IPs behind the Load Balancer.

Secret Management & Injection

- **Storage:** Sensitive data stored in Secret Manager, not Terraform state.
- **Key Secrets:**
  - `WORDPRESS_DB_PASSWORD`: Database credentials.
  - `WP_SALTS`: Randomly generated cryptographic salts.
- **Injection Mechanism:** Secrets mounted as environment variables at runtime.
- **Environment Context:** App reads `WORDPRESS_DB_HOST` and `WORDPRESS_TABLE_PREFIX` directly from env.



Container Specifications

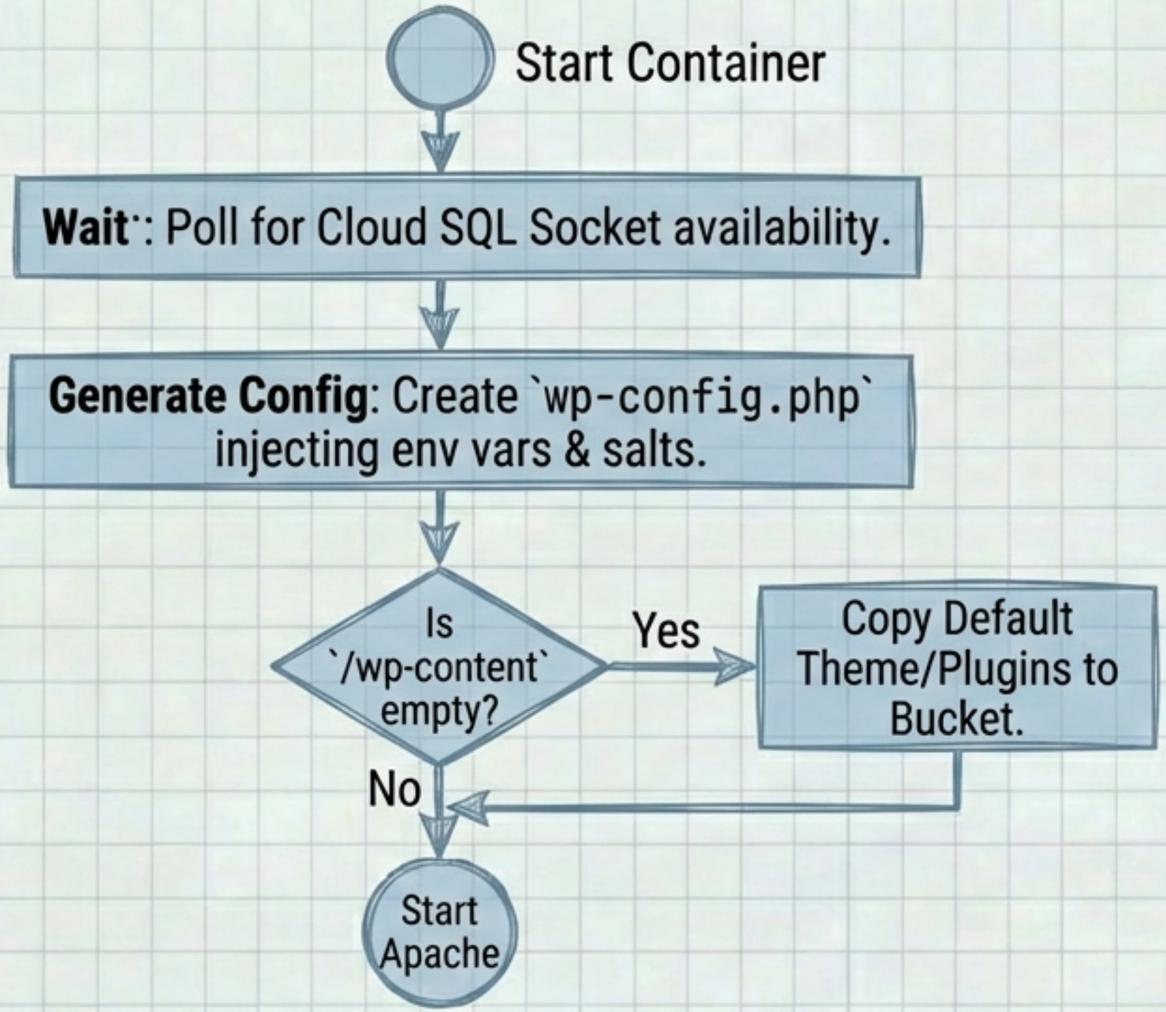
Dockerfile

```
1 FROM php:8.4-apache
2 RUN docker-php-ext-install mysqli
  bcmath intl zip
3 RUN apt-get install libmagickwand-dev
  --no-install-recommends
4 RUN pecl install imagick &&
  docker-php-ext-enable imagick
5 RUN a2enmod remoteip rewrite headers
```

- **Base Image**: Official PHP 8.4 Apache image.
- **Extensions**: Includes `gd`, `imagick` for image processing; `mysqli` for DB.
- **CI/CD**: Supports automated builds via Cloud Build triggers.

Runtime Initialization Logic

Boot Sequence (`docker-entrypoint.sh`)



DB Init Job

Separate process runs on Terraform apply to create schema/users.

Configuration Reference

Variable Name	Default / Setting
memory_limit	2Gi (Container)
php_memory_limit	128M (Overrideable)
timeout_seconds	300s
min_instances	1
max_instances	3
public_access	true (allUsers)

Variables defined in `variables.tf`.

Key Architecture Features

Auto-Initialization

Zero-touch deployment. Schema built and assets populated automatically on first run.

Stateless Hybrid

Combines Cloud Run cost-efficiency with Cloud SQL & GCS persistence.

Observability

Apache logs redirected to `stdout/stderr` for full Cloud Logging integration.

Security

Least-privilege roles, random salts, and Cloud SQL Auth Proxy sidecars.

Enhancements & Best Practices

- **Email Delivery**: PHP `mail()` fails in containers. Configure SMTP (SendGrid/Mailgun) via env vars.
- **Cloud CDN**: Enable on Load Balancer to offload static assets and reduce container load.
- **PHP Tuning**: Optimize `opcache` settings based on specific traffic patterns.
- **WAF Protection:** Enable Cloud Armor for DDoS protection and geographic restrictions.

End of Architecture Reference.