Deploying a Flask Application to Google Cloud Run: A Practical Guide

Hardcover Book Friend Finder Project

January 2026

Abstract

This essay describes the deployment of a Flask-based book recommendation web application to Google Cloud Run, a fully managed serverless platform. We cover the key concepts of containerization, serverless computing, and the specific deployment workflow used to serve the Book Friend Finder application at scale with minimal operational overhead.

1 Introduction

Modern web application deployment has evolved from managing physical servers to leveraging cloud-native platforms that abstract away infrastructure complexity. Google Cloud Run represents the current state of serverless container deployment, offering automatic scaling, pay-per-use pricing, and zero server management.

Our application, the Book Friend Finder, is a Flask web app that serves pre-computed book recommendations. It requires no database connections or external API calls at runtime—an ideal candidate for serverless deployment.

2 Understanding Google Cloud Run

2.1 What is Cloud Run?

Cloud Run is a fully managed compute platform that automatically scales stateless containers. Unlike traditional Platform-as-a-Service (PaaS) offerings, Cloud Run runs any containerized application, not just those built with specific frameworks.

Feature	**Cloud Run**	**Traditional VPS**
Scaling	Automatic (0 to N)	Manual configuration
Pricing	Per-request	Per-hour (always on)
Maintenance	Fully managed	User responsibility
Cold starts	Yes (mitigated)	No
Max instances	Configurable	Fixed

Table 1: Cloud Run vs Traditional Virtual Private Server

2.2 Key Concepts

2.2.1 Containerization

Cloud Run executes Docker containers. A container packages the application code, runtime, libraries, and dependencies into a single deployable unit. This ensures consistency between development and production environments.

Listing 1: Dockerfile for our Flask application

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
ENV PORT=8080
CMD exec gunicorn --bind :$PORT --workers 1 app:app
```

2.2.2 Statelessness

Cloud Run containers are *stateless*—they do not persist data between requests. Each request may be handled by a different container instance. This design enables horizontal scaling but requires that:

- Session data be stored externally (e.g., Redis, Firestore)
- File uploads go to cloud storage (e.g., Google Cloud Storage)
- Application state be reconstructed from external sources

Our application handles this by **pre-computing all recommendations** and bundling them as static JSON files within the container. No runtime state is required.

2.2.3 Scale to Zero

When no requests arrive, Cloud Run scales down to zero instances, incurring no compute charges. When a request arrives, a new instance starts ("cold start"), typically in 1–3 seconds for Python applications.

$$\text{Monthly Cost} = \sum_{i=1}^{N} \text{CPU}_i \times \text{Time}_i + \text{Memory}_i \times \text{Time}_i + \text{Requests} \tag{1}$$

For low-traffic applications, this model is significantly cheaper than always-on servers.

2.2.4 Revisions and Traffic Splitting

Each deployment creates a new *revision*—an immutable snapshot of the container and configuration. Cloud Run maintains revision history and supports:

- Instant rollback to previous revisions
- Traffic splitting for A/B testing (e.g., 90% to v1, 10% to v2)
- Gradual rollouts with automatic rollback on errors

3 Deployment Workflow

3.1 Prerequisites

Before deployment, we ensure the following are in place:

1. **Google Cloud Project**: A project with billing enabled
2. **APIs Enabled**: Cloud Run API, Cloud Build API, Artifact Registry API
3. **gcloud CLI**: Installed and authenticated
4. **Pre-computed Data**: All recommendation JSON files generated

3.2 Application Structure

Our deployable `webapp/` directory contains:

```
webapp/
  app.py # Flask application
  requirements.txt # Python dependencies
  templates/ # Jinja2 HTML templates
  data/
    recommendations.json # Pre-computed matches (2.1MB)
    users.json # User list (556 users)
    clusters.json # Cluster information
```

3.3 The Deployment Command

Cloud Run supports source-based deployment, automatically building the container:

Listing 2: Deployment command

```
gcloud run deploy book-friend-finder \
  --source ./webapp \
  --region us-central1 \
  --allow-unauthenticated \
  --memory 512Mi \
  --cpu 1 \
  --max-instances 10
```

Parameter explanations:

- `--source ./webapp`: Directory containing the application
- `--region us-central1`: Deployment region (affects latency and pricing)
- `--allow-unauthenticated`: Public access without authentication
- `--memory 512Mi`: RAM allocation per instance
- `--cpu 1`: vCPU allocation per instance
- `--max-instances 10`: Upper bound on concurrent instances

3.4 Build Process

When deploying from source, Cloud Build performs these steps:

1. **Upload**: Source files are uploaded to Cloud Storage
2. **Detect**: Buildpacks detect the application type (Python/Flask)
3. **Build**: A container image is created with all dependencies
4. **Push**: The image is stored in Artifact Registry
5. **Deploy**: Cloud Run creates a new revision from the image
6. **Route**: Traffic is switched to the new revision

Total deployment time is typically 4–6 minutes, with subsequent deployments faster due to layer caching.

4 Production Configuration

4.1 Resource Allocation

Choosing appropriate resources involves balancing cost and performance:

Configuration	Memory	CPU	Use Case
Minimal	256Mi	1	Simple APIs
Standard	512Mi	1	Our application
Compute-heavy	2Gi	2	ML inference
Memory-heavy	8Gi	2	Large datasets

Table 2: Resource allocation guidelines

Our application loads a 2.1MB JSON file into memory and performs dictionary lookups. 512MB provides comfortable headroom.

4.2 Concurrency Settings

Cloud Run allows configuring how many concurrent requests each instance handles:

$$\text{Required Instances} = \left\lceil \frac{\text{Concurrent Requests}}{\text{Concurrency per Instance}} \right\rceil \tag{2}$$

The default concurrency is 80 requests per instance. For CPU-bound workloads, lower values (1–10) prevent resource contention. Our I/O-bound Flask app handles 80 concurrent requests efficiently.

4.3 Cold Start Mitigation

Cold starts occur when a new instance must be initialized. Strategies to reduce impact:

1. **Minimum instances**: Keep 1+ instances always warm (`--min-instances 1`)
2. **Smaller containers**: Reduce image size for faster startup
3. **Lazy loading**: Defer non-critical initialization
4. **CPU boost**: Cloud Run provides extra CPU during startup

For our application, cold starts are approximately 2 seconds—acceptable for a recommendation service.

5 Cost Analysis

5.1 Free Tier

Cloud Run provides a generous free tier:

- 2,000,000 requests per month
- 360,000 GB-seconds of memory
- 180,000 vCPU-seconds
- 1 GB network egress (North America)

5.2 Our Application's Usage

$$\text{Request Cost} = 512\text{MB} \times 0.5\text{s} = 0.25 \text{ GB-seconds} \tag{3}$$

$$\text{Free Requests} = \frac{360,000}{0.25} = 1,440,000 \text{ requests/month} \tag{4}$$

For a personal project with moderate traffic (1,000 requests/day = 30,000/month), the application runs entirely within the free tier.

6 Updating the Deployment

When new data is available (e.g., more users, updated recommendations), the update process is:

Listing 3: Update workflow

```
# 1. Generate new pre-computed data
python3 precompute_all.py

# 2. Verify data files
ls -lh webapp/data/

# 3. Deploy new revision
gcloud run deploy book-friend-finder \
  --source ./webapp \
  --region us-central1 \
  --allow-unauthenticated
```

Cloud Run automatically:

- Creates a new revision with the updated files
- Routes 100% of traffic to the new revision
- Keeps previous revisions for instant rollback

7 Conclusion

Google Cloud Run provides an ideal deployment target for stateless web applications like our Book Friend Finder. Key advantages include:

1. **Zero infrastructure management**: No servers to patch or maintain
2. **Automatic scaling**: Handles traffic spikes without configuration
3. **Cost efficiency**: Pay only for actual usage, with generous free tier
4. **Simple deployments**: Single command to deploy from source
5. **Built-in reliability**: Automatic load balancing and health checks

For applications that can be designed as stateless services with pre-computed data, Cloud Run eliminates the operational burden of traditional hosting while providing production-grade reliability and scalability.

Live Application: `https://book-friend-finder-770103525576.us-central1.run.app`