HRMS / Payroll / ERP GCP Deployment & CI/CD Pipeline Guide

Google Cloud Platform Deployment Guide
Multi-Client Multi-Tenant Architecture
with GitHub Actions CI/CD Pipelines

Version 1.0 | February 14, 2026

Classification: Confidential

Document ID	HRMS-ERP-DEPLOY-2026-001
Version	1.0
Status	Final
Date	2026-02-14
Prepared By	DevOps & Infrastructure Team
Target Platform	Google Cloud Platform (GCP)
CI/CD Platform	GitHub Actions
Architecture	Multi-Client / Multi-Tenant

Table of Contents

1. Architecture Overview

The HRMS/Payroll/ERP system is deployed on Google Cloud Platform using a serverless-first, multi-tenant architecture. Each client organization is isolated at the database level via tenant schemas. The system serves two domains per deployment: **hrms.{client-domain}** for the React frontend (via Cloud CDN) and **api.hrms.{client-domain}** for the Django REST API (via Cloud Run). Background job processing is handled by Celery workers running as a separate Cloud Run service.

1.1 System Architecture Diagram

```
Internet
|
+-----------+-----------+
| |
hrms.{client-domain} api.hrms.{client-domain}
| |
Cloud CDN / LB Cloud Run (API)
| | |
GCS (Frontend) Cloud SQL Memorystore
Static Assets PostgreSQL Redis 7
| |
Cloud Run (Celery Worker)
|
GCS (Media/Exports/Backups)
```

1.2 Core Services

Component	GCP Service	Staging	Production
Backend API	Cloud Run	0–5 instances, 1 vCPU, 1 GiB	1–10 instances, 2 vCPU, 1 GiB
Celery Workers	Cloud Run	0–3 instances, 1 vCPU, 1 GiB	1–5 instances, 2 vCPU, 2 GiB
Database	Cloud SQL PostgreSQL 16	db-custom-1-3840, ZONAL	db-custom-4-16384, REGIONAL (HA)
Cache	Memorystore Redis 7	1 GB, BASIC	2 GB, STANDARD_HA
Frontend	GCS + Cloud CDN + HTTPS LB	staging.hrms.{domain}	hrms.{domain}
Container Images	Artifact Registry	{project}-staging-docker	{project}-production-docker
Secrets	Secret Manager	8 secrets	8 secrets
Networking	VPC + Serverless Connector	10.0.0.0/20 + 10.1.0.0/28	10.0.0.0/20 + 10.1.0.0/28

Component	GCP Service	Staging	Production
Security	Cloud Armor WAF	Basic (SQLi, XSS, rate limit)	Full OWASP + bot blocking
Monitoring	Cloud Monitoring + Logging	Email alerts	Email + Slack alerts
IaC	Terraform >= 1.5	infra/environments/staging/	infra/environments/production/
CI/CD	GitHub Actions	5 workflows	5 workflows

1.3 Terraform Modules (11)

All infrastructure is managed as code via Terraform, organized into 11 reusable modules:

Module	Purpose
networking	VPC, subnets, serverless VPC connector
security	IAM, service accounts, Cloud Armor, Workload Identity
secrets	Secret Manager (8 secrets per environment)
registry	Artifact Registry for Docker images
database	Cloud SQL PostgreSQL 16 (ZONAL or REGIONAL HA)
cache	Memorystore Redis 7
storage	GCS buckets (media, exports, backups)
backend-service	Cloud Run API + domain mapping
worker-service	Cloud Run Celery worker
frontend-cdn	GCS + CDN + HTTPS LB + SSL certificate
monitoring	Uptime checks, alert policies, dashboard, log metrics

2. Prerequisites Checklist

Before starting the deployment, ensure every item below is satisfied. Missing prerequisites will cause deployment failures.

2.1 GCP Requirements

- GCP Project created with billing enabled
  - Staging: **{project}-staging**
  - Production: **{project}-production**
- Organization-level access (or sufficient project-level IAM) to enable APIs
- Billing alert configured (recommended: notify at 50%, 80%, 100%)

2.2 Local Tooling

Tool	Version	Purpose
gcloud CLI	>= 450.0	GCP resource management and authentication
Terraform	>= 1.5.0, < 2.0.0	Infrastructure as Code provisioning
Docker	Latest stable	Container image building
Node.js	20.x	Frontend build toolchain
Python	3.12	Backend local management commands
cloud-sql-proxy	Latest	Local database access via proxy

2.3 GitHub Requirements

- GitHub repository with Actions enabled (e.g. **{org}/hrms**)
- Branch protection configured on **main** (require PR reviews)
- GitHub Environments created: **staging** and **production**
- Production environment should have **required reviewers** configured

2.4 Domain & DNS

- Domain access for the client's DNS zone
- Ability to create A/CNAME records for:
  - hrms.{client-domain} (production frontend)
  - staging.hrms.{client-domain} (staging frontend)
  - api.hrms.{client-domain} (production API)
  - api.staging.hrms.{client-domain} (staging API)

2.5 Notification Channels

- Email address for alert notifications (e.g. **devops@{client-domain}**)
- Slack webhook URL (optional, for critical production alerts to **#hrms-alerts**)

3. Step-by-Step First Deployment

This section provides a complete walkthrough for deploying the HRMS/Payroll/ERP system to GCP for the first time. All commands assume you are in the repository root unless otherwise specified.

3.1 Bootstrap GCP Project

This step enables required GCP APIs, creates the Terraform state bucket, and creates the Terraform service account. Perform this for each GCP project (staging and production).

Step 1: Set Environment Variables

```
export PROJECT_ID="{project}-staging" # e.g. "acme-hrms-staging"
export REGION="us-central1" # or your preferred region
export TF_STATE_BUCKET="{project}-terraform-state-staging"

gcloud auth login
gcloud config set project $PROJECT_ID
```

Step 2: Enable Required GCP APIs

```
gcloud services enable \
run.googleapis.com \
sqladmin.googleapis.com \
redis.googleapis.com \
artifactregistry.googleapis.com \
secretmanager.googleapis.com \
compute.googleapis.com \
vpcaccess.googleapis.com \
servicenetworking.googleapis.com \
cloudresourcemanager.googleapis.com \
iam.googleapis.com \
iamcredentials.googleapis.com \
monitoring.googleapis.com \
logging.googleapis.com \
cloudbuild.googleapis.com \
containerscanning.googleapis.com \
--project=$PROJECT_ID
```

Step 3: Create Terraform State Bucket

```
gcloud storage buckets create gs://$TF_STATE_BUCKET \
--project=$PROJECT_ID \
--location=$REGION \
--uniform-bucket-level-access \
--public-access-prevention

# Enable versioning for state recovery
gcloud storage buckets update gs://$TF_STATE_BUCKET --versioning
```

Step 4: Create Terraform Service Account

Create a service account with the required roles for Terraform to manage all GCP resources:

```
gcloud iam service-accounts create terraform \
--display-name="Terraform" --project=$PROJECT_ID

# Grant required roles
for ROLE in roles/editor roles/iam.securityAdmin \
roles/secretmanager.admin roles/compute.networkAdmin \
roles/run.admin roles/cloudsql.admin roles/redis.admin \
roles/storage.admin roles/monitoring.admin \
roles/logging.admin roles/artifactregistry.admin \
roles/iam.workloadIdentityPoolAdmin; do
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:terraform@${PROJECT_ID}.iam.gserviceaccount.com" \
--role="$ROLE" --quiet
done
```

WARNING: Delete local service account keys after initial bootstrapping. Once Workload Identity Federation is configured, GitHub Actions uses OIDC tokens instead.

3.2 Configure GitHub Secrets

Navigate to GitHub > Repository Settings > Secrets and Variables > Actions and configure the following secrets and variables.

Repository-Level Secrets (Shared)

Secret Name	Description
GCP_WORKLOAD_IDENTITY_PROVIDER	Workload Identity Provider resource name (from Terraform output)
GCP_SERVICE_ACCOUNT_EMAIL	CI/CD service account email (from Terraform output)

Environment-Level Secrets

Secret Name	Staging Value	Production Value
GCP_PROJECT_ID_*	{project}-staging	{project}-production
GCP_WORKLOAD_IDENTITY_PROVIDER	(staging pool)	(production pool)
GCP_SERVICE_ACCOUNT_EMAIL	(staging CI/CD SA)	(production CI/CD SA)

Environment-Level Variables

Variable	Staging	Production
GCP_REGION	us-central1	us-central1
STAGING_API_URL	https://api.staging.hrms.{domain}	—
STAGING_BUCKET	{project}-staging-frontend	—
PRODUCTION_API_URL	—	https://api.hrms.{domain}
PRODUCTION_BUCKET	—	{project}-production-frontend
SLACK_WEBHOOK_URL	(optional)	(recommended)

TIP: The Workload Identity Provider and CI/CD service account are created by Terraform. On the first deploy, bootstrap manually or use a temporary service account key, then update these secrets after terraform apply.

3.3 Deploy Infrastructure with Terraform

All Terraform operations use the Makefile in the **infra/** directory. The Makefile validates that ENV is set and that the corresponding environment config exists.

```
cd infra
# Step 1: Initialize Terraform for staging
make init ENV=staging
# Step 2: Review the execution plan
make plan ENV=staging
```

```
# Step 3: Apply (after careful review)
make apply ENV=staging
```

Expected Resources Created (First Run)

- 1 VPC network + subnet + serverless VPC connector
- 2 service accounts (API + Worker) + 1 CI/CD service account
- 1 Cloud Armor WAF policy
- 1 Workload Identity Pool + Provider (for GitHub Actions)
- 8 Secret Manager secrets (empty — populate in step 3.5)
- 1 Artifact Registry repository
- 1 Cloud SQL PostgreSQL instance
- 1 Memorystore Redis instance
- 3 GCS buckets (media, exports, backups)
- Cloud Run API + Worker services
- GCS frontend bucket + CDN + HTTPS LB + SSL certificate
- Monitoring: uptime checks, 16 alert policies, 4 log metrics, 1 dashboard

```
# Save outputs (you will need them for subsequent steps)
make output ENV=staging
```

> WARNING: Cloud Run requires a container image to exist. If Terraform partially fails because the Artifact Registry is empty, proceed to step 3.4 to push an initial image, then re-run make apply ENV=staging.

3.4 Build & Push Docker Images

Before Cloud Run can start, it needs at least one container image in Artifact Registry.

```
# Variables (from Terraform output)
export PROJECT_ID="{project}-staging"
export REGION="us-central1"
export REGISTRY="${REGION}-docker.pkg.dev"
export REPOSITORY="${PROJECT_ID}/{project}-staging-docker"

# Authenticate Docker to Artifact Registry
gcloud auth configure-docker ${REGISTRY} --quiet

# Build and push the backend API image
docker build \
-t ${REGISTRY}/${REPOSITORY}/backend:initial \
-t ${REGISTRY}/${REPOSITORY}/backend:latest \
./HRMS/backend

docker push ${REGISTRY}/${REPOSITORY}/backend:initial
docker push ${REGISTRY}/${REPOSITORY}/backend:latest

# Build and push the Celery worker image
docker build \
-t ${REGISTRY}/${REPOSITORY}/celery-worker:initial \
-t ${REGISTRY}/${REPOSITORY}/celery-worker:latest \
-f ./HRMS/backend/Dockerfile.celery \
./HRMS/backend

docker push ${REGISTRY}/${REPOSITORY}/celery-worker:initial
docker push ${REGISTRY}/${REPOSITORY}/celery-worker:latest
```

> TIP: After this initial push, all subsequent image builds are handled by GitHub Actions CI/CD (deploy-staging.yml and deploy-production.yml).

3.5 Configure Secrets in Secret Manager

Terraform created 8 empty secrets. You must add the initial secret values manually.

Secret ID Pattern	Description	Generation Method
*-django-secret-key	Django SECRET_KEY	python3 -c 'import secrets; print(secrets.token_urlsafe(64))'
*-db-password	Cloud SQL database password	python3 -c 'import secrets; print(secrets.token_urlsafe(32))'
*-redis-auth	Memorystore Redis AUTH	python3 -c 'import secrets; print(secrets.token_urlsafe(32))'
*-jwt-signing-key	JWT token signing key	python3 -c 'import secrets; print(secrets.token_urlsafe(64))'
*-anthropic-api-key	Anthropic API key	From Anthropic account dashboard
*-email-host-password	SMTP email password	From SMTP provider
*-ldap-bind-password	LDAP bind password	From Active Directory admin

Secret ID Pattern	Description	Generation Method
*-azure-client-secret	Azure AD client secret	From Azure portal

```
# Example: Set Django secret key
echo -n "$(python3 -c 'import secrets; print(secrets.token_urlsafe(64))')" | \
gcloud secrets versions add "${PREFIX}-django-secret-key" \
--data-file=- --project=$PROJECT_ID
```

CRITICAL: You must also set the database password on the Cloud SQL instance itself using gcloud sql users set-password. Use the SAME password stored in Secret Manager.

3.6 Run Initial Migrations

Database migrations must be applied before the API can serve requests.

Option A: Via Cloud Run Job (Preferred)

```
gcloud run jobs execute ${PROJECT_ID}-api-migrate \
--region=$REGION --project=$PROJECT_ID --wait
```

Option B: Via Cloud SQL Auth Proxy (Local)

```
# Start proxy
CONN_NAME=$(cd infra && terraform output -raw db_connection_name)
cloud-sql-proxy $CONN_NAME --port=5432 &

# Run migrations
cd HRMS/backend
export DJANGO_ENV=production USE_POSTGRES=true
export DB_HOST=127.0.0.1 DB_PORT=5432 DB_NAME=hrms DB_USER=hrms_app
export DB_PASSWORD="<from-secret-manager>"
python manage.py migrate --noinput
```

3.7 Create Superuser & Load Seed Data

```
# Create superuser
python manage.py shell -c "
from accounts.models import User
if not User.objects.filter(email='admin@{domain}').exists():
User.objects.create_superuser(
email='admin@{domain}',
username='system-admin',
password='CHANGE-ME-ON-FIRST-LOGIN',
)"
```

CRITICAL: Change the superuser password immediately after first login.

```
# Load seed data (idempotent)
./scripts/deployment/seed-production.sh --env staging --proxy
```

3.8 Build & Upload Frontend

The frontend is a React/Vite SPA served from GCS via Cloud CDN.

```
cd HRMS/frontend
npm ci
VITE_API_URL=https://api.staging.hrms.{domain} npm run build

BUCKET="${PROJECT_ID}-frontend"

# Static assets: immutable, 1-year cache
gcloud storage cp dist/assets/* "gs://${BUCKET}/assets/" \
--recursive --cache-control="public, max-age=31536000, immutable"

# HTML: no cache (always fetch latest)
gcloud storage cp dist/index.html "gs://${BUCKET}/" \
--cache-control="no-cache, no-store, must-revalidate"

# Invalidate CDN cache
gcloud compute url-maps invalidate-cdn-cache \
${PROJECT_ID}-frontend-urlmap --path="/index.html" \
--project=$PROJECT_ID --async
```

3.9 Configure DNS & Verify SSL

DNS Records to Create

Type	Host	Value	TTL
A	staging.hrms.{domain}	(from terraform output)	300
A	hrms.{domain}		300
CNAME	api.staging.hrms.{domain}	ghs.googlehosted.com.	300
CNAME	api.hrms.{domain}	ghs.googlehosted.com.	300
CNAME	www.hrms.{domain}	hrms.{domain}	3600
CAA	hrms.{domain}	0 issue "pki.goog"	3600

Google-managed SSL certificates are provisioned automatically. Certificate provisioning typically takes 15–60 minutes after DNS propagation but can take up to 24 hours.

```
# Verify DNS
dig staging.hrms.{domain} +short
dig api.staging.hrms.{domain} +short

# Check SSL certificate status
gcloud compute ssl-certificates list --project=$PROJECT_ID \
--format="table(name, type, managed.status, managed.domainStatus)"
```

3.10 Smoke Tests

Test	Command	Expected
API Health	curl -s -o /dev/null -w '%{http_code}' ${API_URL}/healthz/	200
API Readiness	curl -s ${API_URL}/readyz/ | python3 -m json.tool	JSON with DB+Redis status
Frontend	curl -s -o /dev/null -w '%{http_code}' https://staging.hrms.{domain}	200
Worker	gcloud run services describe ${PROJECT_ID}-worker ...	status: True
Cache	curl ${API_URL}/api/v1/core/cache/stats/ -H 'Authorization: Bearer ...'	JSON stats

4. CI/CD Pipeline Reference

The project uses 5 GitHub Actions workflows that automate the entire build, test, and deployment lifecycle. All workflows are stored in **.github/workflows/**.

4.1 CI Pipeline

File: .github/workflows/ci.yml

Triggers: Pull requests to develop and main

Job	Description	Key Tools
backend-lint	Ruff lint + format check, Bandit security scan	Ruff, Bandit
backend-test	Django test suite with PostgreSQL 16 + Redis 7	pytest, coverage >= 80%
backend-security	Dependency vulnerability scan	pip-audit
frontend-lint	ESLint + TypeScript type check	ESLint, tsc --noEmit
frontend-build	Vite production build verification	npm run build
frontend-security	Frontend dependency audit	npm audit
docker-build	Verify Dockerfiles build successfully	docker build

TIP: CI runs automatically on every pull request. All 7 jobs must pass before merging is allowed.

4.2 Deploy Staging

File: .github/workflows/deploy-staging.yml

Triggers: Push to develop branch

Flow:

- 1. Run CI checks (lint, test, security, build)
- 2. Build & push Docker images to Artifact Registry
- 3. Run database migrations via Cloud Run Job
- 4. Deploy API to Cloud Run (canary then 100%)
- 5. Build & upload frontend to GCS
- 6. Invalidate CDN cache
- 7. Run smoke tests
- 8. Send Slack notification (if configured)

4.3 Deploy Production (Canary)

File: .github/workflows/deploy-production.yml

Triggers: Push to main branch

Production deployments use a canary strategy with automatic rollback. The workflow requires manual approval from designated reviewers before proceeding.

Production-Specific Steps:

- 1. CI checks (all 7 jobs must pass)
- 2. Build & push Docker images
- 3. **Pre-deploy database backup** (automatic snapshot)
- 4. Run database migrations
- 5. **Canary deployment** (0% → 10% → 50% → 100%)
- 6. Deploy frontend + invalidate CDN
- 7. Run smoke tests
- 8. **Create Git tag + GitHub Release**
- 9. Slack notification

4.4 Rollback Workflow

File: .github/workflows/rollback.yml

Triggers: Manual dispatch (workflow_dispatch)

Input	Type	Description
environment	Choice	staging or production
revision	String	Cloud Run revision name or 'previous'
reason	String	Required text explanation for the rollback

Actions: Shifts traffic to specified revision, invalidates CDN, verifies health, notifies Slack.

4.5 Scheduled Maintenance

File: .github/workflows/scheduled-maintenance.yml

Schedule	Frequency	Action
Mon 06:00 UTC	Weekly	Python + Node.js dependency vulnerability audit
Daily 08:00 UTC	Daily	Verify production Cloud SQL backup exists and is < 25h old
1st of month 09:00 UTC	Monthly	Service account key rotation audit

Creates GitHub issues automatically if vulnerabilities or backup failures are detected.

5. GitHub Actions Workflow Details

This section provides detailed implementation guidance for each GitHub Actions workflow.

5.1 Authentication: Workload Identity Federation

GitHub Actions authenticates to GCP using Workload Identity Federation (WIF), eliminating the need for long-lived service account keys. This is the recommended and most secure approach.

```
# In every workflow job that interacts with GCP:
- uses: google-github-actions/auth@v2
with:
workload_identity_provider: ${{ secrets.GCP_WORKLOAD_IDENTITY_PROVIDER }}
service_account: ${{ secrets.GCP_SERVICE_ACCOUNT_EMAIL }}
# Setup gcloud CLI
- uses: google-github-actions/setup-gcloud@v2
with:
project_id: ${{ vars.GCP_PROJECT_ID }}
```

How WIF Works

- GitHub OIDC provider issues a short-lived JWT token to the workflow
- The token is exchanged with GCP's Security Token Service (STS)
- STS validates the token against the Workload Identity Pool/Provider
- A federated access token is returned, scoped to the CI/CD service account
- No service account keys are stored in GitHub Secrets

> TIP: Workload Identity Federation is configured by the Terraform 'security' module. The pool and provider are automatically created during terraform apply.

5.2 Docker Image Build & Push

```
# Build step in deploy workflow
- name: Configure Docker for Artifact Registry
run: gcloud auth configure-docker ${REGION}-docker.pkg.dev --quiet
- name: Build and push API image
run: |
docker build \
-t ${REGISTRY}/${REPOSITORY}/backend:${GITHUB_SHA::8} \
-t ${REGISTRY}/${REPOSITORY}/backend:latest \
./HRMS/backend
docker push ${REGISTRY}/${REPOSITORY}/backend --all-tags
- name: Build and push Worker image
run: |
docker build \
-t ${REGISTRY}/${REPOSITORY}/celery-worker:${GITHUB_SHA::8} \
-t ${REGISTRY}/${REPOSITORY}/celery-worker:latest \
-f ./HRMS/backend/Dockerfile.celery \
./HRMS/backend
```

```
docker push ${REGISTRY}/${REPOSITORY}/celery-worker --all-tags
```

Images are tagged with both the short commit SHA (for traceability) and **latest** (for convenience). Artifact Registry retains all versions.

5.3 Database Migration Job

```
- name: Run migrations
run: |
gcloud run jobs execute ${PROJECT_PREFIX}-${ENV}-api-migrate \
--region=${{ vars.GCP_REGION }} \
--project=${{ vars.GCP_PROJECT_ID }} \
--wait
```

The migration Cloud Run Job is defined in Terraform. It uses the same container image as the API but runs **python manage.py migrate --noinput** as its entrypoint. The job has access to the database via the VPC connector and reads credentials from Secret Manager.

5.4 Frontend Deploy Step

```
- name: Build frontend
working-directory: HRMS/frontend
run: |
npm ci
VITE_API_URL=${{ vars.API_URL }} npm run build
- name: Upload to GCS
run: |
# Static assets with immutable caching
gcloud storage cp dist/assets/* gs://${BUCKET}/assets/ \
--recursive --cache-control="public, max-age=31536000, immutable"
# HTML with no-cache
gcloud storage cp dist/index.html gs://${BUCKET}/ \
--cache-control="no-cache, no-store, must-revalidate"
- name: Invalidate CDN
run: |
gcloud compute url-maps invalidate-cdn-cache \
${PROJECT_PREFIX}-${ENV}-frontend-urlmap \
--path="/index.html" --project=$PROJECT_ID --async
```

6. Workload Identity Federation

Workload Identity Federation allows GitHub Actions to authenticate to GCP without long-lived service account keys. This section details the setup and security model.

6.1 Architecture

```
GitHub Actions Runner
|
| (1) Request OIDC token from GitHub
v
GitHub OIDC Provider --> JWT with claims:
| - iss: https://token.actions.githubusercontent.com
| - sub: repo:{org}/hrms:ref:refs/heads/develop
| - aud: https://iam.googleapis.com/...
|
| (2) Exchange JWT for GCP access token
v
GCP Security Token Service (STS)
|
| (3) Validate against Workload Identity Pool
v
Workload Identity Pool + Provider
|
| (4) Return federated access token
v
CI/CD Service Account --> Scoped permissions
```

6.2 Terraform Configuration (security module)

The Terraform security module creates the following resources:

- **Workload Identity Pool:** {project}-{env}-github-pool
- **Workload Identity Provider:** {project}-{env}-github-provider
- **CI/CD Service Account:** {project}-{env}-cicd-sa
- **Attribute mapping:** google.subject = assertion.sub
- **Attribute condition:** assertion.repository == '{org}/hrms'

> WARNING: The attribute condition restricts authentication to your specific repository only. No other repository can impersonate the CI/CD service account.

6.3 CI/CD Service Account Roles

Role	Purpose
roles/run.admin	Deploy and manage Cloud Run services
roles/artifactregistry.writer	Push Docker images

Role	Purpose
roles/storage.objectAdmin	Upload frontend to GCS, manage media
roles/secretmanager.secretAccessor	Read secrets for deployment
roles/cloudsql.client	Connect to Cloud SQL for migrations
roles/compute.urlMapUpdater	Invalidate CDN cache
roles/monitoring.viewer	Read metrics for canary decision
roles/iam.serviceAccountUser	Act as API/Worker service accounts

7. Canary Deployment Strategy

Production deployments use a gradual traffic shifting (canary) strategy with automatic rollback based on error rate monitoring. This minimizes blast radius when deploying potentially breaking changes.

7.1 Traffic Shifting Stages

Stage	Traffic %	Duration	Success Criteria	Failure Action
Deploy	0%	Immediate	Revision created successfully	Abort deployment
Canary	10%	5 minutes	Error rate < 1%	Auto-rollback to previous
Ramp	50%	5 minutes	Error rate < 1%	Auto-rollback to previous
Full	100%	Permanent	Deployment complete	—

7.2 Error Rate Monitoring

During each canary stage, the workflow queries Cloud Monitoring for the error rate (5xx responses / total responses) over the monitoring window:

```
# Query error rate from Cloud Monitoring
ERROR_RATE=$(gcloud monitoring time-series list \
--project=$PROJECT_ID \
--filter="resource.type=cloud_run_revision AND
metric.type=run.googleapis.com/request_count AND
resource.labels.service_name=$SERVICE AND
metric.labels.response_code_class=5xx" \
--interval-start=$(date -u -d '5 minutes ago' +%Y-%m-%dT%H:%M:%SZ) \
--format=json | jq '...')
if [ "$(echo "$ERROR_RATE > 0.01" | bc)" -eq 1 ]; then
echo 'Error rate exceeds 1%. Rolling back...'
gcloud run services update-traffic $SERVICE \
--to-revisions=${PREV_REVISION}=100 ...
fi
```

7.3 Rollback Flow

```
Canary Error Rate > 1%
|
+--> Shift 100% traffic to previous revision
+--> Invalidate CDN cache
+--> Send Slack alert with failure details
+--> Mark GitHub Actions workflow as failed
+--> Create GitHub issue for investigation
```

8. Rollback Procedures

8.1 Cloud Run API Rollback

Option A: Via GitHub Actions (Recommended)

Go to Actions > Rollback > Run workflow, select the environment, and specify **previous** or a specific revision name.

Option B: Via CLI

```
# List recent revisions
gcloud run revisions list --service=$SERVICE \
--region=$REGION --project=$PROJECT_ID \
--sort-by="~metadata.creationTimestamp" --limit=5

# Roll back to specific revision
gcloud run services update-traffic $SERVICE \
--to-revisions="${PREV_REVISION}=100" \
--region=$REGION --project=$PROJECT_ID
```

8.2 Database Rollback

Method	Use Case	Downtime
Reverse Django migration	Known migration caused the issue	None (online)
Point-in-time recovery (PITR)	Data corruption, accidental deletes	Minutes (clone + swap)
Backup restore	Full recovery needed	Yes (replaces all data)

8.3 Frontend Rollback

Re-upload the previous build artifacts to GCS and invalidate the CDN cache:

```
# Re-upload previous build
gcloud storage cp /path/to/previous/dist/assets/* "gs://${BUCKET}/assets/" \
--recursive --cache-control="public, max-age=31536000, immutable"
gcloud storage cp /path/to/previous/dist/index.html "gs://${BUCKET}/" \
--cache-control="no-cache, no-store, must-revalidate"

# Invalidate CDN
gcloud compute url-maps invalidate-cdn-cache \
${PROJECT_PREFIX}-production-frontend-urlmap --path="/*" \
--project=$PROJECT_ID --async
```

8.4 Emergency Kill Switch

CRITICAL: Use only when the application is causing active harm and must be stopped immediately.

```
# Scale API and worker to zero (takes effect within seconds)
```

```
gcloud run services update ${PROJECT_PREFIX}-production-api \
--min-instances=0 --max-instances=0 \
--region=$REGION --project=$PROJECT_ID
gcloud run services update ${PROJECT_PREFIX}-production-worker \
--min-instances=0 --max-instances=0 \
--region=$REGION --project=$PROJECT_ID
```

9. Infrastructure Reference

9.1 Terraform State

Environment	State Bucket	Prefix
Staging	{project}-terraform-state-staging	terraform/state
Production	{project}-terraform-state-production	terraform/state

9.2 Makefile Targets

Target	Description
make init ENV=	Initialize Terraform backend
make plan ENV=	Generate execution plan
make apply ENV=	Apply changes
make output ENV=	Show outputs
make destroy ENV=	Destroy all resources (requires confirmation)
make state-list ENV=	List all resources in state
make fmt	Format all .tf files
make validate	Validate configuration
make staging-plan	Shortcut: init + plan for staging
make staging-apply	Shortcut: init + apply for staging
make production-plan	Shortcut: init + plan for production
make production-apply	Shortcut: init + apply for production

9.3 Secret Manager Secrets (8 per environment)

Secret ID Pattern	Description	Used By
*-django-secret-key	Django SECRET_KEY	API, Worker
*-db-password	Cloud SQL database password	API, Worker
*-redis-auth	Memorystore Redis AUTH string	API, Worker
*-jwt-signing-key	JWT token signing key	API, Worker
*-anthropic-api-key	Anthropic API key for AI features	API, Worker
*-email-host-password	SMTP email password	API, Worker

Secret ID Pattern	Description	Used By
*-ldap-bind-password	LDAP bind password	API, Worker
*-azure-client-secret	Azure AD client secret	API, Worker

9.4 Cloud Run Services

Service	Staging	Production
API	{project}-staging-api	{project}-production-api
Celery Worker	{project}-staging-worker	{project}-production-worker
Migration Job	{project}-staging-api-migrate	{project}-production-api-migrate

9.5 Container Images

Image	Dockerfile	Entrypoint
backend	HRMS/backend/Dockerfile	Gunicorn on port 8080
celery-worker	HRMS/backend/Dockerfile.celery	Celery: 4 queues, concurrency 4

9.6 Network Architecture

```
VPC: 10.0.0.0/20
Subnet: Primary range
Secondary ranges (reserved for future GKE):
Pods: 10.4.0.0/14
Services: 10.8.0.0/20
Serverless VPC Connector: 10.1.0.0/28
Cloud SQL: Private IP (via VPC peering)
Redis: Private IP (via VPC peering)
Cloud Run: Connects via Serverless VPC Connector
(PRIVATE_RANGES_ONLY egress)
```

9.7 Staging vs Production Differences

Setting	Staging	Production
DB Availability	ZONAL	REGIONAL (HA)
Redis Tier	BASIC	STANDARD_HA
Cloud Armor	Basic (SQLi, XSS, rate limit)	Full OWASP + bot blocking
Min API Instances	0 (scale to zero)	1 (always warm)

Setting	Staging	Production
Min Worker Instances	0 (scale to zero)	1 (always warm)
Deletion Protection	Disabled	Enabled
Point-in-time Recovery	Disabled	Enabled
Backup Retention	7 days	30 days

10. Monitoring & Alerting

10.1 Alert Policies (16 per environment)

Critical Alerts (Email + Slack):

Alert	Condition	Duration
API Liveness failure	Uptime check down	5 min
API Latency > 8s p95	Sustained high latency	5 min
API Error Rate > 5%	High 5xx error rate	5 min
Cloud SQL CPU > 90%	Database CPU saturation	5 min
Cloud SQL Memory > 92%	Database memory pressure	5 min
Cloud SQL Disk > 90%	Database disk near full	5 min
Redis Memory > 85%	Cache memory pressure	5 min
App Error Spike > 50	Burst of application errors	5 min

Warning Alerts (Email only):

Alert	Condition	Duration
API Readiness failure	Heavy check down	10 min
API Latency > 3s p95	Elevated latency	5 min
API Error Rate > 2%	Elevated error rate	5 min
Cloud SQL CPU > 70%	Database CPU elevated	5 min
Cloud SQL Memory > 80%	Database memory elevated	5 min
Cloud SQL Disk > 75%	Database disk growing	5 min
Redis Memory > 70%	Cache memory elevated	5 min
Worker at Max Scale	All worker slots consumed	10 min

10.2 Custom Dashboard (12 widgets)

A Terraform-managed dashboard is created automatically with the following widgets:

- API Request Latency (p50/p95/p99)
- API Request Count by Status Code
- API Instance Count
- Cloud SQL CPU Utilization
- Cloud SQL Memory Utilization
- Cloud SQL Active Connections

- Cloud SQL Disk Utilization
- Redis Memory Usage
- Redis Cache Hit Ratio
- Worker Instance Count
- Application Errors (log-based)
- Celery Task Failures (log-based)

10.3 Log-Based Metrics

Metric	Source Filter	Type
*_app_errors	jsonPayload.level="ERROR"	DELTA / INT64
*_slow_queries	jsonPayload.event="slow_query"	DISTRIBUTION
*_task_failures	jsonPayload.event="task_failure"	DELTA / INT64
*_http_request_latency	jsonPayload.event="http_request"	DISTRIBUTION

11. Troubleshooting

Common issues encountered during first deployment and their resolutions.

1. Terraform: '409 Conflict' creating Cloud SQL

Cloud SQL instance names are globally unique and cannot be reused for 7 days after deletion. Change the name_prefix in your tfvars or wait 7 days.

2. Cloud Run: 'Image not found'

Terraform tried to create the Cloud Run service before images exist. Push initial images (step 3.4), then re-run make apply.

3. Cloud Run: 'Container failed to start'

Check logs with: gcloud run services logs read SERVICE --region=REGION --limit=50. Common causes: missing secrets, VPC connector issue, database unreachable.

4. SSL certificate stuck in PROVISIONING

Verify DNS records resolve correctly. Check for CAA records blocking Google's CA (pki.goog). Wait up to 24 hours. Ensure domain in ssl_certificate_domains matches DNS record exactly.

5. 'Permission denied' errors

Verify service account roles: gcloud projects get-iam-policy $PROJECT_ID --flatten='bindings[].members' --filter='bindings.members:serviceAccount:*api-sa*'

6. Database connection refused from Cloud Run

Verify VPC connector is healthy, private services access is configured, and Cloud SQL instance has a private IP.

7. Redis connection timeout

Verify Memorystore instance is in the same VPC, Redis AUTH password matches Secret Manager, and VPC connector egress is PRIVATE_RANGES_ONLY.

8. Frontend returns 403 Forbidden

Verify bucket-level public access is enabled, allUsers has roles/storage.objectViewer, and index.html exists in the bucket root.

9. Celery worker not processing tasks

Check worker logs, verify Redis connectivity (Celery broker), and confirm the worker is running queues: default,imports,reports,payroll.

Useful Debugging Commands

```
# View Cloud Run service status
gcloud run services describe SERVICE --region=REGION --project=PROJECT_ID
# Stream logs in real time
gcloud run services logs tail SERVICE --region=REGION --project=PROJECT_ID
# View Cloud SQL instance
```

```
gcloud sql instances describe INSTANCE --project=PROJECT_ID
# View Redis instance
gcloud redis instances describe INSTANCE --region=REGION --project=PROJECT_ID
# Check Terraform state
cd infra && make state-list ENV=staging
```

12. Post-Deployment Checklist

After completing the first deployment, verify every item below passes.

#	Check	Description
1	API Health	API responds at /healthz/ with HTTP 200
2	API Readiness	API responds at /readyz/ with HTTP 200 (database + Redis connected)
3	Frontend	Frontend loads at the custom domain
4	SSL	SSL certificate is ACTIVE (not PROVISIONING)
5	Auth	Admin can log in to the Django admin panel
6	Workers	Celery worker is processing tasks
7	Monitoring	Monitoring dashboard shows data
8	Uptime	Uptime checks are passing in Cloud Monitoring
9	Alerts	Alert notification channels are verified (email + Slack)
10	Maintenance	Scheduled maintenance workflow is enabled in GitHub Actions
11	WIF	GitHub Actions can authenticate via Workload Identity Federation
12	Seed Data	Seed data loaded (roles, permissions, tax brackets, bank codes)
13	Backups	Database backups are running (check Cloud SQL backup schedule)
14	CDN	CDN is serving cached assets (check x-cache header in dev tools)
15	WAF	Cloud Armor WAF is blocking test SQL injection attempts

---

This document is maintained by the DevOps & Infrastructure Team. Replace all {project}, {domain}, and {org} placeholders with your client-specific values.