Deploying Budget App to Google Cloud Platform (GCP)

Overview

This guide walks you through deploying a full-stack budget application to GCP, from development to production. We'll cover multiple deployment strategies based on your app architecture.

Architecture Options

Option 1: Cloud Run (Recommended for Starting)

Best for: Serverless, containerized apps with automatic scaling

- **Frontend**: Next.js app
- **Backend**: Next.js API routes (same container)
- **Database**: Cloud SQL (PostgreSQL)
- **Cost**: Pay per request, scales to zero

Option 2: App Engine

Best for: Simpler deployment, less configuration

- **Frontend**: Static files or Next.js
- **Backend**: Node.js standard environment
- **Database**: Cloud SQL
- **Cost**: Always-on instances

Option 3: Cloud Run + Cloud Storage (Static Frontend)

Best for: Separation of frontend/backend

- **Frontend**: Static files on Cloud Storage + CDN
- **Backend**: Node.js API on Cloud Run
- **Database**: Cloud SQL
- **Cost**: Very low for frontend, pay per request for backend

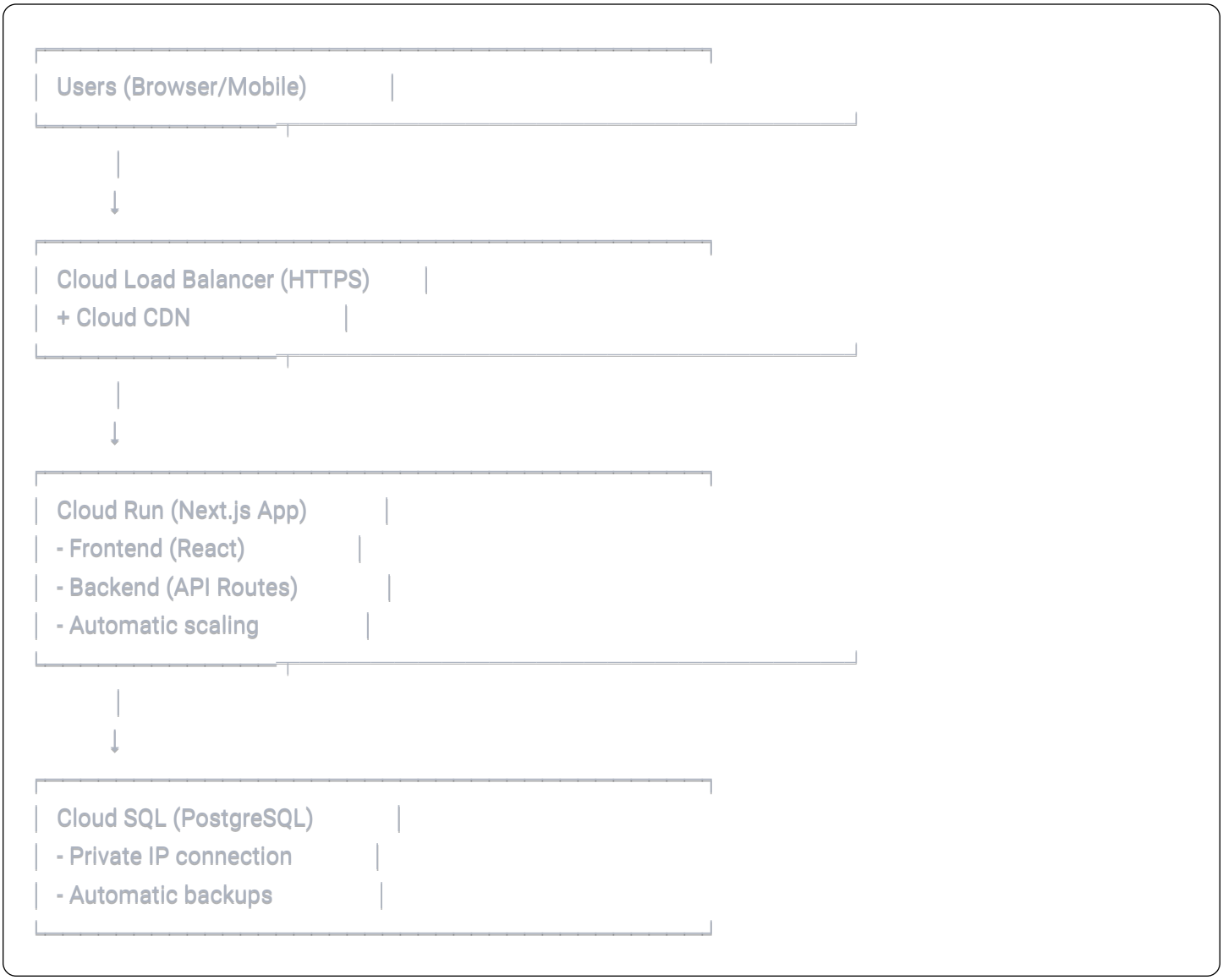
Option 4: GKE (Kubernetes)

Best for: Complex apps, microservices, large scale

- **Frontend**: Kubernetes deployment
- **Backend**: Kubernetes service
- **Database**: Cloud SQL or self-managed
- **Cost**: Higher, always-on cluster

Recommended Stack: Cloud Run + Cloud SQL

This guide focuses on **Cloud Run** as it's the best balance of simplicity, cost, and scalability.



Prerequisites

1. GCP Account Setup

```bash
# Install gcloud CLI
# macOS
brew install google-cloud-sdk

# Linux
curl https://sdk.cloud.google.com | bash

# Initialize and authenticate
gcloud init
gcloud auth login
```

2. Create GCP Project

```bash
# Create new project
gcloud projects create budget-app-prod --name="Budget App"

# Set as active project
gcloud config set project budget-app-prod

# Enable billing (required)
# Go to: https://console.cloud.google.com/billing
```

3. Enable Required APIs

```bash
# Enable all necessary APIs
gcloud services enable \
  run.googleapis.com \
  sql-component.googleapis.com \
  sqladmin.googleapis.com \
  cloudbuild.googleapis.com \
  secretmanager.googleapis.com \
  compute.googleapis.com \
  vpcaccess.googleapis.com
```

Step-by-Step Deployment

Phase 1: Prepare Your Application

1.1 Project Structure

```
budget-app/
├── src/
│   ├── app/                # Next.js 14 App Router
│   │   ├── api/            # API routes
│   │   ├── dashboard/      # Dashboard page
│   │   └── page.tsx        # Home page
│   ├── components/         # React components
│   ├── lib/                # Utilities
│   │   ├── db.ts           # Database connection
│   │   └── auth.ts         # Authentication
├── public/                 # Static assets
├── prisma/                 # Database schema
│   └── schema.prisma
├── Dockerfile              # Container config
├── .dockerignore
├── package.json
├── next.config.js
└── .env.example
```

1.2 Create Dockerfile

dockerfile

```
# Dockerfile
FROM node:18-alpine AS base

# Install dependencies only when needed
FROM base AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

COPY package.json package-lock.json* ./
RUN npm ci

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Generate Prisma Client
RUN npx prisma generate

# Build Next.js
ENV NEXT_TELEMETRY_DISABLED 1
RUN npm run build

# Production image
FROM base AS runner
WORKDIR /app

ENV NODE_ENV production
ENV NEXT_TELEMETRY_DISABLED 1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 8080
ENV PORT 8080
ENV HOSTNAME "0.0.0.0"

CMD ["node", "server.js"]
```

```
CMD ["node", "server.js"]
```

1.3 Create .dockerignore

```
# .dockerignore
node_modules
npm-debug.log
.next
.git
.gitignore
README.md
.env
.env.local
.DS_Store
```

1.4 Update next.config.js

javascript

```javascript
// next.config.js
/** @type {import('next').NextConfig} */
const nextConfig = {
  output: 'standalone', // Important for Cloud Run
  experimental: {
    serverComponentsExternalPackages: ['@prisma/client']
  }
}

module.exports = nextConfig
```

Phase 2: Set Up Database

2.1 Create Cloud SQL Instance

bash

```bash
# Set variables
export PROJECT_ID="budget-app-prod"
export REGION="us-central1"
export INSTANCE_NAME="budget-db"
export DB_NAME="budgetapp"
export DB_PASSWORD="$(openssl rand -base64 32)"

# Create Cloud SQL instance (PostgreSQL)
gcloud sql instances create $INSTANCE_NAME \
  --database-version=POSTGRES_15 \
  --tier=db-f1-micro \
  --region=$REGION \
  --network=default \
  --no-assign-ip \
  --root-password=$DB_PASSWORD

# Create database
gcloud sql databases create $DB_NAME \
  --instance=$INSTANCE_NAME

# Save password to Secret Manager
echo -n $DB_PASSWORD | gcloud secrets create db-password \
  --data-file=- \
  --replication-policy="automatic"

echo "Database password saved! Connection string:"
echo "postgresql://postgres:$DB_PASSWORD@localhost/budgetapp"
```

2.2 Set Up Cloud SQL Proxy (for local development)

bash

```bash
# Download Cloud SQL Proxy
curl -o cloud-sql-proxy https://storage.googleapis.com/cloud-sql-connectors/cloud-sql-proxy/v2.8.2/cloud-sql-pro
chmod +x cloud-sql-proxy

# Get connection name
export CONNECTION_NAME=$(gcloud sql instances describe $INSTANCE_NAME \
  --format='value(connectionName)')

# Start proxy
./cloud-sql-proxy $CONNECTION_NAME
```

2.3 Run Database Migrations

```bash
# Create .env file
cat > .env << EOF
DATABASE_URL="postgresql://postgres:$DB_PASSWORD@localhost:5432/$DB_NAME"
NEXTAUTH_SECRET="$(openssl rand -base64 32)"
NEXTAUTH_URL="http://localhost:3000"
EOF

# Run Prisma migrations
npx prisma migrate deploy
npx prisma generate
```

Phase 3: Create VPC Connector (for Cloud Run to access Cloud SQL)

```bash
# Create VPC Access connector
gcloud compute networks vpc-access connectors create budget-connector \
  --network=default \
  --region=$REGION \
  --range=10.8.0.0/28

# This allows Cloud Run to communicate with Cloud SQL via private IP
```

Phase 4: Build and Deploy to Cloud Run

4.1 Build Container Image

```bash
# Build with Cloud Build
gcloud builds submit --tag gcr.io/$PROJECT_ID/budget-app

# Or build locally and push
docker build -t gcr.io/$PROJECT_ID/budget-app .
docker push gcr.io/$PROJECT_ID/budget-app
```

4.2 Deploy to Cloud Run

bash

```
# Get Cloud SQL connection name
export CONNECTION_NAME=$(gcloud sql instances describe $INSTANCE_NAME \
  --format='value(connectionName)')

# Deploy to Cloud Run
gcloud run deploy budget-app \
  --image gcr.io/$PROJECT_ID/budget-app \
  --platform managed \
  --region $REGION \
  --allow-unauthenticated \
  --set-env-vars "DATABASE_URL=postgresql://postgres:$DB_PASSWORD@localhost:5432/$DB_NAME?host=/clou
  --set-env-vars "NEXTAUTH_URL=https://budget-app-RANDOM.run.app" \
  --set-secrets "NEXTAUTH_SECRET=nextauth-secret:latest" \
  --add-cloudsql-instances $CONNECTION_NAME \
  --vpc-connector budget-connector \
  --min-instances 0 \
  --max-instances 10 \
  --memory 512Mi \
  --cpu 1 \
  --timeout 300

# Get the deployed URL
gcloud run services describe budget-app \
  --region $REGION \
  --format 'value(status.url)'
```

4.3 Create Secret for NextAuth

bash

```
# Create NextAuth secret
echo -n "$(openssl rand -base64 32)" | gcloud secrets create nextauth-secret \
  --data-file=- \
  --replication-policy="automatic"

# Grant Cloud Run access to secret
gcloud secrets add-iam-policy-binding nextauth-secret \
  --member="serviceAccount:$(gcloud run services describe budget-app \
  --region=$REGION --format='value(spec.template.spec.serviceAccountName)')" \
  --role="roles/secretmanager.secretAccessor"
```

Phase 5: Set Up Custom Domain (Optional)

5.1 Map Custom Domain

```bash
# Add custom domain
gcloud run domain-mappings create \
  --service budget-app \
  --domain budget.yourdomain.com \
  --region $REGION

# Get DNS records to add to your domain registrar
gcloud run domain-mappings describe \
  --domain budget.yourdomain.com \
  --region $REGION
```

5.2 Configure DNS

Add these records to your DNS provider:

```
Type: CNAME
Name: budget
Value: ghs.googlehosted.com
```

Phase 6: Set Up CI/CD with Cloud Build

6.1 Create cloudbuild.yaml

yaml

```yaml
# cloudbuild.yaml
steps:
  # Build the container image
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'gcr.io/$PROJECT_ID/budget-app:$COMMIT_SHA', '.']

  # Push to Container Registry
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'gcr.io/$PROJECT_ID/budget-app:$COMMIT_SHA']

  # Deploy to Cloud Run
  - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
    entrypoint: gcloud
    args:
      - 'run'
      - 'deploy'
      - 'budget-app'
      - '--image=gcr.io/$PROJECT_ID/budget-app:$COMMIT_SHA'
      - '--region=us-central1'
      - '--platform=managed'
      - '--allow-unauthenticated'

images:
  - 'gcr.io/$PROJECT_ID/budget-app:$COMMIT_SHA'

options:
  logging: CLOUD_LOGGING_ONLY
```

6.2 Connect GitHub Repository

bash

```bash
# Install Cloud Build GitHub app
# Visit: https://github.com/apps/google-cloud-build

# Create trigger
gcloud builds triggers create github \
  --repo-name=budget-app \
  --repo-owner=YOUR_GITHUB_USERNAME \
  --branch-pattern="^main$" \
  --build-config=cloudbuild.yaml
```

Now every push to `main` branch automatically deploys!

Phase 7: Monitoring and Logging

7.1 View Logs

```bash
# View Cloud Run logs
gcloud run services logs tail budget-app --region $REGION

# View in console
# https://console.cloud.google.com/logs
```

7.2 Set Up Monitoring

```bash
# Create uptime check
gcloud monitoring uptime-check-configs create budget-app-uptime \
  --display-name="Budget App Uptime" \
  --http-check-path="/" \
  --http-check-port=443 \
  --monitored-resource-type="uptime_url" \
  --monitored-resource="host=$(gcloud run services describe budget-app \
    --region=$REGION --format='value(status.url)' | sed 's|https://||')"
```

7.3 Set Up Alerts

```bash
# Create alert policy
gcloud alpha monitoring policies create \
  --notification-channels=CHANNEL_ID \
  --display-name="Budget App High Error Rate" \
  --condition-display-name="Error Rate > 5%" \
  --condition-threshold-value=0.05
```

Cost Optimization

Current Setup Costs (Monthly Estimates)

```
Cloud Run:
 - First 2 million requests: FREE
 - Additional requests: $0.40 per million
 - CPU time: $0.00002400 per vCPU-second
 - Memory: $0.00000250 per GiB-second

Cloud SQL (db-f1-micro):
 - Instance: ~$7/month
 - Storage: $0.17/GB/month
 - Network egress: $0.12/GB

Cloud Build:
 - First 120 build-minutes: FREE
 - Additional: $0.003 per build-minute

Estimated Total: $10-30/month (low traffic)
```

Optimization Tips

bash

```bash
# 1. Use minimum instances only for production
gcloud run services update budget-app \
  --min-instances 0 \
  --region $REGION

# 2. Set request timeout
gcloud run services update budget-app \
  --timeout 60 \
  --region $REGION

# 3. Optimize memory allocation
gcloud run services update budget-app \
  --memory 256Mi \
  --region $REGION

# 4. Enable Cloud CDN for static assets
gcloud compute backend-services update budget-app-backend \
  --enable-cdn

# 5. Use Cloud SQL connection pooling
# In your DATABASE_URL:
# postgresql://user:pass@localhost:5432/db?connection_limit=5
```

Security Best Practices

1. Use Secret Manager for Sensitive Data

```bash
# Never hardcode secrets in environment variables
# Always use Secret Manager

# Create secrets
echo -n "production-secret" | gcloud secrets create api-key --data-file=-

# Reference in Cloud Run
gcloud run services update budget-app \
  --set-secrets "API_KEY=api-key:latest"
```

2. Enable Cloud Armor (DDoS Protection)

```bash
# Create security policy
gcloud compute security-policies create budget-app-policy \
  --description "Budget App Security Policy"

# Add rate limiting rule
gcloud compute security-policies rules create 1000 \
  --security-policy budget-app-policy \
  --expression "true" \
  --action "rate-based-ban" \
  --rate-limit-threshold-count 100 \
  --rate-limit-threshold-interval-sec 60
```

3. Restrict Cloud Run Access

```bash
# Require authentication
gcloud run services update budget-app \
  --no-allow-unauthenticated \
  --region $REGION

# Or use Cloud IAP (Identity-Aware Proxy)
gcloud iap web enable --resource-type=backend-services
```

4. Enable VPC Service Controls

```bash
# Create service perimeter
gcloud access-context-manager perimeters create budget-app-perimeter \
  --title="Budget App Perimeter" \
  --resources=projects/$PROJECT_ID \
  --restricted-services=run.googleapis.com,sqladmin.googleapis.com
```

Troubleshooting

Issue: Cloud Run can't connect to Cloud SQL

```bash
# Check VPC connector is working
gcloud compute networks vpc-access connectors describe budget-connector \
  --region $REGION

# Verify Cloud SQL connection name
gcloud sql instances describe $INSTANCE_NAME

# Check IAM permissions
gcloud projects get-iam-policy $PROJECT_ID \
  --flatten="bindings[].members" \
  --filter="bindings.members:serviceAccount:"
```

Issue: Build fails

```bash
# Check Cloud Build logs
gcloud builds list --limit 5

# View specific build
gcloud builds log BUILD_ID

# Common fixes:
# - Check Dockerfile syntax
# - Verify all dependencies in package.json
# - Check memory limits in Cloud Build
```

Issue: High costs

```bash
# Check Cloud Run metrics
gcloud run services describe budget-app \
  --region $REGION \
  --format="value(status.traffic)"

# Check Cloud SQL metrics
gcloud sql operations list --instance $INSTANCE_NAME

# Analyze costs
gcloud billing accounts list
# Then visit Cloud Console > Billing > Reports
```

Alternative: Deploy Frontend Separately

Static Frontend on Cloud Storage

```bash
# Build Next.js as static
npm run build
npm run export

# Create bucket
gsutil mb gs://budget-app-frontend

# Upload files
gsutil -m cp -r out/* gs://budget-app-frontend

# Make public
gsutil iam ch allUsers:objectViewer gs://budget-app-frontend

# Enable website
gsutil web set -m index.html -e 404.html gs://budget-app-frontend

# Set up Cloud CDN
gcloud compute backend-buckets create budget-frontend \
  --gcs-bucket-name=budget-app-frontend \
  --enable-cdn
```

Terraform Alternative

For infrastructure as code:

hcl

```hcl
# main.tf
terraform {
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 5.0"
    }
  }
}

provider "google" {
  project = "budget-app-prod"
  region  = "us-central1"
}

resource "google_cloud_run_service" "budget_app" {
  name     = "budget-app"
  location = "us-central1"

  template {
    spec {
      containers {
        image = "gcr.io/budget-app-prod/budget-app:latest"

        env {
          name  = "DATABASE_URL"
          value = google_sql_database_instance.postgres.connection_name
        }
      }
    }
  }

  traffic {
    percent         = 100
    latest_revision = true
  }
}

resource "google_sql_database_instance" "postgres" {
  name             = "budget-db"
  database_version = "POSTGRES_15"
  region           = "us-central1"

  settings {
```

```
  tier = "db-f1-micro"
 }
}
```

Summary: Quick Deploy Checklist

- [ ] Create GCP project
- [ ] Enable APIs
- [ ] Create Cloud SQL instance
- [ ] Set up VPC connector
- [ ] Create Dockerfile
- [ ] Build container image
- [ ] Deploy to Cloud Run
- [ ] Set up secrets
- [ ] Configure domain (optional)
- [ ] Set up CI/CD
- [ ] Configure monitoring
- [ ] Test application
- [ ] Optimize costs

Next Steps

1. **Set up development/staging environments**
2. **Implement proper error tracking** (Sentry, Cloud Error Reporting)
3. **Add performance monitoring** (Cloud Trace, Cloud Profiler)
4. **Set up backup strategy** (Cloud SQL automated backups)
5. **Implement rate limiting** (Cloud Armor)
6. **Add authentication** (Firebase Auth, Auth0, NextAuth)
7. **Configure CDN** for static assets
8. **Set up disaster recovery** plan

Resources

- Cloud Run Documentation
- Cloud SQL Documentation
- Next.js Deployment
- GCP Cost Calculator
- GCP Free Tier

Estimated Time: 2-3 hours for initial deployment **Estimated Cost**: $10-30/month for low-traffic production app