Allice Deployment Guide for Google Cloud Run

This guide provides comprehensive instructions for deploying Allice to Google Cloud Run as a production-ready, scalable service.

📋 Table of Contents

🎯 Overview

This deployment setup includes:
- **Allice**: Fully autonomous AI agent system
- **PostgreSQL**: Database for session management and data persistence
- **Google Cloud Run**: Serverless container platform
- **Multi-model support**: Ollama (local), Cloud APIs (OpenAI, Anthropic, etc.)
- **100 concurrent instances**: Support for 6 agents per instance

✅ Prerequisites

Required Tools

```
# Google Cloud SDK
gcloud --version

# Docker
docker --version

# Git
git --version
```

Google Cloud Setup

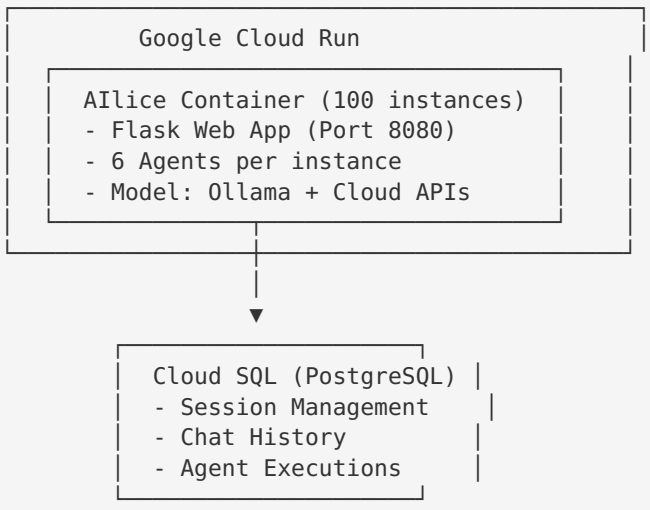
```
# 1. Set your project
export PROJECT_ID="eighth-beacon-479707-c3"
export REGION="us-central1"
export SERVICE_NAME="viralspark-ailice"

gcloud config set project $PROJECT_ID

# 2. Enable required APIs
gcloud services enable \
    run.googleapis.com \
    cloudbuild.googleapis.com \
    containerregistry.googleapis.com \
    sqladmin.googleapis.com \
    compute.googleapis.com \
    artifactregistry.googleapis.com

# 3. Authenticate
gcloud auth login
gcloud auth configure-docker
```

🏗️ Architecture



🔧 Local Setup

1. Clone and Install Dependencies

```
# Clone repository (already done)
cd /home/ubuntu/viralspark_ailice

# Create virtual environment
python3 -m venv venv
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Install AIlice in development mode
pip install -e .
```

2. Local Testing

```
# Set environment variables
export DATABASE_URL="postgresql://ailice:ailice@localhost:5432/ailice"
export PORT=8080

# Run locally
python3 cloud_run_app.py
```

Test in browser: http://localhost:8080

🗄️ Database Setup

Option 1: Cloud SQL (Recommended for Production)

```
# 1. Create Cloud SQL instance
gcloud sql instances create ailice-db \
    --database-version=POSTGRES_14 \
    --tier=db-f1-micro \
    --region=$REGION \
    --root-password=YOUR_SECURE_PASSWORD \
    --database-flags=max_connections=100

# 2. Create database
gcloud sql databases create ailice \
    --instance=ailice-db

# 3. Create user
gcloud sql users create ailice \
    --instance=ailice-db \
    --password=YOUR_SECURE_PASSWORD

# 4. Get connection name
export INSTANCE_CONNECTION_NAME=$(gcloud sql instances describe ailice-db \
    --format='value(connectionName)')

echo "Instance Connection: $INSTANCE_CONNECTION_NAME"
```

Option 2: Local PostgreSQL (Development)

```
# Install PostgreSQL
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib

# Create database and user
sudo -u postgres psql << EOF
CREATE DATABASE ailice;
CREATE USER ailice WITH PASSWORD 'ailice';
GRANT ALL PRIVILEGES ON DATABASE ailice TO ailice;
EOF
```

⚙️ Configuration

1. Update config.json

The user's config.json is already integrated at `/home/ubuntu/viralspark_ailice/config.json`.

Key configurations to review:
- **Model endpoints**: Ensure Ollama URL is correct for your setup
- **API keys**: Set in environment variables (see below)
- **Services**: Configure MCP service endpoints

2. Environment Variables

Create `.env` file (do not commit to git):

```
# Copy from example
cp .env.example .env

# Edit with your values
nano .env
```

Required environment variables:

```
# Database
DATABASE_URL=postgresql://user:password@/dbname?host=/cloudsql/PROJECT:REGION:INSTANCE

# Models (if using cloud APIs)
OPENAI_API_KEY=sk-...
ANTHROPIC_API_KEY=sk-ant-...
GROQ_API_KEY=gsk_...

# Server
PORT=8080
AILICE_HOST=0.0.0.0
LOG_LEVEL=INFO

# GCP
GCP_PROJECT_ID=eighth-beacon-479707-c3
GCP_REGION=us-central1
```

🚀 Deployment to Google Cloud Run

Method 1: Using gcloud (Recommended)

```
# 1. Set variables
export PROJECT_ID="eighth-beacon-479707-c3"
export REGION="us-central1"
export SERVICE_NAME="viralspark-ailice"
export IMAGE_NAME="gcr.io/$PROJECT_ID/$SERVICE_NAME"

# 2. Build and push Docker image
gcloud builds submit --tag $IMAGE_NAME

# 3. Deploy to Cloud Run
gcloud run deploy $SERVICE_NAME \
    --image=$IMAGE_NAME \
    --region=$REGION \
    --platform=managed \
    --allow-unauthenticated \
    --memory=4Gi \
    --cpu=2 \
    --timeout=3600 \
    --max-instances=100 \
    --min-instances=1 \
    --concurrency=80 \
    --port=8080 \
    --add-cloudsql-instances=$INSTANCE_CONNECTION_NAME \
    --set-env-vars="DATABASE_URL=postgresql://ailice:PASSWORD@/ailice?host=/cloudsql/$
INSTANCE_CONNECTION_NAME" \
    --set-env-vars="AILICE_HOST=0.0.0.0" \
    --set-env-vars="LOG_LEVEL=INFO"

# 4. Get service URL
gcloud run services describe $SERVICE_NAME \
    --region=$REGION \
    --format='value(status.url)'
```

Method 2: Using Docker + Artifact Registry

```
# 1. Create Artifact Registry repository
gcloud artifacts repositories create ailice-repo \
    --repository-format=docker \
    --location=$REGION

# 2. Configure Docker authentication
gcloud auth configure-docker $REGION-docker.pkg.dev

# 3. Build and tag image
export IMAGE_NAME="$REGION-docker.pkg.dev/$PROJECT_ID/ailice-repo/ailice:latest"
docker build -t $IMAGE_NAME .

# 4. Push to Artifact Registry
docker push $IMAGE_NAME

# 5. Deploy
gcloud run deploy $SERVICE_NAME \
    --image=$IMAGE_NAME \
    --region=$REGION \
    --platform=managed \
    [... same flags as Method 1 ...]
```

Setting Secrets (Secure API Keys)

```
# 1. Create secrets
echo -n "your-openai-key" | gcloud secrets create openai-api-key --data-file=-
echo -n "your-anthropic-key" | gcloud secrets create anthropic-api-key --data-file=-

# 2. Deploy with secrets
gcloud run deploy $SERVICE_NAME \
    --image=$IMAGE_NAME \
    --region=$REGION \
    --update-secrets=OPENAI_API_KEY=openai-api-key:latest \
    --update-secrets=ANTHROPIC_API_KEY=anthropic-api-key:latest \
    [... other flags ...]
```

📊 Post-Deployment

1. Test the Deployment

```
# Get service URL
SERVICE_URL=$(gcloud run services describe $SERVICE_NAME \
    --region=$REGION \
    --format='value(status.url)')

# Test health
curl $SERVICE_URL

# Test API
curl -X POST $SERVICE_URL/message \
    -F "message=Hello AIlice!"
```

2. Configure Custom Domain (Optional)

```
# Map custom domain
gcloud run domain-mappings create \
    --service=$SERVICE_NAME \
    --domain=ailice.yourdomain.com \
    --region=$REGION
```

3. Set up HTTPS/SSL

Cloud Run automatically provides HTTPS with managed SSL certificates.

📈 Scaling and Performance

Automatic Scaling Configuration

```
gcloud run services update $SERVICE_NAME \
    --region=$REGION \
    --min-instances=5 \
    --max-instances=100 \
    --concurrency=80 \
    --cpu-throttling \
    --memory=4Gi \
    --cpu=2
```

Performance Considerations

1. **Instance Scaling**:
   - Min instances: 1-5 (for quick response)
   - Max instances: 100 (as per requirements)
   - Concurrency: 80 requests per instance

2. **Resource Allocation**:
   - Memory: 4Gi (adjustable based on model requirements)
   - CPU: 2 vCPU
   - Timeout: 3600s (1 hour for long-running agents)

3. **Database Connection Pooling**:
   - Configured in `ADatabase.py`
   - Pool size: 10
   - Max overflow: 20

📊 Monitoring and Logging

View Logs

```
# Real-time logs
gcloud run services logs tail $SERVICE_NAME \
    --region=$REGION

# Filter by severity
gcloud run services logs read $SERVICE_NAME \
    --region=$REGION \
    --filter="severity>=ERROR"

# View in Cloud Console
echo "https://console.cloud.google.com/run/detail/$REGION/$SERVICE_NAME/logs"
```

Set up Monitoring

```
# Create uptime check
gcloud monitoring uptime create $SERVICE_NAME \
    --resource-type=uptime-url \
    --host=$SERVICE_URL

# View metrics
echo "https://console.cloud.google.com/run/detail/$REGION/$SERVICE_NAME/metrics"
```

Key Metrics to Monitor

1. **Request Count**: Total requests served
2. **Request Latency**: Response time
3. **Container Instances**: Active instances
4. **Memory Utilization**: Memory usage
5. **CPU Utilization**: CPU usage
6. **Error Rate**: 4xx and 5xx errors

🔧 Troubleshooting

Common Issues

1. Container fails to start

```
# Check logs
gcloud run services logs tail $SERVICE_NAME --region=$REGION

# Common causes:
# - Missing environment variables
# - Database connection issues
# - Port configuration (must use PORT env var)
```

2. Database connection errors

```
# Verify Cloud SQL instance
gcloud sql instances describe ailice-db

# Test connection
gcloud sql connect ailice-db --user=ailice

# Check Cloud Run service account permissions
gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member="serviceAccount:SERVICE_ACCOUNT" \
    --role="roles/cloudsql.client"
```

3. High latency or timeouts

```
# Increase timeout
gcloud run services update $SERVICE_NAME \
    --timeout=3600 \
    --region=$REGION

# Increase resources
gcloud run services update $SERVICE_NAME \
    --memory=8Gi \
    --cpu=4 \
    --region=$REGION
```

4. Model API errors

Check environment variables:

```
gcloud run services describe $SERVICE_NAME \
    --region=$REGION \
    --format="value(spec.template.spec.containers[0].env)"
```

Debug Mode

```
# Deploy with debug logging
gcloud run deploy $SERVICE_NAME \
    --image=$IMAGE_NAME \
    --set-env-vars="LOG_LEVEL=DEBUG" \
    --region=$REGION
```

🔄 Updates and Maintenance

Deploy New Version

```
# Build new image
gcloud builds submit --tag $IMAGE_NAME

# Deploy update (zero-downtime)
gcloud run deploy $SERVICE_NAME \
    --image=$IMAGE_NAME \
    --region=$REGION
```

Rollback

```
# List revisions
gcloud run revisions list \
    --service=$SERVICE_NAME \
    --region=$REGION

# Rollback to previous revision
gcloud run services update-traffic $SERVICE_NAME \
    --to-revisions=REVISION_NAME=100 \
    --region=$REGION
```

Database Migrations

```
# Connect to Cloud SQL
gcloud sql connect ailice-db --user=ailice

# Or use Cloud SQL Proxy
cloud_sql_proxy -instances=$INSTANCE_CONNECTION_NAME=tcp:5432

# Run migrations (if using Alembic)
alembic upgrade head
```

💰 Cost Optimization

1. **Adjust min instances**: Set to 0 during low-traffic periods
2. **Right-size resources**: Start with 2Gi/1CPU, scale as needed
3. **Use Cloud SQL shared-core instances**: For development/testing
4. **Enable Cloud CDN**: For static assets
5. **Set appropriate timeouts**: Avoid long-running idle connections

Cost Estimates

- Cloud Run: ~$0.00002400 per vCPU-second, ~$0.00000250 per GiB-second
- Cloud SQL: db-f1-micro ~$8/month, db-n1-standard-1 ~$50/month
- Networking: First 1GB free, then $0.12/GB

📚 Additional Resources

- Allice Documentation (https://github.com/myshell-ai/Allice)
- Google Cloud Run Docs (https://cloud.google.com/run/docs)
- Cloud SQL for PostgreSQL (https://cloud.google.com/sql/docs/postgres)
- Best Practices for Cloud Run (https://cloud.google.com/run/docs/best-practices)

🆘 Support

For issues specific to:
- **Allice**: See original repository
- **Deployment**: Check Cloud Run documentation
- **Database**: Review Cloud SQL troubleshooting guide

Quick Deploy Script

Save this as `deploy.sh`:

```
#!/bin/bash
set -e

# Configuration
PROJECT_ID="eighth-beacon-479707-c3"
REGION="us-central1"
SERVICE_NAME="viralspark-ailice"
IMAGE_NAME="gcr.io/$PROJECT_ID/$SERVICE_NAME"

# Build and deploy
echo "Building Docker image..."
gcloud builds submit --tag $IMAGE_NAME

echo "Deploying to Cloud Run..."
gcloud run deploy $SERVICE_NAME \
    --image=$IMAGE_NAME \
    --region=$REGION \
    --platform=managed \
    --allow-unauthenticated \
    --memory=4Gi \
    --cpu=2 \
    --timeout=3600 \
    --max-instances=100 \
    --min-instances=1 \
    --concurrency=80 \
    --port=8080

echo "Deployment complete!"
SERVICE_URL=$(gcloud run services describe $SERVICE_NAME \
    --region=$REGION \
    --format='value(status.url)')
echo "Service URL: $SERVICE_URL"
```

Make it executable: `chmod +x deploy.sh`

Run: `./deploy.sh`