Deploy Insurance Agent to Google Cloud Run

Prerequisites

- Google Cloud SDK installed and authenticated (`gcloud init`).
- A Google Cloud project with billing enabled.
- Docker installed locally.
- The repository cloned locally (see workshop guide).
- `GEMINI_API_KEY` ready.

1. Prepare the Backend Docker Image

Create a **Dockerfile** in `backend/` (if not already present):

```
# Use official Python image
FROM python:3.12-slim

# Set working directory
WORKDIR /app

# Install system dependencies (git, build-essential if needed)
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Copy only requirements first for caching
COPY backend/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy the source code
COPY backend/ .

# Expose the port FastAPI will run on
EXPOSE 8080

# Set environment variable for port (Cloud Run expects $PORT)
ENV PORT=8080

# Run the app with Uvicorn
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Build & Push the Image

```
# From the repository root
cd backend
# Build the image (replace PROJECT_ID with your GCP project)
PROJECT_ID=$(gcloud config get-value project)
```

```
IMAGE_NAME=gcr.io/$PROJECT_ID/insurance-agent-backend

docker build -t $IMAGE_NAME .

# Push to Google Container Registry (or Artifact Registry)
docker push $IMAGE_NAME
```

2. Deploy the Backend to Cloud Run

```
# Deploy, allowing unauthenticated access so the frontend can call it
gcloud run deploy insurance-agent-backend \
  --image $IMAGE_NAME \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars GEMINI_API_KEY=$GEMINI_API_KEY
```

- Note the URL that Cloud Run returns, e.g. `https://insurance-agent-backend-xxxxx-uc.a.run.app`.
- The service will listen on the `$PORT` env var (8080 by default).

3. Prepare the Frontend Docker Image (optional)

You can either serve the built static files from a separate Cloud Run service **or** host them on a CDN (e.g., Firebase Hosting). Below is the Cloud Run option.

Create a **Dockerfile** in `frontend/`:

```
# Build stage
FROM node:20-alpine AS builder
WORKDIR /app
COPY frontend/package*.json ./
RUN npm ci
COPY frontend/ ./
RUN npm run build   # creates ./dist

# Serve stage (using a lightweight web server)
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

Build & Push the Frontend Image

```
cd frontend
IMAGE_NAME=gcr.io/$PROJECT_ID/insurance-agent-frontend
```

```
docker build -t $IMAGE_NAME .

docker push $IMAGE_NAME
```

4. Deploy the Frontend to Cloud Run

```
# The frontend needs to know the backend URL. Pass it as an env var.
BACKEND_URL=$(gcloud run services describe insurance-agent-backend \
  --platform managed --region us-central1 \
  --format='value(status.url)')

gcloud run deploy insurance-agent-frontend \
  --image $IMAGE_NAME \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated \
  --set-env-vars REACT_APP_BACKEND_URL=$BACKEND_URL
```

- The React code should read `process.env.REACT_APP_BACKEND_URL` (Vite automatically injects `import.meta.env.VITE_BACKEND_URL`). If the current `ChatInterface.jsx` uses a hard-coded URL, replace it with the env var:

```
const backendUrl = import.meta.env.VITE_BACKEND_URL || "http://localhost:8001";
fetch(`${backendUrl}/api/chat`, {...})
```

- Re-build the frontend after this change before creating the Docker image.

5. Verify the Deployment

1. Open the frontend Cloud Run URL in a browser.
2. Interact with the chatbot – the request will be proxied to the backend URL you supplied.
3. To inspect backend state, you can call the health endpoint:

```
curl https://<backend-url>/health
```

It returns JSON with session count, etc. 4. If you need to view the in-memory sessions, add a temporary debug endpoint or use Cloud Run logs (`gcloud logs read`).

6. Optional – Single-Service Deployment

If you prefer a single Cloud Run service, you can serve the built frontend as static files from the same container that runs FastAPI. Add the following to the backend Dockerfile **after** installing requirements:

```
# Copy built frontend assets
COPY --from=frontend-builder /app/dist /app/static
# In main.py, mount StaticFiles
from fastapi.staticfiles import StaticFiles
app.mount("/", StaticFiles(directory="static", html=True), name="static")
```

Then rebuild and redeploy only the backend image.

7. Clean-up

When you are done testing, you can delete the services:

```
gcloud run services delete insurance-agent-backend --region us-central1

gcloud run services delete insurance-agent-frontend --region us-central1
```

And optionally remove the images from GCR:

```
gcloud container images delete $IMAGE_NAME --quiet
```

Quick Reference Commands

```
# Build & push backend
docker build -t gcr.io/$PROJECT_ID/insurance-agent-backend backend
docker push gcr.io/$PROJECT_ID/insurance-agent-backend

# Deploy backend
gcloud run deploy insurance-agent-backend \
  --image gcr.io/$PROJECT_ID/insurance-agent-backend \
  --allow-unauthenticated \
  --set-env-vars GEMINI_API_KEY=$GEMINI_API_KEY

# Build & push frontend
docker build -t gcr.io/$PROJECT_ID/insurance-agent-frontend frontend
docker push gcr.io/$PROJECT_ID/insurance-agent-frontend

# Deploy frontend (replace BACKEND_URL with actual URL)
gcloud run deploy insurance-agent-frontend \
  --image gcr.io/$PROJECT_ID/insurance-agent-frontend \
  --allow-unauthenticated \
  --set-env-vars REACT_APP_BACKEND_URL=$BACKEND_URL
```

You now have the Insurance Agent running on Google Cloud Run, accessible via public URLs, with the Gemini API key securely injected as an environment variable.