File Storage on Cloud Run: The Real Story

🤔 Your Question: Is Google Just Trying to Sell Storage?

Short answer: No, this is how ALL containerized platforms work (AWS, Azure, etc.).

Long answer: Cloud Run is based on containers, which are designed to be **stateless** and **immutable**. This isn't a Google limitation - it's a fundamental principle of containerization. Think of it like this:

- **Traditional servers**: Like owning a house - you can store stuff and it stays there
- **Containers**: Like staying in a hotel room - when you check out, everything is cleaned up for the next guest

This design makes containers:
- ✅ Fast to deploy
- ✅ Easy to scale
- ✅ Cheaper (pay per use, not for idle storage)
- ✅ More reliable (crashed containers just restart fresh)

📦 When Are Files Actually Lost?

❌ Files ARE Lost When:

1. **You deploy a new revision** (push code to GitHub)
2. **Cloud Run scales down to zero** (if min instances = 0)
3. **Container crashes and restarts**
4. **Google moves your container** (for maintenance, regional failures, etc.)

✅ Files MIGHT Survive When:

1. **Just accessing the app normally** (same container keeps running)
2. **Min instances = 1** (at least one container always running)

⚠️ The Problem:

Even with min instances = 1, you'll lose files on every code deployment. For a shift reporting system where reports with attachments are critical, this is **unacceptable for production**.

🎯 Solution 1: Cloud Run Volumes + GCS FUSE (Recommended - MINIMAL CODE CHANGES)

What Is This?

Google Cloud Storage (GCS) bucket **mounted as a folder** in your container. Your code keeps writing to `/uploads/`, but behind the scenes, files go to cloud storage.

Why This Is Best:

- ✅ **Almost zero code changes** (maybe 5 lines)
- ✅ **No API calls to learn**
- ✅ **Files persist forever**
- ✅ **Works with multiple instances**
- ✅ **Automatic backups/versioning available**

Cost:

- **Storage**: $0.020 per GB/month (1GB = 2 cents/month)
- **Operations**: Basically free for your use case
- **Example**: 10GB of files = $0.20/month

How To Implement (30 minutes):

Step 1: Create a GCS Bucket

```
# In Google Cloud Console or Cloud Shell
gcloud storage buckets create gs://hotel-shift-log-uploads \
  --location=us-central1 \
  --uniform-bucket-level-access
```

Or via Console: https://console.cloud.google.com/storage → Create Bucket

Step 2: Update Cloud Run Configuration

When deploying (or editing existing service), add a volume:

Via Console:
1. Edit your Cloud Run service
2. Go to **“Container(s), Volumes, Networking, Security”** tab
3. Click **“Volumes” → “Add Volume”**
4. Select **“Cloud Storage bucket”**
5. Bucket: `hotel-shift-log-uploads`
6. Mount path: `/app/nextjs_space/uploads`
7. Mount as read-only: **NO** (unchecked)

Via YAML (if you prefer):

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hotel-shift-log
spec:
  template:
    metadata:
      annotations:
        run.googleapis.com/cloudsql-instances: YOUR_CLOUD_SQL_CONNECTION
        run.googleapis.com/execution-environment: gen2  # Required for volumes
    spec:
      volumes:
      - name: uploads
        csi:
          driver: gcsfuse.run.googleapis.com
          volumeAttributes:
            bucketName: hotel-shift-log-uploads
      containers:
      - image: gcr.io/YOUR_PROJECT/hotel-shift-log
        volumeMounts:
        - name: uploads
          mountPath: /app/nextjs_space/uploads
```

Step 3: Grant Permissions

```bash
# Your service account needs access to the bucket
gcloud storage buckets add-iam-policy-binding gs://hotel-shift-log-uploads \
  --member="serviceAccount:143559442445-compute@developer.gserviceaccount.com" \
  --role="roles/storage.objectAdmin"
```

Or via Console:

1. Go to your GCS bucket
2. Click **“Permissions”**
3. Click **“Grant Access”**
4. Principal: `143559442445-compute@developer.gserviceaccount.com`
5. Role: **“Storage Object Admin”**

Step 4: Verify in Code (Optional - Probably Not Needed)

Your existing file upload code should work as-is! But if you want to verify:

```javascript
// In your upload handler (probably already working)
const uploadDir = path.join(process.cwd(), 'uploads');

// This now writes to GCS automatically!
fs.writeFileSync(path.join(uploadDir, filename), buffer);

// Reading also works the same way
const fileBuffer = fs.readFileSync(path.join(uploadDir, filename));
```

That’s it! Your code doesn’t change at all.

Step 5: Deploy

Just redeploy your Cloud Run service with the new volume configuration. Existing files in local storage will be lost (they were going to be anyway), but all NEW uploads go to GCS.

🎯 Solution 2: Direct Google Cloud Storage API (More Code Changes)

When To Use This:

- You want more control (file metadata, signed URLs, etc.)
- You need to migrate existing local files to cloud
- You want to generate temporary download links

Cost:

Same as Solution 1 (~$0.02/GB/month)

Implementation:

Step 1: Install SDK

```
cd /home/ubuntu/hotel_shift_log/nextjs_space
yarn add @google-cloud/storage
```

Step 2: Create Storage Utility

```
// lib/cloud-storage.ts
import { Storage } from '@google-cloud/storage';

const storage = new Storage();
const bucketName = 'hotel-shift-log-uploads';

export async function uploadFile(
  file: Buffer,
  filename: string,
  folder: string = ''
): Promise<string> {
  const bucket = storage.bucket(bucketName);
  const blob = bucket.file(`${folder}/${filename}`);

  await blob.save(file, {
    metadata: { contentType: 'auto' }
  });

  return `${folder}/${filename}`;
}

export async function downloadFile(filepath: string): Promise<Buffer> {
  const bucket = storage.bucket(bucketName);
  const blob = bucket.file(filepath);
  const [buffer] = await blob.download();
  return buffer;
}

export async function deleteFile(filepath: string): Promise<void> {
  const bucket = storage.bucket(bucketName);
  await bucket.file(filepath).delete();
}
```

Step 3: Update Upload Handlers

Before (local filesystem):

```
// api/files/upload/route.ts
const uploadDir = path.join(process.cwd(), 'uploads');
const filepath = path.join(uploadDir, filename);
await fs.promises.writeFile(filepath, buffer);
```

After (GCS):

```
// api/files/upload/route.ts
import { uploadFile } from '@/lib/cloud-storage';

const filepath = await uploadFile(buffer, filename, 'comments');
// Store filepath in database as before
```

Step 4: Update Download Handlers

Before:

```
const filePath = path.join(process.cwd(), 'uploads', 'comments', filename);
const fileBuffer = await fs.promises.readFile(filePath);
```

After:

```
import { downloadFile } from '@/lib/cloud-storage';

const fileBuffer = await downloadFile(`comments/${filename}`);
```

Effort: 2-3 hours to update all file handlers

🎯 Solution 3: Keep Local Storage (Accept Limitations)

When This Is Acceptable:

- You're still in testing/development
- You'll manually backup files before redeployments
- You deploy rarely (once a month or less)
- File loss is recoverable (can re-upload)

What You Need to Do:

1. **Set min instances to 1** (already done)
2. **Create a backup script** for before deployments
3. **Accept that redeploys = data loss**

Backup Script (Run Before Redeploying):

```
#!/bin/bash
# backup-files.sh

# Download all files from Cloud Run container
gcloud run services describe hotel-shift-log \
  --region=us-central1 \
  --format='value(status.url)' | \
  xargs -I {} curl {}/api/backup-files -o uploads-backup.zip

# Or use cloud shell to access the container
gcloud run services proxy hotel-shift-log --region=us-central1
# Then copy files manually
```

Not recommended for production, but okay for initial testing.

📊 Comparison Table

Solution	Code Changes	Setup Time	Cost/Month	Reliability	Best For
Cloud Run Volumes (GCS FUSE)	None/minimal	30 min	$0.02-1	⭐⭐⭐⭐⭐	**RECOMMENDED**
GCS API Direct	Moderate	2-3 hours	$0.02-1	⭐⭐⭐⭐⭐	Advanced control
Local Storage	None	0 min	$0	⭐⭐	Testing only

🤷 Why Doesn't Google Just Use Persistent Disks by Default?

Actually, they DO offer persistent disks now! But:

1. **Cost**: Persistent disks cost $0.10/GB/month (5x more than GCS)
2. **Speed**: Only one instance can write to a disk at a time (doesn't scale)
3. **Backups**: You have to manage backups manually
4. **Philosophy**: Cloud Run is designed for stateless apps

GCS is the right tool for this use case: cheap, reliable, scales to millions of files.

✅ My Recommendation

Use Solution 1: Cloud Run Volumes with GCS FUSE

Why:
- Takes 30 minutes to set up
- Zero (or near-zero) code changes
- Files never lost again
- Costs pennies per month
- Industry standard approach

When to implement:
- ✅ **Now**: If you're deploying to production soon
- ✅ **Now**: If you can't afford to lose uploaded files
- ⏳ **Later**: If you're just testing with dummy data for a few days

---

🚀 Quick Start: Implementing GCS FUSE Volumes Right Now

If you want me to help you implement this immediately, here's what I can do:

1. ✅ Guide you through creating the GCS bucket (2 minutes via console)
2. ✅ Show you exactly how to configure Cloud Run volumes (5 minutes via console)
3. ✅ Help grant the necessary permissions (1 minute)
4. ✅ Test that uploads work after redeployment

Total time: 15-30 minutes, and you'll never worry about lost files again.

📞 Need Help?

Want me to:
- [ ] Walk you through setting up GCS FUSE volumes right now?
- [ ] Write the code for direct GCS API integration?
- [ ] Create a backup script for local storage?
- [ ] Something else?

Just let me know! 🎉

Bottom Line: No, Google isn't trying to scam you - containerized platforms are just fundamentally different from traditional servers. But the good news is that GCS FUSE makes persistent storage nearly effortless, and it costs almost nothing.