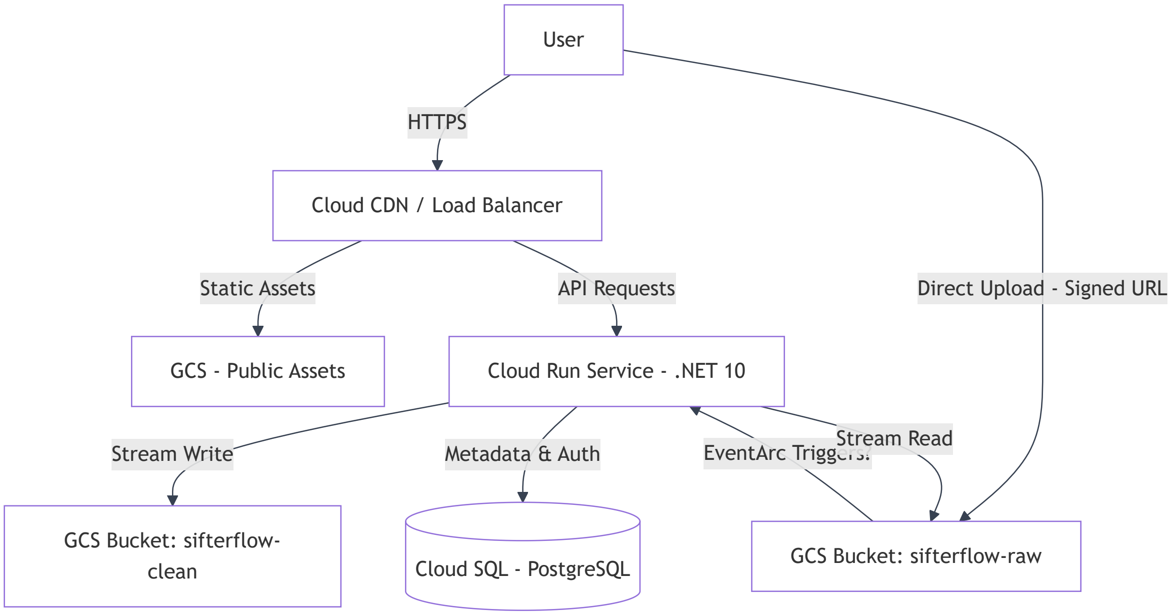
Technical Specification: SifterFlow

🏗️ Architecture Overview

SifterFlow follows a **Serverless, Event-Driven** architecture on Google Cloud Platform (GCP). It prioritizes stateless compute (keeping costs low) and offloads heavy lifting to Streaming I/O.


User
HTTPS
Cloud CDN / Load Balancer
Static Assets
API Requests
Direct Upload - Signed URL
GCS - Public Assets
Cloud Run Service - .NET 10
Stream Write
Metadata & Auth
EventArc Triggers
Stream Read
GCS Bucket: sifterflow-clean
Cloud SQL - PostgreSQL
GCS Bucket: sifterflow-raw

☁️ GCP Services Breakdown

Service	Purpose	Configuration Notes
Cloud Run	Host the .NET 10 Web API.	**Autoscaling:** 0 to 10 instances. **Memory:** 512MB-1GB (Low RAM footprint due to streaming).
Cloud Storage (GCS)	Store temporary and processed CSVs.	**Bucket 1 (`-raw`):** Lifecycle = Delete after 24h (Safety net). Code deletes immediately. **Bucket 2 (`-clean`):** Lifecycle = Delete after 24h.
Cloud SQL	Store User Accounts, Recipes, Usage Logs.	**Engine:** PostgreSQL 16. **Tier:** db-f1-micro (Shared CPU) for MVP. Scale up later.
Artifact Registry	Store Docker Images.	Standard Docker repository.
Secret Manager	Store Connection Strings & Keys.	No hardcoded secrets in env vars.

❓ Why PostgreSQL over SQLite?

While SQLite is fantastic for local development, it is **not suitable** for Cloud Run (serverless containers).

- **The Problem:** Cloud Run containers are ephemeral. If we write to a local `app.db` file, it vanishes when the container spins down.
- **Shared Volumes:** Mounting a shared volume (like Cloud Storage FUSE) for SQLite is slow and prone to locking corruption with multiple users.
- **Recommendation:** Use **PostgreSQL**.
    - **Local Dev:** Run Postgres via Docker (Aspire handles this automatically).
    - **Production:** Cloud SQL (Managed Postgres).

📂 Project Structure (Monorepo)

We will use a standard monorepo structre managed by the Solution file.

```
/
├── SifterFlow.sln              # Solution File
├── SifterFlow.AppHost/         # .NET Aspire Orchestrator (Runs everything locally)
├── SifterFlow.ServiceDefaults/ # Standard health checks, telemetry
│
├── src/
│   ├── SifterFlow.Api/         # Backend: .NET 10 Web API
│   │   ├── Endpoints/          # Minimal APIs (Upload, Process, Auth)
│   │   ├── Services/           # Cloud Services implementation
│   │   └── Dockerfile
│   │
│   ├── SifterFlow.Web/         # Frontend: Svelte 5 (Vite)
│   │   ├── src/
│   │   │   ├── lib/            # Shared UI Components
│   │   │   └── routes/         # Application Pages
│   │   └── Dockerfile
│   │
│   ├── SifterFlow.Core/        # Shared Domain (Enums, Models, Interfaces)
│   │
│   └── SifterFlow.Infrastructure/ # GCP Implementations
│       ├── Storage/            # GcpStorageService.cs
│       └── Data/               # EfCore PostgreSqlContext.cs
│
└── infra/                      # Terraform / Bicep for GCP Provisioning
```

🛠️ Technology Stack Details

- **Language:** C# 12 / .NET 10
- **Orchestration:** .NET Aspire (simplifies running Postgres/Redis/API/Frontend together locally).
- **Frontend:** Svelte 5 + Typescript + TailwindCSS (v4).
- **Data Access:** EF Core 9.
- **CSV Processing:** `Sep` (Fastest C# CSV Parser).

Backend Requirements:

```
// PUT /api/v2/recipes/{recipeId} Request
{
  "name": "Updated Recipe Name",      // optional
  "steps": [...],                      // optional
  "isAutoApply": true                  // optional
}

// PUT /api/v2/recipes/{recipeId} Response
{
  "id": "recipe-uuid",
  "name": "Updated Recipe Name",
  "steps": [...],
  "isAutoApply": true,
  "createdAt": "2026-01-10T12:00:00Z",
  "lastUsedAt": null
}
```

X_2 and Y^2

The quadratic formula is $x = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$

Einstein’s famous equation: $E = mc^2$

Inline math: $\pi \approx 3.14159$

$$\int_{-\infty}^{\infty} e < sup > -x < /sup > 2dx = \sqrt{\pi}$$

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$