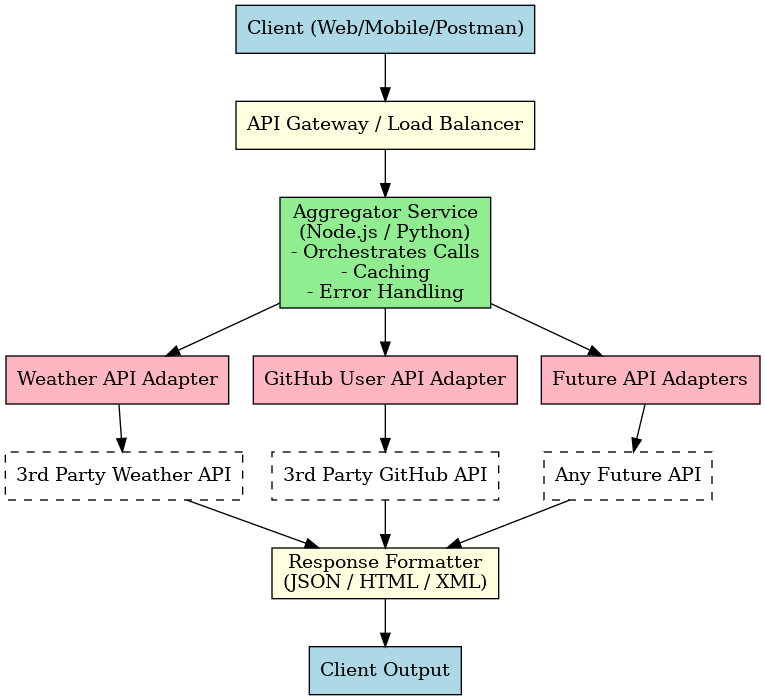
**1. Architecture Diagram (MVP with Scalability in Mind)**

Here’s a sample diagram for an MVP using **Weather API** and **GitHub User API**, but structured to handle many APIs later



**2. Design Decisions & Tradeoffs**

**a) Chosen Tech Stack (Example)**

* **Backend:** Node.js + Express (fast, async, well-supported for API aggregation)
* **Frontend (optional for MVP):** Minimal React/Next.js interface or just Postman testing
* **Cache:** Redis for API response caching (improves performance, avoids hitting rate limits)
* **API Gateway:** Nginx or AWS API Gateway for routing & scalability
* **Data Format:** JSON for interoperability
* **Deployment:** Docker containers for portability; deploy on AWS ECS/Fargate or similar

**b) Tradeoffs**

1. **Monolith vs Microservices**

MVP: Single Aggregator Service (simpler, faster to build)

Tradeoff: Harder to scale individual API connectors independently

1. **Synchronous vs Asynchronous Calls**

MVP: Parallel API calls to reduce latency

Tradeoff: Requires more complex error handling

1. **Direct Client → API Calls vs Aggregator**

MVP: Aggregator controls all calls (better for workflow chaining)

Tradeoff: Adds server load, but improves security and flexibility

**3. Risks & Mitigations**

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| **API Rate Limits** | Service could fail if public APIs block requests | Implement caching (Redis), request throttling, and API key rotation if available |
| **Third-Party API Downtime** | Partial or total feature outage | Fallback responses, retries with exponential backoff, and circuit breaker pattern |
| **Scalability for 50+ APIs** | Bottlenecks as APIs increase | Modular architecture with pluggable adapters, containerized services, auto-scaling |