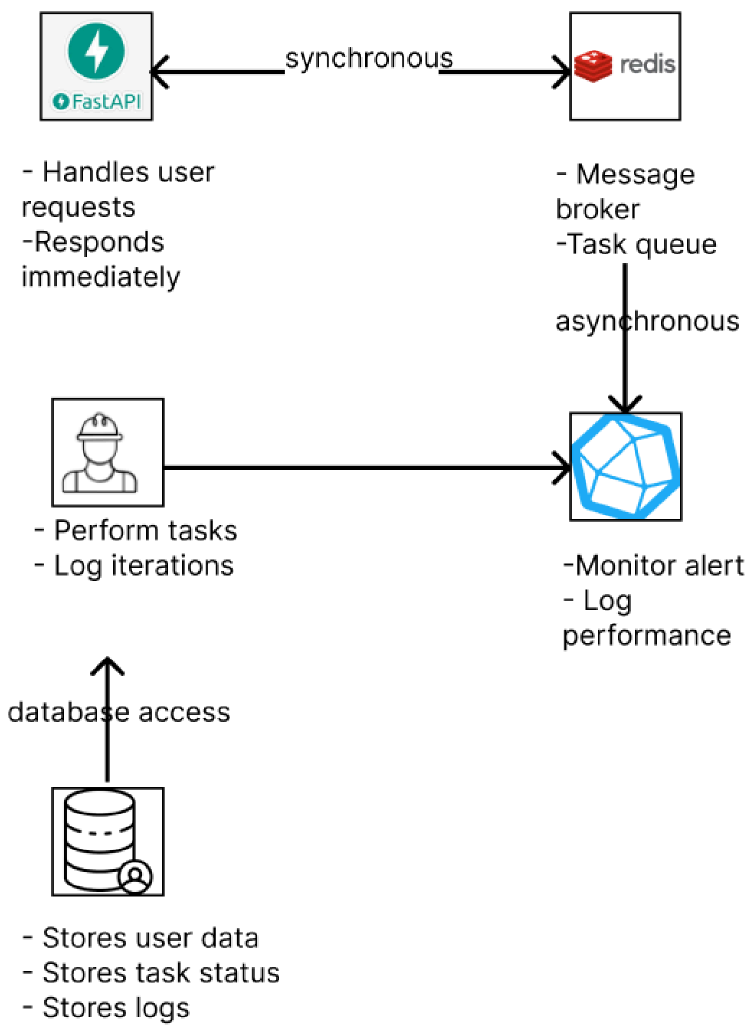
System Architecture

This system handles user requests using FastAPI, Redis, Celery workers, a central database, and a logging/monitoring tool (InfluxDB in my case). The architecture separates synchronous and asynchronous tasks to ensure responsiveness and scalability.



Components

1. **FastAPI**
   - **Role**: Entry point for user requests.
   - **Operations**:
     1) Accepts and parses http requests.
     2) Validates and stores incoming message metadata.

3) Responds to the user **immediately** after queuing the task.

- **Interaction**: Communicates **synchronously** with Redis to enqueue tasks.

2. **Redis (Message broker / Task queue)**
   - **Role**: Acts as an intermediary between the API and workers.
   - **Operations**:
     1) Queues tasks for background processing.

   - **Interaction**: Receives tasks **synchronously** from FastAPI and passes them **asynchronously** to workers using Celery.

3. **Celery Worker**

   - **Role**: Executes tasks such as message processing, iteration logging.
   - **Operations**:
     1) Retrieves tasks from Redis.
     2) Store message content, run analysis.
     3) Logs task iterations and sends performance metrics to the monitoring service.
     4) Updates task status in the database.

   - **Interaction**: Operates **asynchronously**, and interacts with both the monitoring service and the database.

4. **Monitoring and Logging Service (InfluxDB)**

   - **Role**: Tracks performance metrics and task behavior.

   - **Operations**:

     1) Receives logs and alerts from workers.
     2) Allows real-time monitoring of system behavior.

   - **Interaction**: Communicates **asynchronously** with the worker.

5. **Database (InfluxDB)**
   - **Role**: Central data storage.
   - **Operations**:

     1) Stores user data, task status, and log entries.

2) Accessed by the worker when needed.

- **Interaction**: Accessed **synchronously** by the worker for reads/writes.

Resource Scaling Estimation

10 simultaneous connections:

For a small number of users, the setup can stay pretty lightweight:

- **Web Server**: 1 core and about 1–2 GB of RAM should be enough.
- **Celery Workers**: One worker should do the job here, also with 1 core and 1–2 GB RAM.
- **Redis**: Doesn't need much—1 core and less than 1 GB of RAM will work fine.
- **InfluxDB**: Can run smoothly on 1 core and 1–2 GB of RAM.

50 simultaneous connections:

With more traffic, we'll need to scale things up a bit:

- **Web Server**: 2 cores and 2–4 GB of RAM should keep things responsive.
- **Celery Workers**: Around 2 to 3 workers should handle the load, each with 2–3 cores and 2–3 GB RAM.
- **Redis**: Lift it up to 1–2 cores and 1–2 GB of RAM.
- **InfluxDB**: Somewhere between 1–2 cores and 2–4 GB RAM should be safe.

100 or more simultaneous connections:

For heavy usage, we'll need to perform more scaling:

- **Web Server**: 4 cores and 4–8 GB RAM to keep up with incoming requests.
- **Celery Workers**: We'll probably need 5–7 workers, each with 5–7 cores and 4–6 GB RAM to handle tasks efficiently.
- **Redis**: Should be scaled to 2–4 cores and 4–6 GB of RAM.
- **InfluxDB**: Needs more power too—2–4 cores and 4–8 GB RAM should be enough.

But in general, no matter how many users are connected, it's important to keep an eye on overall system performance. If any part of the system—like the web server, a worker, or the database—starts becoming a bottleneck, that's where we should focus on scaling. The idea is to reinforce the components that are slowing things down and give them the resources they need, so the whole system keeps running smoothly.