A) Assuming that the device is sending continuously at 16000Hz for the X,Y, and Z acceleration measurements, what would your strategy be in handling and processing the data? How would you design the server infrastructure? Please enumerate the steps, software, algorithms, and services that you would use to ensure that the servers can handle the incoming data from our users. Diagrams can be really helpful for this.

1. Using Cloud Run to Create an API Server

Cloud Run includes a built-in load balancer, which means that when deploy the API server there, Google automatically manages traffic distribution.

1. **Avoiding cold starts:** I'd set a minimum number of instances to ensure there are always warm instances available, even during low traffic, improving response time.
2. **Choosing the right region:** I'd pick a region with plenty of resources to avoid hitting capacity limits. If needed, I'd go for a multi-region setup to spread the traffic across different regions.
3. **Configuring concurrency**: I'd adjust concurrency settings based on the expected traffic and the nature of the data load.

   For example, if handling 1,000 requests per second with 10 concurrent requests per instance, Cloud Run would scale to about 100 instances.

2. Using Multi Region Load Balancer

Cloud Run's built-in load balancer only works in one region. So, if that region gets too much traffic or goes down, the API might not work. To avoid that, I'd set up a multi-region load balancer. That way, traffic gets spread across different regions, and the API stays online even if one region has problems or traffic spikes.

3. Using Monitoring System

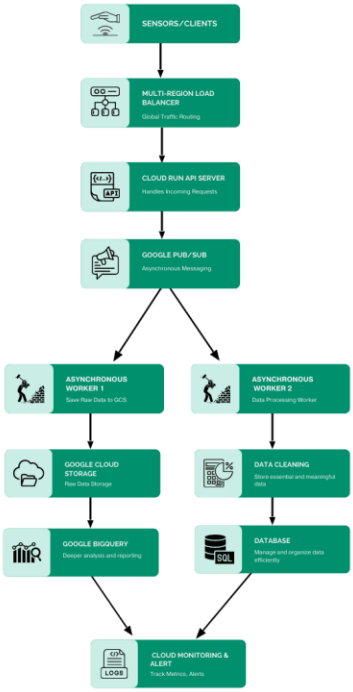
1. Creating a custom dashboard in monitoring helps to see important metrics like traffic, CPU usage, memory, and how many instances are running. This makes it easier to track how the API server is doing.
2. I'd also set up alerts to catch any problems early. For example, I could create an alert if the number of instances goes over a certain limit or if there are errors in the API server.

4. Background Processing via Pub/Sub

There's an API route in Cloud Run to save input data to database. Before saving it, we need to calculate the data requested every second, and each request has a 48,000 sampling rate per second. This can cause the API server to slow down or even time out because there's just too much to calculate. To fix this, I send the data to Pub/Sub and let a background Cloud Run worker handle the processing. This way, the API can continue accepting other requests without issues. It's like using Celery, but serverless. Unlike Celery, which needs a worker running all the time, Cloud Run isn't ideal for that since instances shut down when they're not in use. Pub/Sub works better here because it automatically triggers a new Cloud Run instance each time a message is published, allowing it to handle the task without blocking other requests.

5. Save the raw data in Google Cloud storage

During the calculation process triggered by Pub/Sub, we also save the raw data to Google Cloud Storage (GCS). This provides a backup of the original input data, which we can later load into BigQuery for deeper analysis. We format the data using the user ID and the date, making it easier to organize and query when analyzing large datasets.



B. If the sensors were upgraded to modern mobile devices, how would you change your architecture from A?

I think the current cloud setup in A can still handle an upgrade to modern sensors or deal with API requests pushing more than 48,000 data points per second, since it's built to scale and handle high traffic.

However, if we decide to upgrade, we could split things up using Cloud Run for the API, another for calculations, one for storing raw data, and another for the database. The key is making sure all these services can scale and handle heavy traffic. If the smaller services can't scale, even if the main one can, it'll cause backlogs as requests pile up. This would make the extra services unnecessary and add complexity to the infrastructure, as we'd need to manage each service separately.