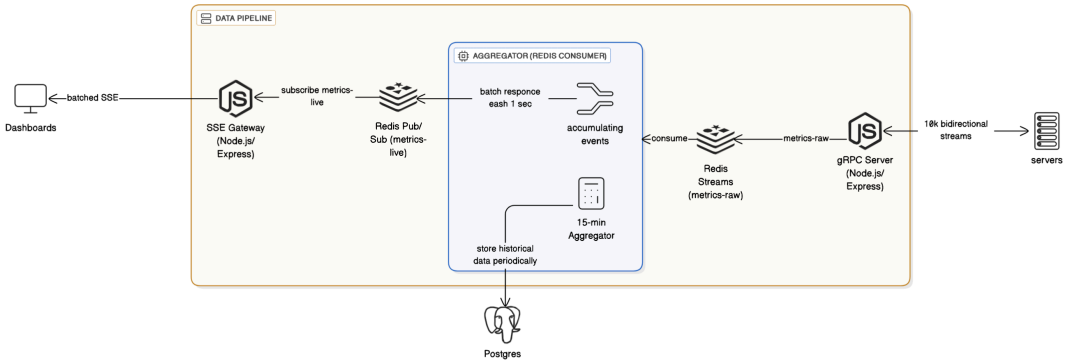
ARCHITCTUR2

System architecture diagram



OSI model layer breakdown of your protocols

Network Layer:IP (IPv4/IPv6) handles routing packets, Ensures that all data reaches the correct destination.

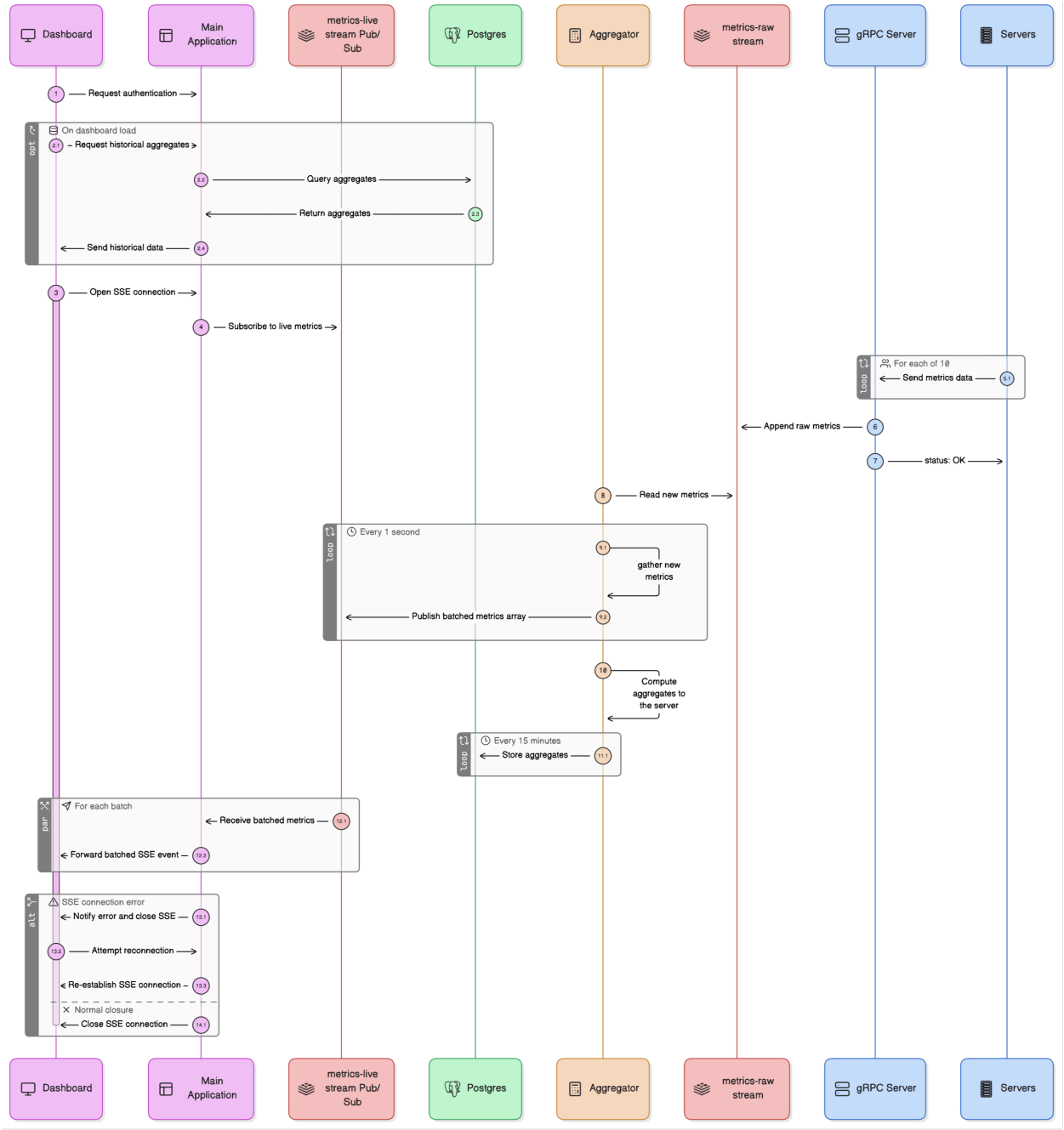
Transport Layer: TCP for HTTP/1 and HTTP/2

Session Layer: HTTP/1, HTTP/2 manage sessions and connection lifetimes.

Presentation Layer: Protocol Buffers , JSON formatting.

Application Layer: gRPC service methods, HTTP/1 requests, Redis Pub/Sub, SSE for real-time updates.

Sequence diagrams for key flows



Why you chose specific patterns

Server-to-Server gRPC

- For high-frequency metric ingestion system, gRPC is used for server-to-server communication. It allows remote calls to feel like local function invocations while leveraging HTTP/2's multiplexed

streams, low-latency communication, and efficient binary serialization. This makes gRPC ideal for high-throughput, low-overhead data transfer between servers.

dashboard-to-Application http1

- http/1 is preferred for real-time browser updates due to its simplicity, native support, and lower overhead compared to gRPC-Web which is heavy and more complex .

Decoupled Aggregation Layer

- Metrics are sent to a separate aggregation service, using Redis as a temporary store.
- This decoupling reduces latency for each request, thereby making the system more responsive and improving throughput.
- publish-subscribe pattern avoids constant polling and ensures dashboards receive near real-time updates.

Batching for Efficiency

- Metrics are **batched** before sending, which **reduces the number of messages and network load** while maintaining timely updates.
- **Problem with sending individually:** Each JSON metric ≈ **100 bytes**; at **10,000 metrics/sec** for **500 dashboards**, this would be **500 MB/s (~4 Gbps)**, which is highly resource-intensive and risks saturating server and network resources.

Design decisions and trade-offs

1. **gRPC for Server-to-Server Communication:**

   Browsers cannot natively use full gRPC streams, so an extra layer is needed for client(engineer) communication.

2. **Decoupled Aggregation Layer**
   adding layer for aggregation requires monitoring Redis performance.

3. **Batching Metrics**
   batching slightly increases the latency per metric.