Summary Report on Real-Time Communication Techniques

A Practical Overview of Redis, Kafka, WebSockets, and Server-Sent Events (SSE)

Sandeep Kumar (CS24M112)
Ashant Kumar (CS24M113)
Dharmendra Chauhan (CS24M115)

Abstract

This report briefly summarizes the core real-time communication methods used in our Seminar Course project (source code: GitHub). Our intention is to compile the key functions, their purposes, and their typical usage patterns so that this document can serve as a quick reference for any student who wishes to understand or extend similar systems. The focus is on four technologies—Kafka, Redis, WebSockets, and Server-Sent Events (SSE)—that form the backbone of most modern real-time applications.

1 Kafka: Producer and Consumer Essentials

Kafka acts as a high-throughput messaging backbone. Below are the fundamental functions we used while implementing producers and consumers.

1. Producer Setup

Purpose: To establish a producer that can push messages into a Kafka topic. **Inputs:** Broker address and message content. **Output:** Message published to the Kafka topic.

```
const { Kafka } = require("kafkajs");
const kafka = new Kafka({ brokers: ["localhost:9092"] });

const producer = kafka.producer();
await producer.connect();
await producer.send({
  topic: "orders",
  messages: [{ value: JSON.stringify(orderData) }]
});
```

2. Consumer Setup

Purpose: To continuously listen for new events arriving in a topic. **Input:** Topic name. **Output:** A handler is triggered whenever a new message appears.

```
const consumer = kafka.consumer({ groupId: "log-group" });
await consumer.connect();
await consumer.subscribe({ topic: "logs", fromBeginning: false });

await consumer.run({
  eachMessage: async ({ message }) => {
    console.log("Received:", message.value.toString());
  }
});
```

2 Redis: Lightweight Real-Time Communication

Redis Pub/Sub allows fast message distribution inside our system.

3. Publishing Messages

Purpose: Broadcast information to all listening services. **Input:** Channel name and serialized message.

```
const Redis = require("ioredis");
const redis = new Redis();
redis.publish("order-updates", JSON.stringify(orderData));
```

4. Subscribing to a Channel

Purpose: React to messages as soon as they are published.

```
const sub = new Redis();
sub.subscribe("order-updates");

sub.on("message", (channel, message) => {
  console.log("Update:", JSON.parse(message));
});
```

3 WebSockets: Two-Way Real-Time Interaction

WebSockets enable full-duplex communication between the browser and the server.

5. WebSocket Server

Purpose: Accept WebSocket clients and maintain live connections.

```
const WebSocket = require("ws");
const wss = new WebSocket.Server({ port: 5000 });

wss.on("connection", (ws) => {
  ws.send("Connected to WebSocket Server");
});
```

6. Sending Data to the Client

Purpose: Push updates directly to the active users.

```
ws.send(JSON.stringify({ event: "location-update", data }));
```

4 Server-Sent Events (SSE)

SSE is useful when only the server needs to push updates to the client.

7. Creating an SSE Stream

Purpose: Send continuous updates over a single HTTP connection.

```
res.writeHead(200, {
 "Content-Type": "text/event-stream",
 "Cache-Control": "no-cache",
 Connection: "keep-alive"
});
res.write(`data: ${JSON.stringify(log)}\n\n`);
```

5 Frontend Real-Time Integrations

8. WebSocket Client Usage

Purpose: Receive real-time information from the backend.

```
const ws = new WebSocket("ws://localhost:5000");

ws.onmessage = (msg) => {
  updateMap(JSON.parse(msg.data));
};
```

9. SSE Client Usage

Purpose: Get log events streamed directly from the server.

```
const events = new EventSource("/stream-logs");

events.onmessage = (e) => {
  displayLog(JSON.parse(e.data));
};
```

6 Docker Setup for Redis and Kafka

For convenience during development, we used Docker to quickly launch Kafka, ZooKeeper, and Redis.

10. docker-compose.yml Example

```
version: "3.8"
services:
  zookeeper:
    image: confluentinc/cp-zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
    ports:
      - "2181:2181"

  kafka:
    image: confluentinc/cp-kafka
    depends_on: [zookeeper]
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
    ports:
      - "9092:9092"

  redis:
    image: redis:latest
    ports:
      - "6379:6379"
```

Start all services:

```
docker-compose up -d
```

Conclusion

This short report highlights the main functions and tools we used while developing a real-time system for our seminar course. By understanding these building blocks, any student can follow the structure of our project and adapt the same ideas to different real-time applications.