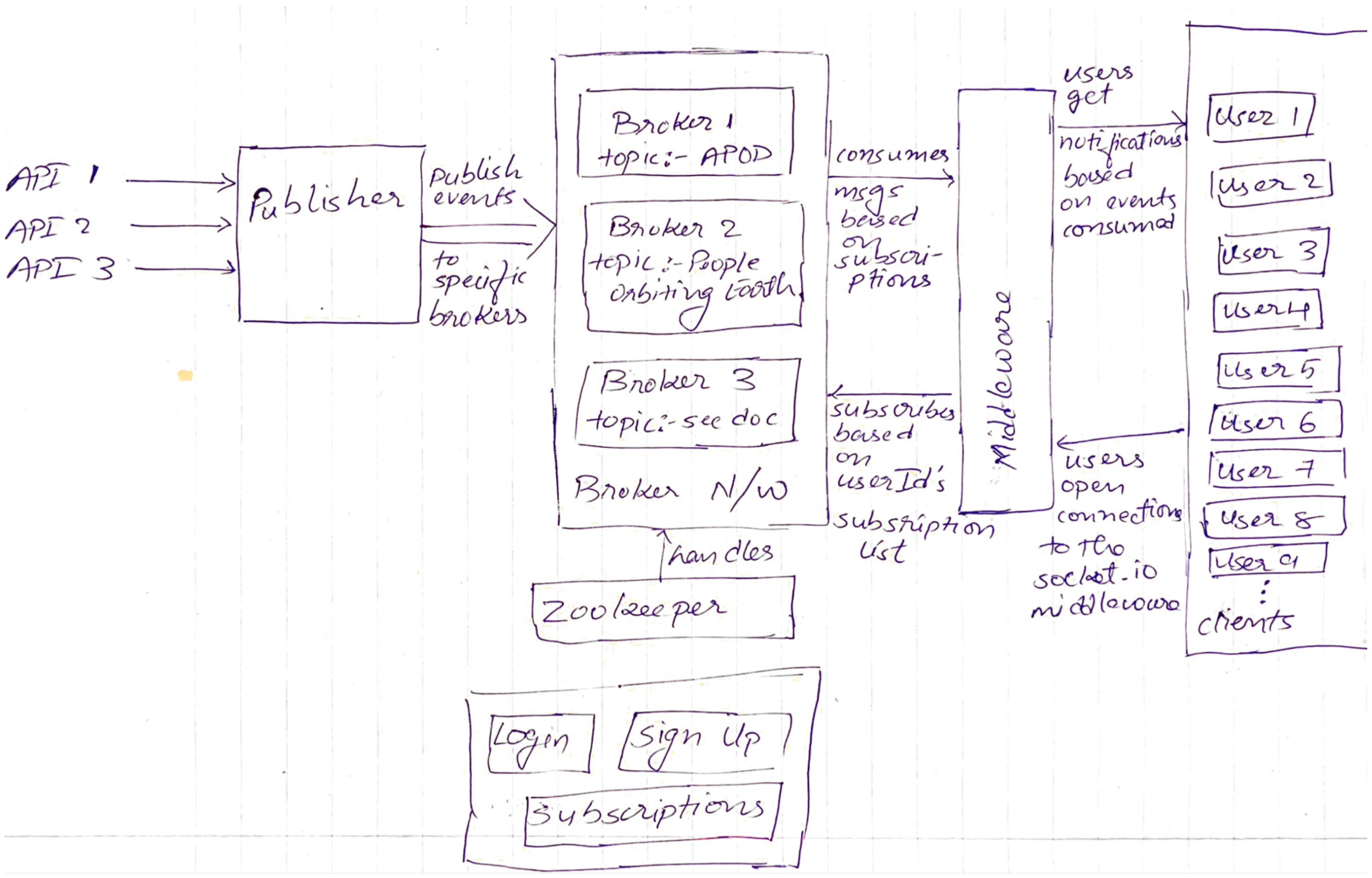
Architecture


API 1
API 2
API 3
Publisher
Publish events to specific brokers
Broker 1 topic:- APOD
Broker 2 topic:- People orbiting Earth
Broker 3 topic:- see doc
Broker N/W
consumes msgs based on subscriptions
subscribes based on userId's subscription list
handles
Zookeeper
Middleware
users get notifications based on events consumed
users open connections to the socket.io middleware
User 1
User 2
User 3
User 4
User 5
User 6
User 7
User 8
User 9
clients
Login
Sign Up
Subscriptions

Topics	Handler(s)
Astronomy Picture of the Day	Kafka Broker **1**
People Orbiting the Earth	Kafka Broker **2**
ISS Current Location	2 Partitions based on *visibility* data property fetched from the API. If *visibility* = *"daylight"* then partition 0 or else 1. Partition 0 ⇒ Broker **3** (leader) Partition 0 ⇒ Broker **1** (replica of partition 0) Partition 1 ⇒ Broker **1** (leader) Partition 1 ⇒ Broker **2** (replica of partition 1)

Dependencies

1. Docker
2. Docker-Compose

   (Docker Desktop includes 1 and 2.)
   (Below listed will be handled by Docker itself.)

3. Zookeeper
4. Kafka
5. kafkajs (⇒ for nodejs)
6. node.js
7. MongoDB
8. react.js
9. socket.io
10. mongoose (to interact with MongoDB)
11. express (for node.js server)
12. express-graphql (GraphQL API used for authentication and Subscriptions storing in DB)
13. axios (getting data from APIs)

Description

Kafka Broker network is used for dissemination of events to all the users subscribed to various topics.

Producers publish data into the Kafka broker network, the network is responsible for data being accessible to all consumers interacting with any broker node. It is also responsible for load balancing.

Consumers can connect to any broker node and subscribe to any or all of the available topics. Therefore, consumers can consume new events that's been published to the network.

According to the architecture, the publisher gets messages from the APIs, it checks whether it is indeed a new message from the list of stored messages (in MongoDB). It then publishes it to the broker network if it deduces that it is indeed a new event.

Kafka Broker network stores the events (in logs) and consumers consume with the latest (fromBegining = false) offset.

Kafka makes these types of systems extremely simple and efficient, with only possible complications being the setup of the network.

Instantiating the System

1. IF there are connection related issues THEN
    a. uncomment 3 commented lines in docker-compose and comment the above 3 lines for each kafka nodes
    b. replace all the <<your_local_ip>> or 10.84.172.109 from docker-compose file AND replace kafka1, kafka2 and so on from producer (./publisher/app.js) and consumer (./middleware/app.js) files with your device's private IP address.
        i. For MACOS, hold the option key and click the WiFi, Ethernet symbol in the menu bar, then copy the IP Address. OR run *ifconfig* | *grep "inet "* in the terminal and get the non-localhost private IP address. For example, 10.84.172.109
        ii. For LINUX, use the same command *ifconfig* | *grep "inet "* and replace with it.
        iii. For windows, run CMD in admin mode and get the private IP address using *ipconfig* command and replace with it.
        iv. NOTE: for deploying this, instead of using the private address, it will need the static public address of the system where it's deployed.
2. Run ***docker-compose up*** from the workspace directory. Topics and everything else is set up programmatically. So no need to run shell scripts provided by Kafka to create topics.
3. Now open http://localhost:3000
4. Create a user.
5. Subscribe to events listed in the Topics tab.
6. View your subscriptions in the Subscriptions navigation tab.
7. View live incoming notifications in the Notifications tab.

To interact with Kafka brokers, use Docker's exec commands.

That's it. The main critical part is the IP address setup for instantiating the system.