

Project 1 - Phase 3

Cryptocurrency Exchange Data Streaming Implementation (Publisher/Subscriber Model: Distributed Version)

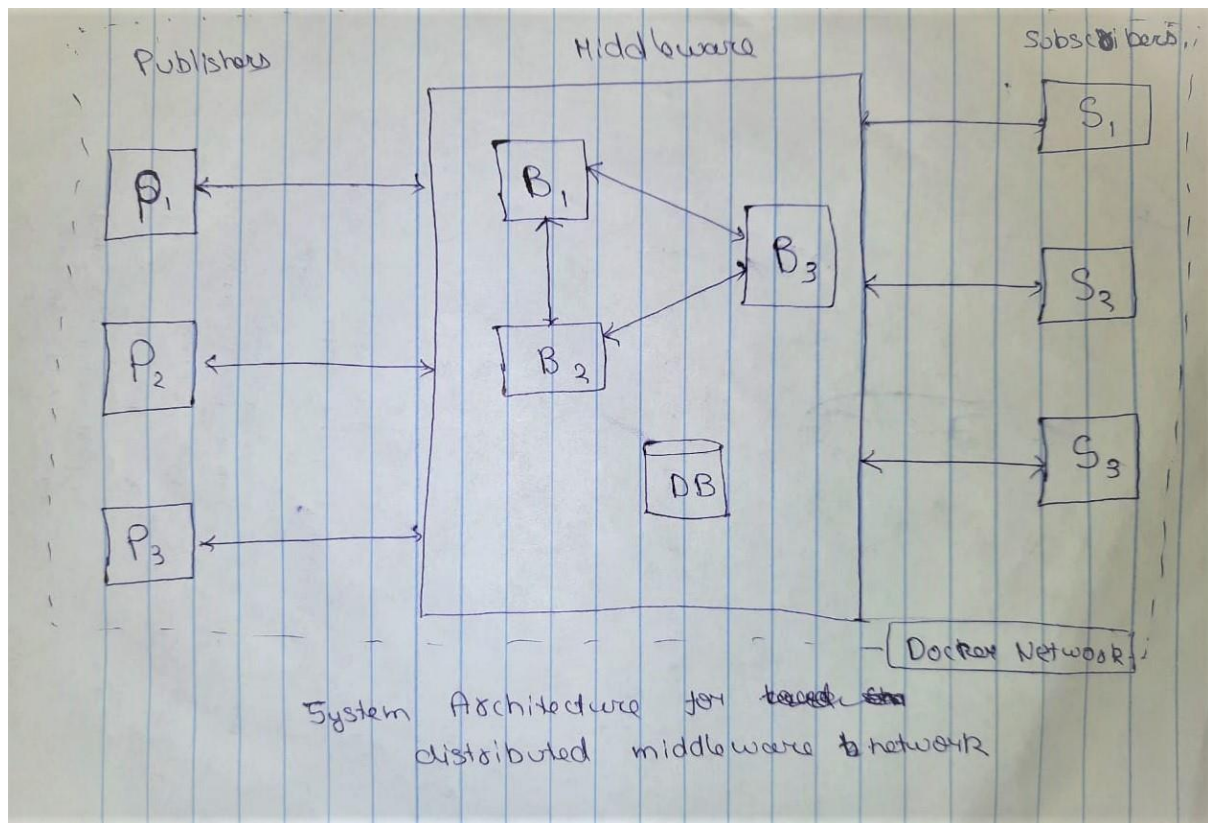
Introduction:

Here we have implemented a data streaming platform for tracking crypto exchange rates from different exchanges. *Distributed Publish/ Subscribe model* is used as middleware here. In addition, we have publisher module that publishes the current crypto prices to the *network of brokers* pub/sub system, and further the system notifies the subscribers interested in a particular topic, cryptocurrency like BTC, ETH, DOGE in our case.

Architectural Model

Modules:

- a) **Publisher:** The publisher module continuously receives the latest prices for coins from independent external sources, cryptocurrency exchanges - Coinbase, Binance, Gemini in our case using their respective APIs . The model further publishes this event of update of new prices to the data broker module for sending to the interested subscribers. Other functionalities like advertise and deadvertise are also implemented at the publisher's end. The advertise method lets publishers publish all the possible topics to all the subscribers via middleware which are available for them to subscribe.
- b) **Network of Brokers (Middleware):** Here we have replaced the centralized Data broker system with network of brokers (3 brokers) which co-operate using Rendezvous approach to notify the published data to the subscribers who have subscribed to that particular topic. The mapping of the subscribers and published data is achieved by maintaining database collections, storing the topic wise published data, and storing the subscribers list for the subscribed topics for each node in broker network. The notify function implementation maps the subscriber with the published topic and to send notification of new prices to the subscriber module.
- c) **Subscriber:** This module has a frontend implementation which lets the user enter the subscriber id and the topic to subscribe. The subscriber system sends a request to middleware to make a new subscription. At this end we have functionality to unsubscribe to a topic as well, here again the subscriber interacts with the pub/sub system to make updates for no more notifications for that particular topic.



Broker Module Implementation using Rendezvous

The broker module is network of 3 brokers each responsible for 1 topic each. Broker 1 maintains the subscriptions of BTC and send out notifications of new prices published from publishers (crypto exchanges) like Binance, Coinbase to the respective subscribers. Similarly Broker 2 is responsible for ETH and Broker 3 is responsible for DOGE coin respectively. Like Broker 1 these later two brokers also maintain subscribers for the topics they are handling and notifies for the same.

The interaction among the brokers is handled using Rendezvous algorithm. Each Rendezvous node maintains subscribers list and notifies about the event to the subscriber it is responsible for or otherwise forward the event to the rendezvous list. So, in our case if Broker 1 which is responsible for BTC receives the prices for ETH, it forwards the event to Broker to which is responsible for ETH which send out notifications for the same. Subscriptions are also handled in similar way, if broker 1 receives the subscriptions for BTC it adds to the subscribers list otherwise if it is for ETH or DOGE it forwards it to the other rendezvous nodes.

Containerization Info:

We have divided our system into multiple docker containers. We have a separate container for each publisher and subscribers. Also, each broker and mongo database are running in a separate containers.

Steps to deploy the systems and run containers

Step 1 - Open terminal and go to the directory of the unzipped folder with docker compose files(.yaml).

Step 2 - `$ docker-compose -f mongo.yaml build`
(Build container image for Database)

Step 3 - `$ docker-compose -f middleware.yaml build`
(Build container images for middleware pub/sub module)

Step 4 - `$ docker-compose -f publisher.yaml build`
(Build container images for publisher module)

Step 5 - `$ docker-compose -f subscriber.yaml build`
(Build container images for subscriber module)

Step 6 - `$ docker-compose -f mongo.yaml up -d`

Step 7 - `$ docker-compose -f middleware.yaml up -d`

Step 8 - `$ docker-compose -f publisher.yaml up -d`

Step 9 - `$ docker-compose -f subscriber.yaml up -d`

Steps to interact with the publisher

Step 1 - Open a new terminal session for the publisher program to execute.

Step 2 - Run - `$ docker exec -it publisher1 java -jar Coinbase.jar`
Run - `$ docker exec -it publisher2 java -jar Binance.jar`
Run - `$ docker exec -it publisher3 java -jar Gemini.jar`

This will make a session with the container(publisher1).

```
C:\Users\yoges\OneDrive\Documents\NetBeansProjects\Phase3\Phase3>docker exec -it publisher1 java -jar Coinbase.jar
Choose from menu:
Press 1 to Publish new prices
Press 2 to advertise new topic
Press 3 to deadvertise a topic
Press 4 to exit
1
Latest prices from Coinbase published
```

Similarly, we can interact with other publisher 2 and publisher 3.

```
Choose from menu:
Press 1 to Publish new prices
Press 2 to advertise new topic
Press 3 to deadvertise a topic
Press 4 to exit
2
Enter cryptocurrency to advertise
RIP
RIP Advertised!

Choose from menu:
Press 1 to Publish new prices
Press 2 to advertise new topic
Press 3 to deadvertise a topic
Press 4 to exit
3
Enter cryptocurrency to deadvertise
RIP
RIP Deadvertised!
```

Advertise and deadvertise example.

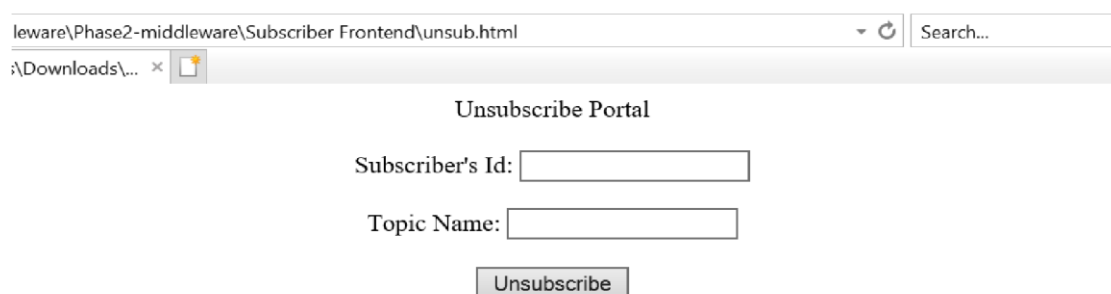
Steps to interact with the Subscriber

Step 1 We have two web pages to interact with subscribers. One is subscribe.html to subscribe to a topic as below. You can enter any sub id and choose a topic from BTC, ETH, DOGE which are available in our case to subscribe from.



The screenshot shows a web browser window with the address bar displaying 'C:\Users\yoges\Downloads\Phase2-middleware\Phase2-middleware\'. The page title is 'Subscribe Here!'. The form contains two input fields: 'Subscriber's Id:' and 'Topic Name:'. Below these fields is a 'Subscribe' button.

Step 2 We can unsubscribe to any topic using html page unsubscribe.html

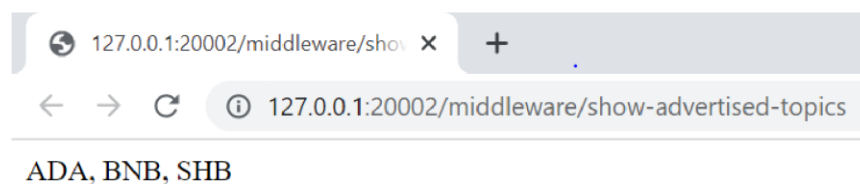


The screenshot shows a web browser window with the address bar displaying 'leware\Phase2-middleware\Subscriber Frontend\unsub.html'. The page title is 'Unsubscribe Portal'. The form contains two input fields: 'Subscriber's Id:' and 'Topic Name:'. Below these fields is an 'Unsubscribe' button.

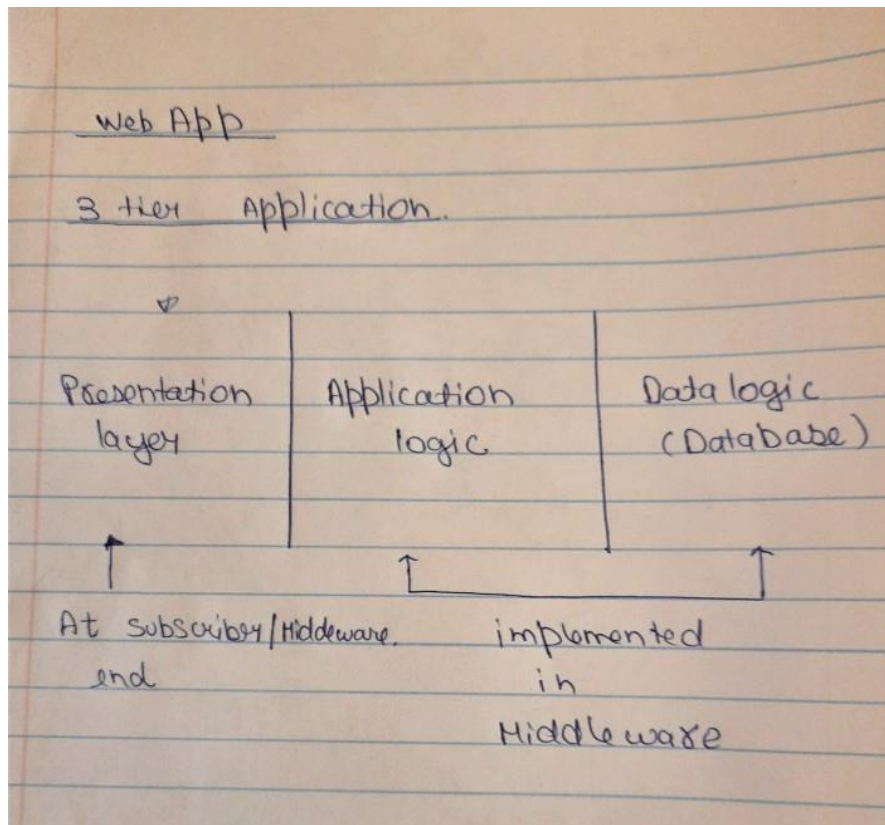
Step 3 We can view the data received by subscribers by hitting their respective APIs. The route is **/sub/displayData** but the IP and port combination varies for each subscriber.



Step 4 We can view the advertised topics to all the subscribers by hitting the advertisement API. The route is **/middleware/show-advertised-topics**.



Application Architecture



Summary:

We implemented a data streaming platform for tracking crypto exchange rates from different exchanges using the distributed publish/subscribe model of indirect communication, enabling loose coupling, thus making our system dynamic, scalable, robust, heterogeneous, and asynchronous.

References:

Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012