Bhavesh Khatri , 2018030
Mohd. Huzaifa , 2018158

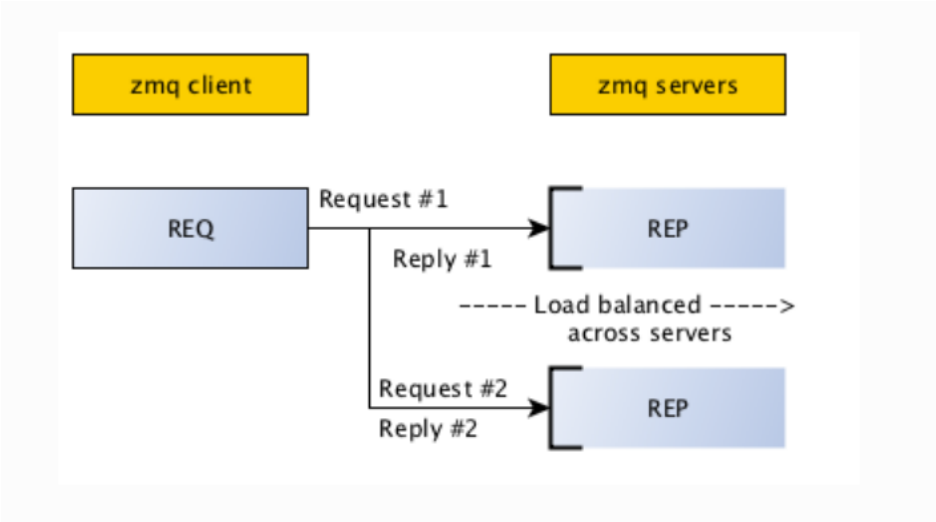
Client-Server

In the Client-Server pattern we have tried to mimic the behaviour of the Amazon Server, where we have multiple articles present in multiple servers (limited to 2 servers as a starting point).

As a point of start, we have already set up two servers that contain multiple articles. And there can be as many clients, who will try to look up to those articles as per their choice. A client would be given a choice to connect with which server or both, and then request for a particular article. If the article is present it would be returned, otherwise a “Not Found” message would be displayed.

Salient Features

- Multiple Clients can connect to the servers to look up for articles. **(Multiple clients)**
- Multiple Servers would be there to serve the clients. **(Multiple servers)**
- Clients would be given the option to choose from which Server they want to look for the articles.
- A Server can serve multiple clients, looking for articles. **(One-Many)**
- Many Servers would be there to serve multiple clients, to load balancing (efficient distribution of traffic). **(Many-Many)**



[Image Source]

Server.py	Client.py
Run : 1. python Server.py 1 2. python Server.py 2	On multiple command prompts Run : Python Client.py

Bhavesh Khatri , 2018030
Mohd. Huzaifa , 2018158

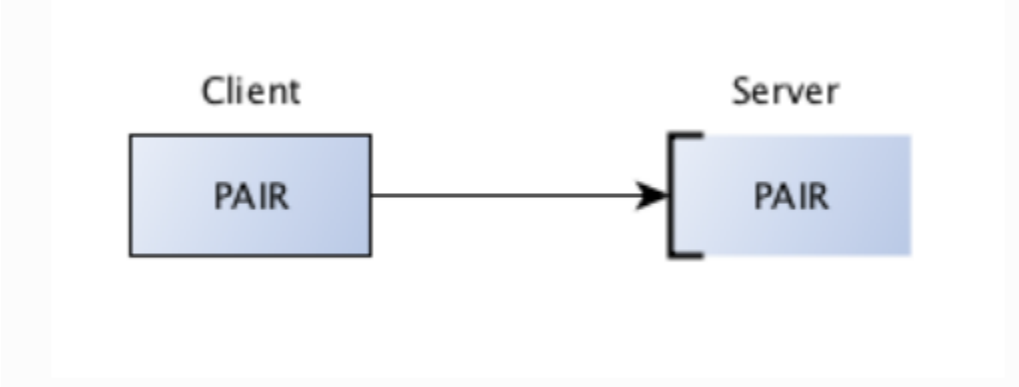
PAIR

In the Pair pattern we have tried to mimic the behaviour of the Expedia Customer Care Executives, where we have multiple executives present to serve multiple clients.

As a point of start, we have already set up two executives for serving the clients. And there can be as many clients, who will try to connect with executives. However, once a client is connected to the executive, then no other client can connect to the same executive until the initial client closes the conversation.

Salient Features

- Multiple Clients can come to connect with the executives. **(Multiple clients)**
- Multiple executives would be there to serve the clients. **(Multiple servers)**
- An executive can serve only one client at a time. **(One-One)**



[Image Source]

Server.py	Client.py
Run : 1. python Server.py 1 2. python Server.py 2	Run : Python Client.py

Bhavesh Khatri , 2018030
Mohd. Huzaifa , 2018158

Publisher-Subscriber

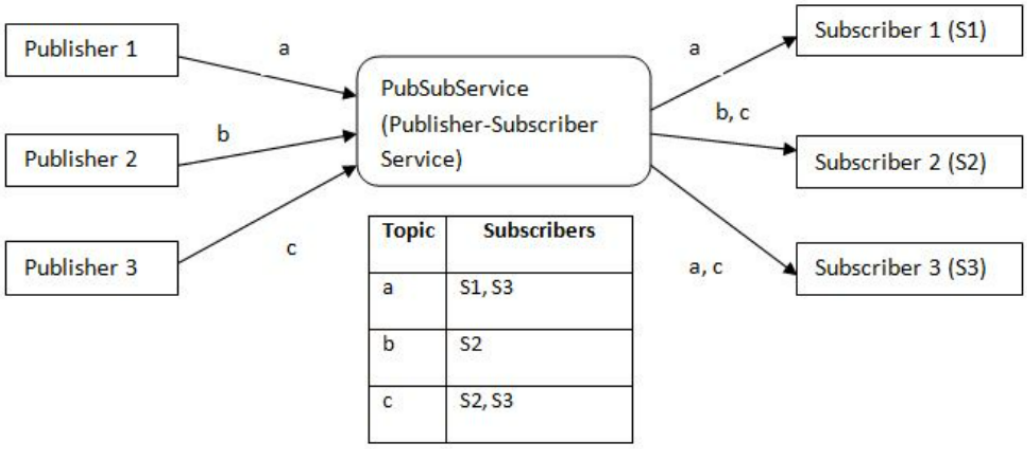
In the Pub-Sub Pattern we have tried to mimic the behaviour of the Youtube application, where we have multiple publishers posting their content (limited to textual messages in our case), and those who have subscribed can view the content.

As a point of start, we have already set up a few publishers (say 5 initially) who can post content. And there can be as many subscribers, who will subscribe to the publishers as per their choice. And then whenever any post is made, then the corresponding subscribers will receive the content.

Salient Features

- Multiple Publishers can join the community and put their posts. **(Multiple pubs)**
- Multiple Subscribers can join the community to receive the posts.**(Multiple subs)**
- Subscribers would be given the option to choose from which publisher they want to see the posts coming.
- Subscribers can subscribe and receive posts from multiple publishers. **[Many-One]**
- A publisher can have multiple subscribers, who can see their posts. **[One-Many]**

Note: Feature 4 and 5 as a combined result acts as **Many-Many**



Publish-Subscribe Design Pattern

[Image Source]

Bhavesh Khatri , 2018030
Mohd. Huzaifa , 2018158

Publisher.py	Subscriber.py
On 5 command prompts : Run: python Publisher.py 1 python Publisher.py 2 python Publisher.py 3 python Publisher.py 4 python Publisher.py 5	On multiple command prompts Run : Python Subscriber.py

Push-Pull Pipeline

In the Push-Pull Pipeline pattern we have tried to mimic the behaviour of the Zomato application, where we have multiple restaurants that are producing orders, and then we have the Delivery person who picks up the order. The delivery person takes some time(say t) to deliver the order and then comes back and again picks up another order.

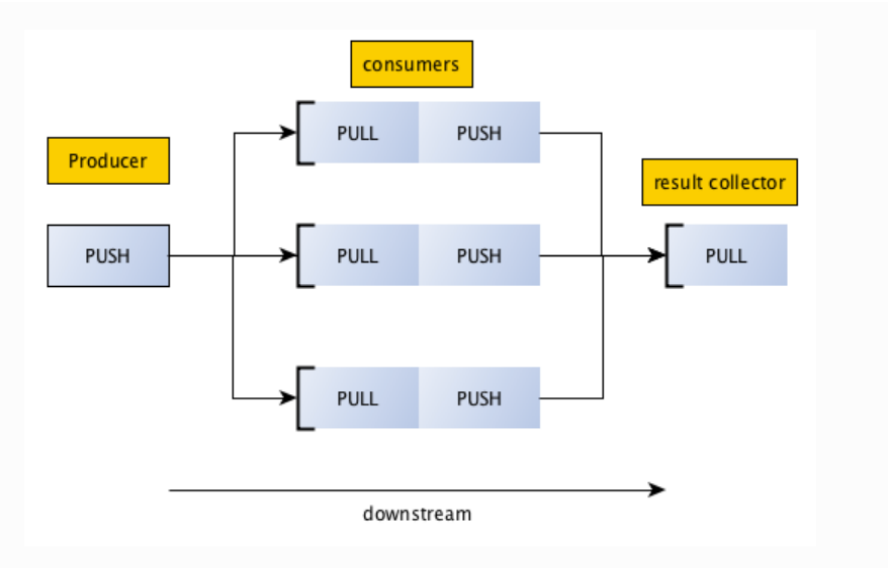
As a starting point, whenever a restaurant comes and enters the orders, the set of Delivery person comes up and picks the order and then returns back to pick further order after delivering the initial order. Once an order has been delivered, the Delivery guy reports the successful delivery, which can be at a later stage used to verify how many deliveries have been done by a particular guy or say number of orders from a particular restaurant.

Salient Features

- Multiple Restaurants can join the community and put their orders. **(Multiple sources)**
- Multiple Delivery people can join the community and start delivering orders. **(Multiple workers)**
- Multiple Delivery guys would be coming together collectively to deliver orders from a particular restaurant. **(push - multiple pulls)**
- Same Delivery can pick orders from multiple restaurants that are serving orders. **(multiple pushes - pull)**

Note: Feature 3 and 4 as combine results, acts as **multiple pushes - multiple pulls**

Bhavesh Khatri , 2018030
Mohd. Huzaifa , 2018158



[Image Source]

Source.py	Worker.py	Sink.py
On multiple command prompts Run : Python Source.py	On multiple command prompts Run : Python Worker.py	Run: Sink.py

References

1. Client-Server
2. Pair
3. Pub-Sub
4. Push-Pull