Homework: In class discussion-EDA systems & Kafka

By: Abdul Moiz Mateen, Abdullah Khan, Faisal Alsheet, Sterling Gallion, Taimour Arshad, Jachimike Ezenwamadu

Q1) What are the different components of both figures and what are their functions?

Event-Driven messaging:

- **API Gateway** - A service that serves as a means of communication between a client and an application.
- **Microservice** - A service independent of other microservices that handles its own part of the application.
- **Event** - A message that originates from a publisher microservice that subscriber microservices might be interested in. They contain information about a state change or action that happened.
- **Event bus** - Works as a message broker that allows microservices to publish events and other microservices to subscribe to those events. In the figure 4-15 example the shopping basket microservice publishes an event to the event bus, then the ordering and inventory microservice

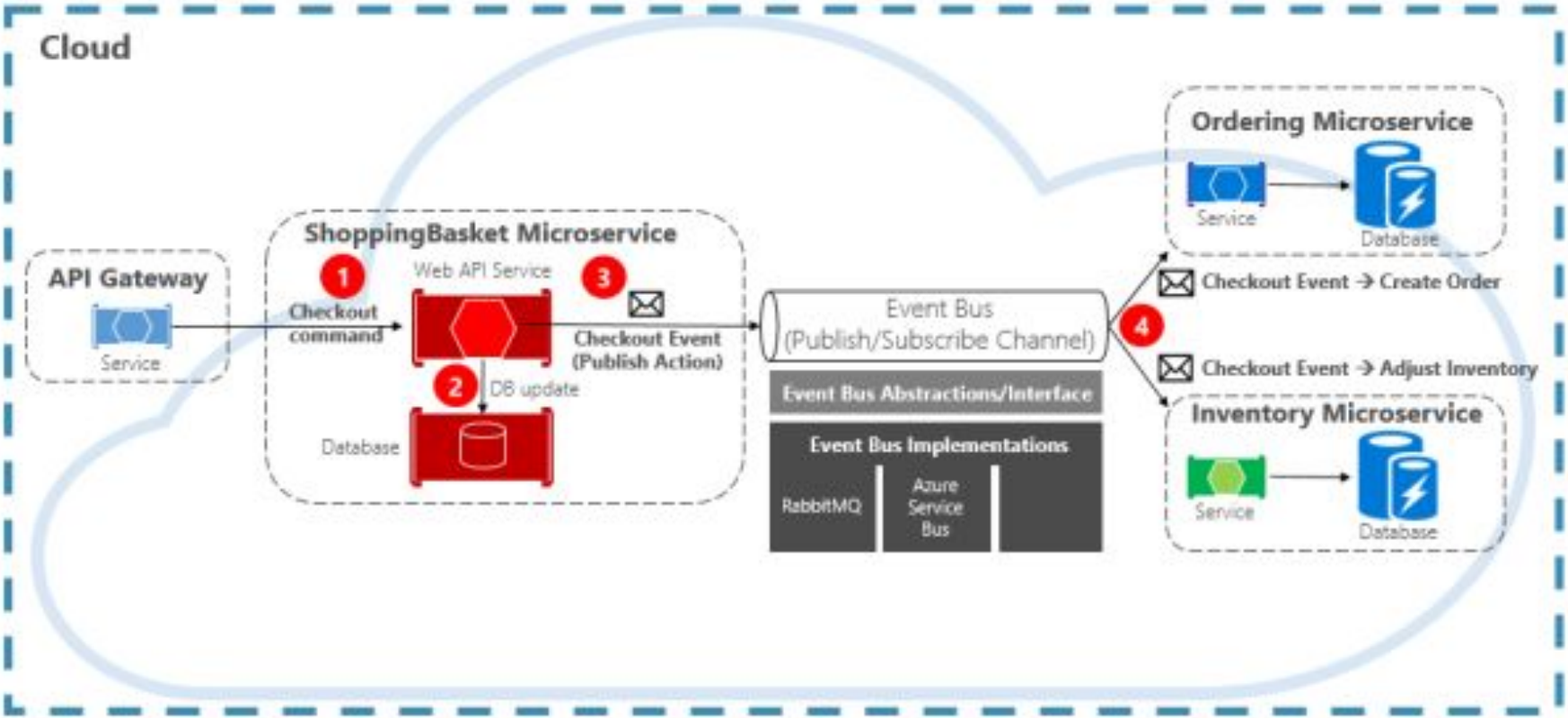


Figure 4-15. Event-Driven messaging

Q1) What are the different components of both figures and what are their functions?

Topic architecture:

- **Publisher** - A microservice that sends a message through the topic to one or more microservices.
- **Topic** - Works like a queue, but can be used in a one-to-many messaging pattern. The topic is used to send the messages from publishers to the subscribers based on the filtering rules.
- **Filtering Rules** - They are rules defined to filter events so they only arrive at specific subscribers. For example a subscriber could receive all messages for logging purposes, or the rules could filter messages that only relate to price to price-related subscribers.
- **Subscribers** - Microservices that are interested in specific information that may come from the publishers.

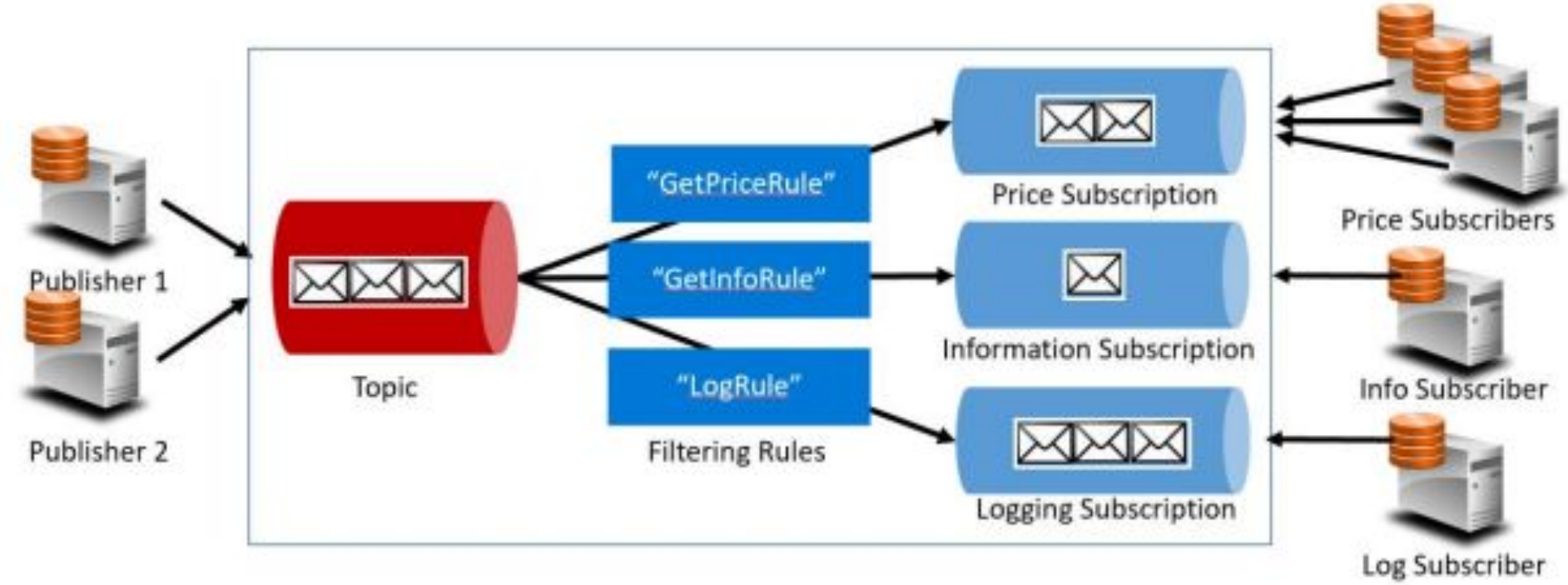


Figure 4-16. Topic architecture

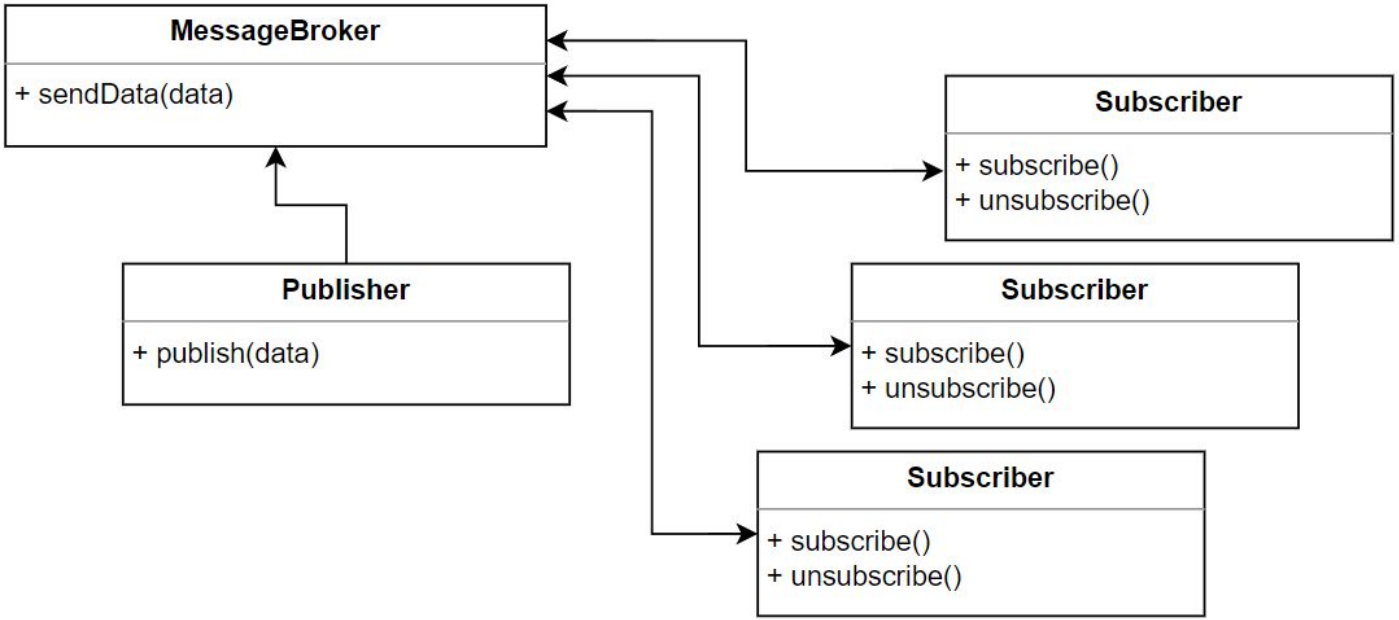
Q1) How do they map to the part you read earlier?

- They would be considered asynchronous messaging patterns. An example of that is the event-driving messaging working in an asynchronous pattern by using RabbitMQ as an event bus to send information from one microservice to another.
- These fall under the event type of communication interaction where the publisher and subscriber are unaware of each other, and the publishers just send information to any interested subscribers

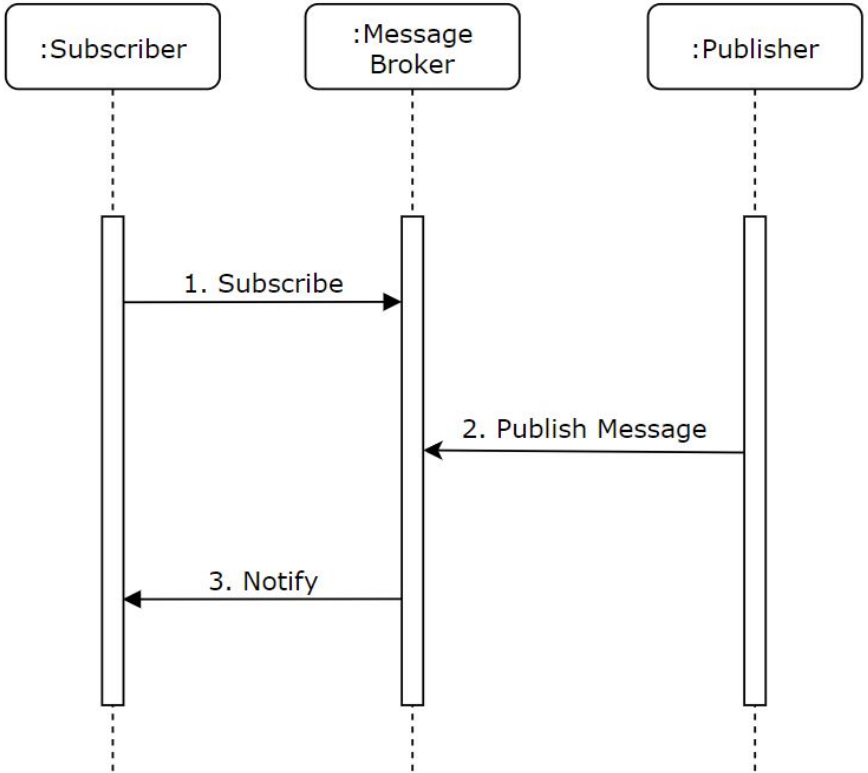
Q2: A major pattern for data ingestion is pub-sub?

- Pub-Sub is used for data distribution and not for data ingestion, however when using Pub-Sub, we can utilize either Apache Kafka Connect which allows us to connect various data sources to Apache Kafka, and then stream that data to specified Kafka topics.
- Also, Google Cloud Pub/Sub can be utilized since it is a fully managed messaging service by Google Cloud and it allows data ingestion and distribution within the Google Cloud's ecosystem.

UML Class Diagram for Pub-Sub:



UML Class Diagram for Pub-Sub:



Q3: How is MQ different from pub-sub?

1. Message Flow:

a. For **MQ,** messages are sent from a producer to a queue, where then the queue sends out the messages to their correct destinations
b. For **Pub-Sub**, messages are directly broadcasted (published) to all interested subscribers

2. Order of Processing Messages:

c. In **MQ**, the messages are processed in the queue in a FIFO manner which ensures sequential processing of messages
d. In **Pub-Sub,** there is no guaranteed order of message delivery to subscribers

Q3: How is MQ different from pub-sub?

3. Number of Message Consumers:

a. In **MQ**, one message is consumed by exactly one receiver
b. In **Pub-Sub**, one message can be sent to many subscribers (receivers)

4. Pattern Use Cases:

a. **MQ** is used for tasks that require a strict ordering and a guaranteed delivery of messages
b. **Pub-Sub** is used for tasks that require broadcasting certain messages to certain group of interested subscribers and pub-sub is used when disconnecting the different publishers from the subscribers is essential in the specific scenario

Message Queue (MQ) vs. Pub-Sub

MQ

- Has a queue that receives messages from different subscribers and then processes these messages sequentially in the queue
- Messages are sent (published) to a single subscriber
- Provides message persistence, acknowledgement, and guaranteed delivery
- Provide load balancing, task distribution, and ensuring no copies of a message are created

Pub-Sub

- Messages are published (sent) to a channel where many subscriber do receive that message
- Subscribers (receivers) can subscribe to specific channels to receive specific messages of their choosing
- Messages published by senders (publishers) are broadcasted to each interested subscriber and each of these subscribers receives one copy of the message
- Often used to broadcast events and real-time communications

Q4:Can you list the main functions of Kafka?

- Replication
  - Kafka allows replication of partitions for reliability and ensuring that if one partition or broker goes down then another replicate can become the lead for that replication.
- Pub/Sub Architecture
  - Kafka uses the pub-sub architecture where producers send data to the partitions which are stored in logs, and consumers can read from the lead replica within the partition
- Scalability
  - With the ability to partition the logs across multiple servers, Kafka allows for horizontal scaling as if there is more consumers, more brokers, partitions and replicas can be created to support the demand from consumers
- Retention
  - Kafka allows for the ability to specify how long messages stay within the logs, whether that be a short amount of time or a long to infinite amount of time.

Q4:Can Kafka be considered a distributed system? why? How?What features of Kafka make it considered such?

Kafka can be considered a distributed system due to the fact that logs can be partitioned across many servers as well as creating replicas of those partitions.If any of the servers fail, then another server can use its replica and make it the lead replica for the missing partition. With the ability to scale horizontally by creating more partitions and replicas, Kafka exhibits the characteristics of a distributed system.

Q5:In a generic-purpose pub-sub such as Kafka，would we need to ave different "subscribers"? "subscriber types"? would we need to have different "topics"? "topic types"?

A generic-purpose pub-sub such as Kafka would need to have different subscribers, subscriber types, topics and topic types. The subscribers would be parts that receive messages and these parts would need to be split into different types based on the system requirements. The topics would be the content of messages that the subscribers would be receiving. Having topic types would mean topics of various relevant content.

Q5:Can you give real-life examples of system that may use Kafka, or the pub-sub pattern, for thier operations?

Linkedin is a real-life example of a system that uses Kafka and it was actually developed by them. LinkedIn saw a huge increase in the digitized information it accumulated and initially the data consisted of records such as educational, job histories and user connections but over time LinkedIn realized that these records constituted only a small percentage. Actions such as viewing another person's profile and entering keywords in the search bar proved to be invaluable in understanding user needs. LinkedIn realized that storing this data wasn’t enough and the only way to make it valuable was to leverage it by integrating all the data together and sending it to locations where it can be utilized. Due to the fact there was no existing infrastructure that met LinkedIn needs they decided to make Kafka to address this problem. With Kafka serving as the central hub for integrating digitized information from various sources, it became became the foundation for various operations such as:

Q5:Can you give real-life examples of system that may use Kafka, or the pub-sub pattern, for thier operations?

Real-time Activity Feeds: When a user performs an action such as liking a post or making a connection, events related to theses actions are published to Kafka topics so that subscribers such as users followers can consume these events and update the feed in real-time.

Notification System: When a user receives a new message or connection request, Kafka facilities the delivery of these events to appropriate users.

Data Integration with Analytics: Linkedin stores vast amounts of data such as user activities, jobs and interactions with other users. Kafka is used to integrate and stream this data so that it can be used for real-time analytics to help LinkedIn understand its users more.

Q5:A summary on Kafka functional and non-functional features.

Kafka functional features include:

1. Pub-Sub Model
2. Fault tolerance
3. High Throughput

Kafka non-functional features:

1. Scalability
2. Performance
3. Reliability
4. Security
5. Manageability

Q5:Does Kafka need to be a distributed system?

Kafka needs to be a distributed system as it is made up of clusters of servers and clients. The goal of Kafka is to facilitate the continuous flow and interpretation of data from multiple sources like computers, databases and software applications.

Q5：Why are distributed systems hard to design and implement?

Distributed problems occur at logical levels of a distributed system and get worse at higher levels due to recursion.

Distributed bugs can spread across an entire system and take some time to show up after deployment.

The result of any network operation can be UNKNOWN, in which case the request may have succeeded, failed, or received but not processed.

Q5:Advantages of distributed systems?

Better Performance: Distributed systems can perform at high levels by using resources from various computers.

Reliability: If a single node malfunctions, it does not pose problems to other servers.

Cost-Effectiveness: Using a system made up of several computers rather than a single system with various processors is much cheaper.

Efficiency: Each of the computers in a distributed system can work on their own to solve problems. This increases efficiency and save time for the user.

Reduced Latency: Distributed systems make sure that if a node is closer to the user, the system receives traffic from that node thus reducing the time it takes to serve them.