1. What is Apache Kafka and why is it used?

Answer:
Kafka is a distributed event streaming platform used for building real-time data pipelines and streaming applications. It is used for high-throughput, low-latency, fault-tolerant publish-subscribe messaging.

2. What are the core components of Kafka?

Answer:

- **Producer**: Publishes messages to Kafka topics
- **Consumer**: Subscribes and processes messages from topics
- **Broker**: Kafka server that stores and serves messages
- **Topic**: Logical stream where messages are categorized
- **Partition**: Topic split for scalability and parallelism
- **Zookeeper**: Coordinates brokers and manages cluster metadata (deprecated in newer Kafka versions)

3. How does Kafka ensure fault tolerance?

Answer:

- **Replication**: Each partition is replicated to multiple brokers
- **Leader & Followers**: One broker is leader, others are followers
- **ACKs**: Producers can specify acks (0, 1, all) for reliability

4. What is a Kafka topic?

Answer:
A topic is a category or feed name to which records are published. Topics are split into partitions for scalability.

5. How do Kafka producers work in Java?

Answer:
Producers use `KafkaProducer` class to send messages to a topic:

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
KafkaProducer<String, String> producer = new KafkaProducer<>(props);
producer.send(new ProducerRecord<>("topicName", "key", "value"));
```

6. How do Kafka consumers work in Java?

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test-group");
props.put("enable.auto.commit", "true");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("topicName"));

while (true) {
   ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
   for (ConsumerRecord<String, String> record : records) {
      System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());
   }
}
```

7. What is a Kafka partition and why is it important?

Answer:
Partitions allow Kafka to scale horizontally by splitting data. Each partition is an ordered sequence of records.

8. What is Kafka offset?

Answer:
Offset is a unique identifier for each record within a partition. Consumers use offsets to track their read position.

9. What is a consumer group?

Answer:
A group of consumers that coordinate to consume partitions of a topic collectively. Each partition is consumed by one member of the group.

10. How does Kafka handle message durability?

Answer:
Kafka persists messages to disk and replicates them across brokers. Durability is configurable using `acks` and `min.insync.replicas`.

11. What are Kafka acks in producers?

Answer:

- **acks=0**: No acknowledgment from broker
- **acks=1**: Leader acknowledges
- **acks=all**: Leader + all ISR replicas must acknowledge

12. What is ISR in Kafka?

Answer:
ISR (In-Sync Replica) is a set of replicas that are fully synced with the leader partition.

13. What happens if a Kafka broker goes down?

Answer:
A new leader is elected from the ISR. Consumers and producers can continue working if they are configured for retries and fault-tolerance.

14. What is Kafka retention policy?

Answer:
Defines how long Kafka retains messages. Controlled by:

- `retention.ms` (time-based)
- `retention.bytes` (size-based)

15. Difference between Kafka and traditional messaging systems (e.g., RabbitMQ)?

Answer:

- Kafka is **distributed**, **durable**, **horizontally scalable**, and supports **high-throughput streaming**
- RabbitMQ is **push-based**, uses **message queues**, suitable for **low-latency messaging**

16. How does Kafka achieve high throughput?

Answer:

- Batching messages
- Asynchronous processing
- Zero-copy (sendfile)
- Disk-based log storage
- Efficient compression

17. What serialization formats does Kafka support?

Answer:

- String (default)
- JSON
- Avro
- Protobuf
- Custom serializers via implementing `Serializer<T>` and `Deserializer<T>`

18. What is Kafka Streams?

Answer:
A client library for building stream processing applications. Offers filtering, windowing, joins, aggregation on Kafka topics.

9. Difference between Kafka Streams and Apache Flink/Spark?

Answer:

- Kafka Streams is **lightweight**, **embedded** in Java apps
- Spark/Flink are **external cluster-based processing engines**

20. How can you ensure message ordering in Kafka?

Answer:
Messages with the same **key** are sent to the same partition, ensuring **per-key order**.

21. What is idempotency in Kafka producers?

Answer:
It ensures that even if a message is sent multiple times (e.g., retries), it is written to Kafka only once. Set with `enable.idempotence=true`.

22. Can Kafka lose messages?

Answer:
If not properly configured (e.g., acks=0 or leader not in ISR), messages can be lost. Proper `acks`, retries, and replication mitigate this.

23. How do you handle backpressure in Kafka?

Answer:

- Use appropriate consumer poll rate
- Tune producer `linger.ms` and `batch.size`
- Apply throttling or rate-limiting

24. What is Kafka Connect?

Answer:
A tool for streaming data between Kafka and external systems (DBs, files, etc.) using source/sink connectors.

25. How do you monitor Kafka health?

Answer:
Using:

- JMX metrics
- Prometheus + Grafana
- Kafka Manager / Confluent Control Center
- Lag exporters (e.g., Burrow)

26. How do you commit offsets in consumers?

Answer:

- Automatically: `enable.auto.commit=true`
- Manually: Use `commitSync()` or `commitAsync()` for more control

27. How does Kafka handle exactly-once delivery?

Answer:
By combining:

- Idempotent producer
- Transactions (`transactional.id`, `initTransactions()`, `beginTransaction()`)
- Consumer offset commits in transactions

28. What is Kafka compaction?

Answer:
Log compaction ensures the latest value for each key is retained by deleting older records. Set `cleanup.policy=compact`.

29. What are common Kafka use cases in banking or fintech?

Answer:

- Real-time fraud detection
- Transaction stream processing
- Audit logging
- Payment event pipelines
- Data replication across services

30. How do you secure a Kafka cluster?

Answer:

- **Authentication**: SSL, SASL (Kerberos, PLAIN)
- **Authorization**: ACLs
- **Encryption**: SSL for data in transit
- **Audit logging**

Project: Order Processing System with Kafka (Microservices Architecture)

□ Overview:

We have 3 services communicating via Kafka:

1. **Order Service** – Publishes order events
2. **Inventory Service** – Listens to order events and updates stock
3. **Notification Service** – Sends confirmation messages

All services are decoupled and communicate via **Kafka topics**.

Kafka Topics Used:

Topic Name	Purpose
`order-events`	Order created events
`inventory-events`	Stock update notifications
`notification-events`	User notifications

Service 1: Order-Service

Dependency:

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

application.properties:

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
```

Producer Code:

```
@Service
public class OrderProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    private final String topic = "order-events";

    public void sendOrder(String orderJson) {
        kafkaTemplate.send(topic, orderJson);
    }
}
```

Controller:

```
@RestController

@RequestMapping("/api/orders")

public class OrderController {


    @Autowired

    private OrderProducer orderProducer;


    @PostMapping

    public ResponseEntity<String> placeOrder(@RequestBody OrderRequest
orderRequest) {

        String orderJson = new Gson().toJson(orderRequest);

        orderProducer.sendOrder(orderJson);

        return ResponseEntity.ok("Order placed!");

    }

}
```

Service 2: Inventory-Service (Consumer)

application.properties:

```
spring.kafka.bootstrap-servers=localhost:9092

spring.kafka.consumer.group-id=inventory-group

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
```

Consumer Code:

```
@Service

public class InventoryConsumer {


    @KafkaListener(topics = "order-events", groupId = "inventory-group")

    public void consumeOrder(String orderJson) {

        System.out.println("Inventory received order: " + orderJson);


        // Deserialize JSON and update stock

        // Then publish inventory status (optional)

    }

}
```

Service 3: Notification-Service (Consumer)

Consumer Code:

```
@Service
public class NotificationConsumer {
    @KafkaListener(topics = "order-events", groupId = "notification-group")
    public void consumeOrderForNotification(String orderJson) {
        System.out.println("Notification Service: Order Received - " + orderJson);
        // Send Email/SMS/Push Notification
    }
}
```

Data Flow:

```
POST /api/orders
        ↓
Order Service (produces to "order-events")
        ↓
-----------------------------
| Kafka Broker (order-events) |
-----------------------------
 ↓              ↓
Inventory Service   Notification Service
(consume & update)  (consume & notify)
```

Use Case Example: Banking (Funds Transfer)

Service	Role	Kafka Topic
TransferService	Publishes transfer events	`transfer-events`
BalanceService	Consumes and updates balance	`balance-events`
AuditService	Consumes for logging	`audit-events`
FraudDetectionService	Consumes for ML scoring	`fraud-events`

Folder Structure (Common)

└── order-service/

├── controller/

├── service/

├── model/

├── config/

└── KafkaProducer.java

└── inventory-service/

├── listener/

└── KafkaConsumer.java

└── notification-service/

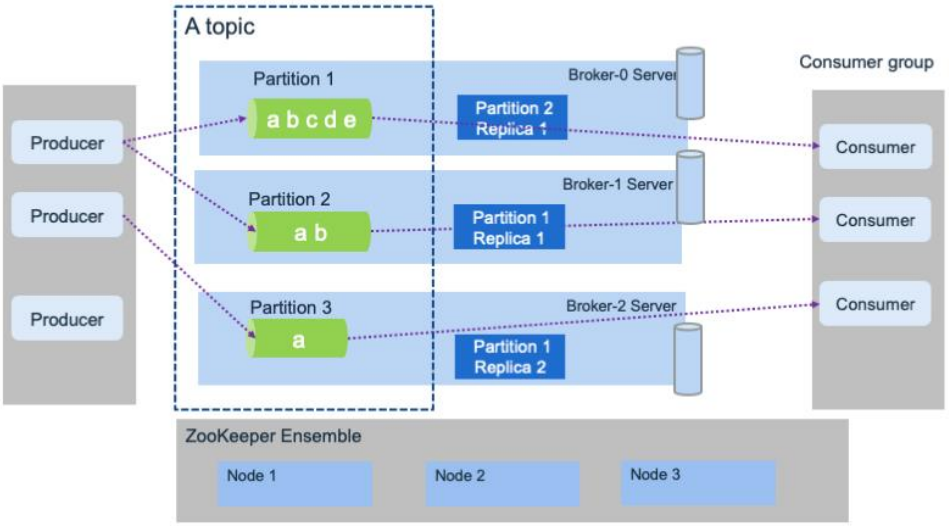
└── listener/

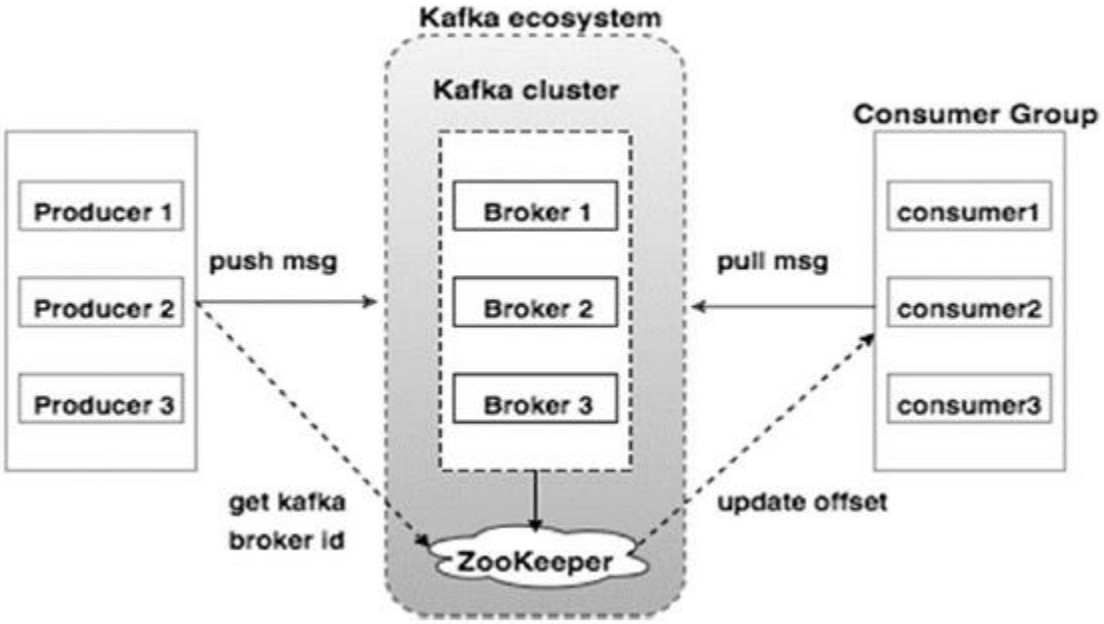
└── KafkaConsumer.java

Docker Compose for Kafka (Local Dev)

```
version: '3'

services:

  zookeeper:

    image: wurstmeister/zookeeper

    ports:

      - "2181:2181"


  kafka:

    image: wurstmeister/kafka

    ports:

      - "9092:9092"

    environment:

      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092

      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

Kafka Architecture





Kafka: Topics, Producers, and Consumers