Spring Framework 6

Beginner to Guru

Introduction to Apache Kafka

What is Apache Kafka?

- Distributed event streaming platform
- High-throughput, fault-tolerant, and scalable
- Used for building real-time data pipelines and streaming applications
- Fast, low latency messages

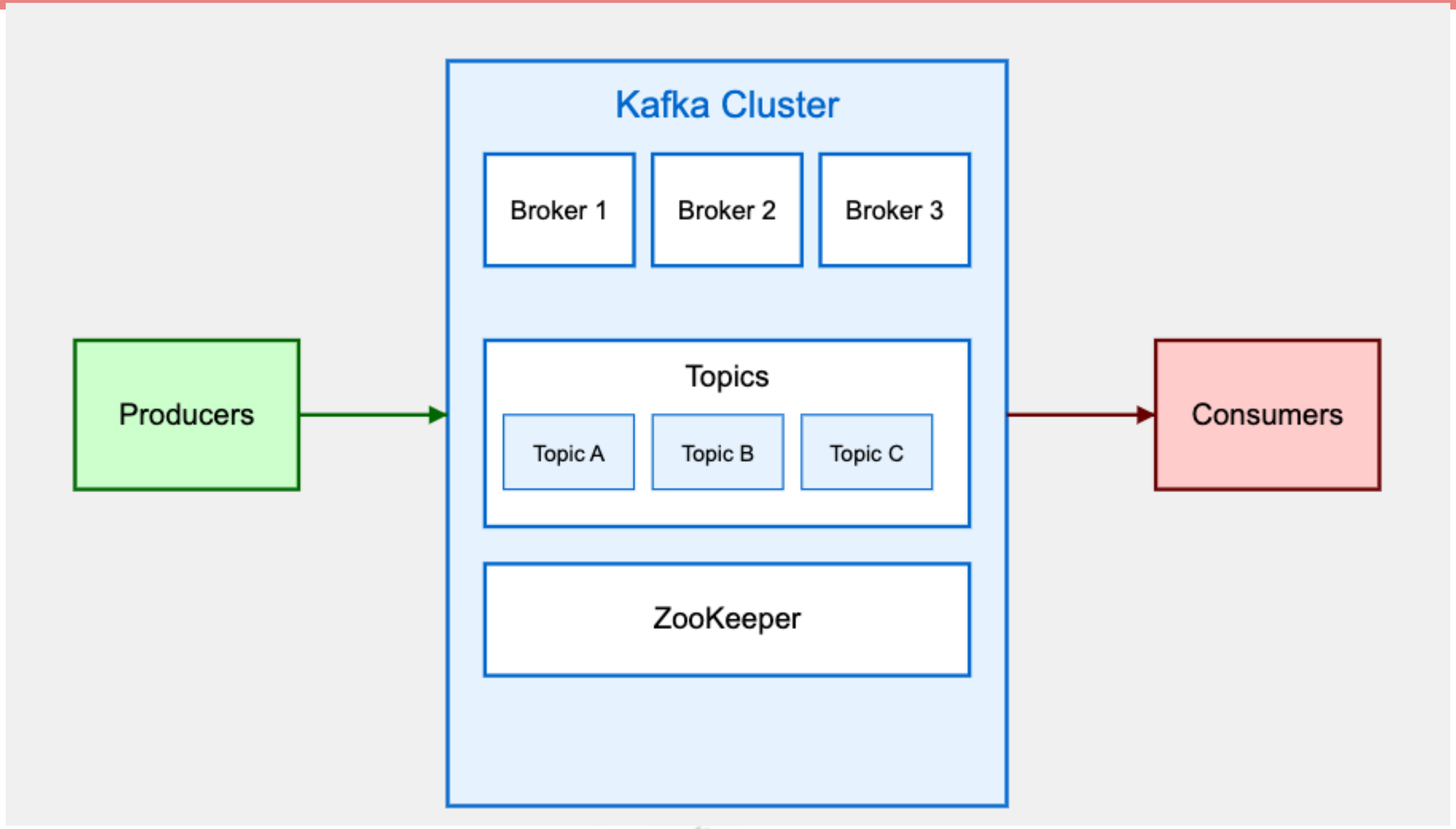
History of Kafka

- Developed by LinkedIn in 2010
- Open-sourced in 2011
- Became a top-level Apache project in 2012
- Widely adopted by thousands of companies for high-performance data pipelines, streaming analytics, and data integration

Core Concepts

- **Topics:** Categories for organizing and storing messages
- **Partitions:** Divisions of topics for parallel processing
- **Producers:** Applications that publish messages to topics
- **Consumers:** Applications that subscribe to topics and process messages
- **Brokers:** Servers that store and manage topics
- **ZooKeeper:** Manages cluster state and configuration (Note: Kafka is moving away from ZooKeeper dependency)
- **Kraft**: Kafka Raft - Replacement for ZooKeeper.

Kafka Architecture



Key Features

- **High throughput:** Can handle millions of messages per second
- **Scalability:** Easy to scale horizontally by adding more brokers
- **Fault tolerance:** Replication ensures data durability
- **Low latency:** Can achieve end-to-end latency of less than 10ms
- **Durability:** Messages are persisted on disk and replicated

Common Use Cases

- Messaging systems
- Activity tracking
- Metrics and logging
- Stream processing
- Event sourcing
- Commit logs

Integration with Java and Spring

- Native Java client libraries available
- Spring for Apache Kafka project
- Easy integration with Spring Boot applications
- Support for both imperative and reactive programming models

Conclusion

- Kafka is a powerful tool for building scalable, real-time data pipelines
- Essential technology in modern distributed systems
- Great fit for Java and Spring ecosystems
- Continuous development and growing community support

Next Steps

- Implement Microservices as previously outlined
- Create Common Messaging API Library
  - To be shared between services for code re-use (DRY Principle)
- Refactor Spring MVC Project to use Common Messaging Library
- Create Message Producer for Order Placed Event
- Testing with Embedded Kafka
- Create Drink Order Message
- Create Order Splitter

Next Steps

- Create Drinks Router
- Create Microservices
- Aggregate Complete Messages
- Publish Order Complete

SPRING FRAMEWORK

GURU