Event-driven system

A system architecture that responds to events generated by external or internal actions. It typically involves event producers and consumers, where events trigger actions asynchronously.

Hyperautomation

The use of advanced technologies, such as artificial intelligence (AI) and machine learning (ML), to automate processes beyond traditional automation capabilities.

Messaging system

A system that facilitates message-oriented communication between distributed applications or components, enabling asynchronous communication and decoupling of producers and consumers.

Streaming

The continuous and real-time transmission of data, typically used in contexts like data analytics, IoT, and multimedia applications.

Zookeeper and Kafka Integration

Zookeeper coordinates distributed systems like Kafka.
Zookeeper manages metadata, broker coordination, leader election, and partition management in Kafka.

The Role of Zookeeper

Zookeeper is a distributed coordination service.
It ensures synchronization, configuration management, and leadership election within distributed systems like Kafka.

Kafka Message

A unit of data in the Apache Kafka messaging system, which includes metadata and payload, facilitating high-throughput, low-latency data transport.

Kafka Message Key

A Kafka message key is a value used to determine the partition for a message.
Kafka hashes the message key to decide which partition the message will go to.

Kafka Message Offset

The offset is a unique identifier for a Kafka message within a partition.
Each message is assigned an offset, which consumers track to ensure no message is missed.

Kafka Partition

A portion of data stored in an Apache Kafka topic, allowing parallelism and scalability (horizontally) by distributing data across multiple brokers.

Kafka cluster

A group of Kafka brokers working together to manage topics, partitions, and data replication in the Apache Kafka distributed messaging system.

Kafka broker

A Kafka server responsible for handling requests from producers and consumers, storing and replicating data, and managing partitions.

Kafka Broker vs. Kafka Cluster

- A **Kafka Broker** is a single server that handles messages, while a **Kafka Cluster** is a group of brokers working together for scalability and redundancy.

Kafka Topic

A category or feed name to which messages are published by producers and from which messages are consumed by consumers.

Kafka Replication of Partitions in a Cluster

Kafka replicates each partition across multiple brokers for fault tolerance.
Each partition has one leader and multiple followers; if the leader fails, a follower takes over.

How to Size the Number of Kafka Brokers in a Kafka Cluster?

Sizing brokers is about determining the number of brokers needed for scalability and fault tolerance.
The number of brokers depends on throughput, disk space, replication, and fault tolerance requirements.

How to Size the Number of Kafka Partitions in a Kafka Topic?

Sizing partitions is about distributing data and balancing load across consumers.
More partitions allow parallel processing but increase overhead; size based on expected load and consumer needs.

Kafka Producer

An application or process that publishes messages to Kafka topics for consumption by consumers.

Kafka Consumer

An application or process that subscribes to Kafka topics to consume messages published by producers.

Kafka Consumer Group

A set of consumers that jointly consume a Kafka topic, dividing the workload and enabling parallel processing of messages.
Consumers share partitions, ensuring each partition is processed by one consumer at a time.

Kafka Consumer Group Rebalance

Kafka Consumer Group Rebalance occurs when the group or partitions change.
When a consumer joins, leaves, or partitions change, Kafka redistributes partitions among consumers.

Processing Rate between Kafka Producer and Kafka Consumer; Which One Has More Computational Consumption?

Refers to the rate at which Kafka producers and consumers handle messages.
Kafka consumers tend to have higher computational consumption due to message processing and offset management.

Kafka Commit Log

A Kafka commit log is an ordered, immutable sequence of messages stored in partitions.
The commit log is append-only and stores messages persistently for durability and replayability.

Microservice

A software architectural style where an application is composed of small, independent services that communicate over well-defined APIs.

REST API

Representational State Transfer – Application Programming Interface

Is a web service using HTTP for communication. It allows clients to perform CRUD operations on resources via HTTP methods like GET, POST, PUT, DELETE.

ESB

Enterprise Service Bus
Software architecture model providing fundamental services for complex, distributed enterprise applications.

SOA

Service-oriented architecture
A design approach where applications are composed of services that communicate through defined protocols and interfaces.

Middleware

A software layer that facilitates communication and data management between disparate applications, systems, and services.

On the one hand, both a software and a DevOps engineer would describe middleware as the layer that “glues” together software by different system components; on the other hand, a network engineer would state that middleware is the fault-tolerant and error-checking integration of network connections.

Enterprise Integration

Is the process of ensuring the interaction between enterprise entities necessary to achieve domain objectives

Enterprise Interoperability

The ability of diverse enterprise systems and software applications to work together without special effort on the part of the user.

MOM

Message Oriented Middleware
Software that enables message-oriented communication between applications or systems.

AMQP

Advanced Message Queuing Protocol
An open standard for messaging middleware, designed to enable the communication between applications.

Publish/Subscribe - messaging pattern

A messaging pattern where publishers categorize messages into topics without specifying recipients, allowing subscribers to receive messages based on their interests.

Is a pattern that is characterized by the sender (publisher) of a piece of data (message) not specifically directing it to a receiver. Instead, the publisher classifies the message somehow, and that receiver (subscriber) subscribes to receive certain classes of messages. - TODO

Reactor pattern

A design pattern that handles multiple concurrent requests or messages with a single thread, associating I/O events with event handlers for efficient processing.
Give the possibility to handle multiple concurrent requests or messages with a single thread. The reactor pattern allows associating I/O events with event handlers. Invokes the event handlers when the expected event is received. Avoiding the creation of a thread for each message, request and connection
It uses an event loop to dispatch events to handlers, avoiding thread creation for each event.

Proactor pattern

Can be seen as an asynchronous version of the reactor, it is useful when long-running event handlers invoke a continuation when they complete. Such mechanisms allow mixing nonblocking and blocking I/O

The Proactor pattern handles long-running, blocking I/O operations asynchronously. It initiates operations and uses callbacks to handle results when the operation completes, ensuring non-blocking execution.

Reactor Pattern vs. Proactor Pattern

- The **Reactor Pattern** uses an event loop to handle I/O events synchronously, while the **Proactor Pattern** initiates asynchronous I/O operations and handles them upon completion.

What Are the Four Properties Required for a System to Be Considered a Reactive System?

A reactive system is responsive, resilient, elastic, and message-driven.
These properties ensure that the system can scale, recover from failures, and remain responsive to events.

Quarkus Non-blocking Database with Pipelining

Quarkus provides asynchronous, non-blocking database access.
It uses reactive programming to send multiple database queries without waiting for each response, improving throughput.

Quarkus Mutiny

Mutiny is a reactive programming library for handling async operations in Quarkus.
It simplifies reactive patterns like `Uni` (single value) and `Multi` (stream of values) for non-blocking operations.

Quarkus Unification of Reactive and Imperative Models

Quarkus allows mixing reactive and imperative programming models.
Both models can be used in a single application, with Quarkus handling integration for seamless operation.

MOM: Time Decoupling vs. Asynchronous Communications

Time Decoupling separates the sender and receiver in time, allowing the message to be processed later, while **Asynchronous Communication** ensures no waiting for an immediate response from the receiver but doesn't necessarily decouple the timing of activities.

Kafka: Fire-and-Forget vs. Synchronous Messages vs. Asynchronous Messages

- **Fire-and-Forget** involves sending a message without waiting for a response, while **Synchronous Messages** require an immediate acknowledgment, and **Asynchronous Messages** allow sending without blocking the sender and acknowledging later.

Blocking Network I/O vs. Multithreaded Blocking Network I/O vs. Non-Blocking Network I/O

- **Blocking Network I/O** makes a thread wait for I/O to complete, **Multithreaded Blocking I/O** uses multiple threads for separate I/O tasks, and **Non-Blocking Network I/O** allows a thread to continue processing without waiting for I/O to finish.