**✅ Kafka Interview Questions and Answers**

**Q1. What is Apache Kafka and how does it work?**  
**A:** Kafka is a distributed event streaming platform that works as a message broker between producers and consumers. It uses a publish-subscribe model where:

* Producers send data to **topics**
* Topics are divided into **partitions**
* Consumers read from partitions sequentially  
  Kafka ensures durability, scalability, and fault tolerance.

**Cross Question:**

* How does Kafka ensure high throughput and durability?

Through sequential disk I/O and replication of partitions.

**Q2. Difference between Kafka and traditional message brokers like RabbitMQ?**  
**A:**

* Kafka stores messages for a configurable time even after consumption.
* Kafka supports high-throughput, distributed log storage.
* RabbitMQ uses push-based consumption; Kafka uses pull-based.

**Q3. What is a Kafka partition and why is it important?**  
**A:**  
Partitioning allows Kafka to parallelize message handling. Each partition can be consumed independently for scalability and fault tolerance.

**Cross Question:**

* What are the implications of increasing partitions?

It increases parallelism but may affect message ordering and throughput.

**Q4. What is Kafka consumer group?**  
**A:** A group of consumers that share the workload of reading from a topic. Each partition is assigned to only one consumer in the group.

**✅ Zookeeper Interview Questions and Answers**

**Q1. What is Zookeeper and why does Kafka need it?**  
**A:** Zookeeper is a distributed coordination service used by Kafka for:

* Maintaining broker metadata
* Leader election
* Keeping track of topic configurations

**Cross Question:**

* Can Kafka run without Zookeeper?

Kafka newer versions (2.8+) support **KRaft mode** (Kafka Raft Metadata mode) which eliminates the need for Zookeeper.

**Q2. What is ZNode in Zookeeper?**  
**A:** A ZNode is a data node in Zookeeper's hierarchical namespace. It can be persistent or ephemeral.

**Q3. How does Zookeeper handle failure?**  
**A:**  
Through leader election. Zookeeper nodes use a quorum-based mechanism to elect a new leader if the current one fails.

**✅ Eureka Interview Questions and Answers**

**Q1. What is Netflix Eureka?**  
**A:** A REST-based service registry for load balancing and failover of middle-tier servers. Part of the Netflix OSS stack.

**Q2. How does Eureka service discovery work?**  
**A:**

* Clients register themselves with the Eureka server.
* Clients query the registry to discover other services.
* It uses heartbeats to verify availability.

**Q3. What happens when Eureka server goes down?**  
**A:**  
Eureka clients use **self-preservation mode** to retain their local registry cache and continue working temporarily.

**Cross Question:**

* How do you ensure Eureka high availability?

Deploy Eureka in a **cluster with multiple nodes** and use client-side load balancing via Ribbon.

**✅ Zipkin Interview Questions and Answers**

**Q1. What is Zipkin and how does it work?**  
**A:** Zipkin is a distributed tracing system. It collects timing data from microservices to visualize request flow and latencies.

**Q2. What data does Zipkin collect?**  
**A:**

* TraceId
* SpanId
* ParentSpanId
* Timestamps (start, end)
* Annotations and tags

**Q3. How do you integrate Zipkin with Spring Boot?**  
**A:** Add the dependency:

xml

CopyEdit

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zipkin</artifactId>

</dependency>

Then configure:

yaml

CopyEdit

spring:

zipkin:

base-url: http://localhost:9411

**✅ Sleuth Interview Questions and Answers**

**Q1. What is Spring Cloud Sleuth?**  
**A:** Sleuth adds tracing information (like Trace ID and Span ID) into logs to help track the flow of requests in microservices.

**Q2. How does Sleuth work with Zipkin?**  
**A:** Sleuth automatically creates spans and sends them to Zipkin for visualization.

**Cross Question:**

* How do Sleuth and MDC (Mapped Diagnostic Context) work together?

Sleuth populates trace IDs into MDC so that logs across services can be correlated.

**✅ Spring Boot Actuator Interview Questions and Answers**

**Q1. What is Spring Boot Actuator?**  
**A:** A set of production-ready features for monitoring and managing a Spring Boot application, like /health, /metrics, /info.

**Q2. How do you secure actuator endpoints?**  
**A:**

* Use Spring Security to restrict access
* Limit endpoints via properties:

yaml

CopyEdit

management:

endpoints:

web:

exposure:

include: health, info

**Q3. How can you expose custom metrics with Actuator?**  
**A:** Use MeterRegistry bean from Micrometer to create custom counters, gauges, etc.

**Q4. Can you explain how health checks work in Actuator?**  
**A:**  
Spring Boot Actuator uses built-in and custom HealthIndicator beans to determine the application status.

**➕ Bonus: Common Microservice Question**

**Q: How do Kafka, Eureka, Sleuth, Zipkin, and Actuator work together in a Spring Boot microservices ecosystem?**  
**A:**

* **Kafka** enables async communication between services
* **Eureka** handles service discovery
* **Sleuth** adds trace context to each request
* **Zipkin** collects trace data and visualizes it
* **Actuator** provides health and metrics endpoints  
  All of them together provide observability, scalability, and resilience in distributed microservices.

Let me know if you want a **PDF version**, **coding exercises**, or **scenario-based design questions** for these tools.

**You said:**

scenario-based design questions for these tools.

**ChatGPT said:**

Here are **scenario-based system design interview questions** for the tools you listed: **Kafka, Zookeeper, Eureka, Zipkin, Sleuth, and Spring Boot Actuator**. These questions test **real-world problem-solving, architecture decisions, trade-offs, and integration knowledge** — ideal for 12+ years of experience.

**🧠 Kafka Scenario-Based Questions**

**Q1: Event-Driven Order Processing System**

**Scenario:** You’re building an e-commerce system. Orders placed by users should be processed asynchronously to update inventory, notify shipping, and send confirmation emails.  
**Design Challenge:**

* How will you use Kafka to decouple these services?
* How will you ensure order message delivery even if a service crashes?
* What strategy will you use for exactly-once or at-least-once delivery?

**Follow-ups:**

* How will you handle message reprocessing?
* How to scale the system if orders spike to 50,000/min?

**Q2: Kafka for Logging Microservices**

**Scenario:** Your company wants to centralize logs from multiple microservices for auditing and analysis.  
**Design Challenge:**

* How can Kafka be used for log aggregation?
* How would you partition logs across topics?

**Follow-ups:**

* How to ensure logs maintain timestamp order?
* What tools can you use to consume and analyze the logs?

**🧠 Zookeeper Scenario-Based Questions**

**Q1: Leader Election Between Microservices**

**Scenario:** You have multiple instances of a Spring Boot batch service but only one should run the job at a time.  
**Design Challenge:**

* How can you use Zookeeper for leader election?
* What happens if the leader crashes?

**Follow-ups:**

* How do ephemeral znodes help in this situation?
* What happens if Zookeeper cluster splits?

**🧠 Eureka Scenario-Based Questions**

**Q1: Auto-Scaling Microservices with Eureka**

**Scenario:** Your app runs on Kubernetes with auto-scaling enabled. When traffic spikes, new instances spin up.  
**Design Challenge:**

* How does Eureka help the new instances get discovered?
* How would you configure the heartbeat and instance deregistration?

**Follow-ups:**

* What happens if Eureka Server is down?
* How do you handle stale instances in the registry?

**Q2: Multi-Region Eureka Setup**

**Scenario:** You’re deploying services in multiple data centers. Each region has its own Eureka server.  
**Design Challenge:**

* How would you enable service discovery across regions?
* What is the trade-off between availability and consistency?

**🧠 Zipkin & Sleuth Scenario-Based Questions**

**Q1: Debugging Latency in Microservices Chain**

**Scenario:** A frontend API call is taking 12 seconds. The request passes through 6 microservices.  
**Design Challenge:**

* How will Sleuth and Zipkin help identify the latency bottleneck?
* How do you ensure trace context is propagated across HTTP/gRPC calls?

**Follow-ups:**

* How will you trace messages across Kafka with Sleuth?
* What happens if Zipkin is temporarily down? Will tracing still work?

**Q2: Sampling Strategy**

**Scenario:** Tracing all requests is causing performance issues in prod.  
**Design Challenge:**

* How would you adjust the sampling rate in Sleuth?
* Which types of requests would you always trace?

**Cross:**

* How to selectively trace only failed or slow transactions?

**🧠 Spring Boot Actuator Scenario-Based Questions**

**Q1: Monitoring and Alerts**

**Scenario:** You want to monitor health and metrics for 50+ microservices.  
**Design Challenge:**

* Which Actuator endpoints will you enable?
* How will you expose metrics to Prometheus/Grafana?

**Follow-ups:**

* How to restrict access to sensitive endpoints?
* How to expose custom business-level metrics (e.g., “orders failed”)?

**Q2: Auto-Restart on Health Check Failure**

**Scenario:** You want a service to restart automatically if /health returns DOWN.  
**Design Challenge:**

* How would you configure readiness/liveness probes (in Kubernetes) using Actuator?
* How do you prevent flapping due to transient failures?

**🧩 Full-System Design Challenge: Combine All Tools**

**Q: Design an Observability-First Microservice Architecture**

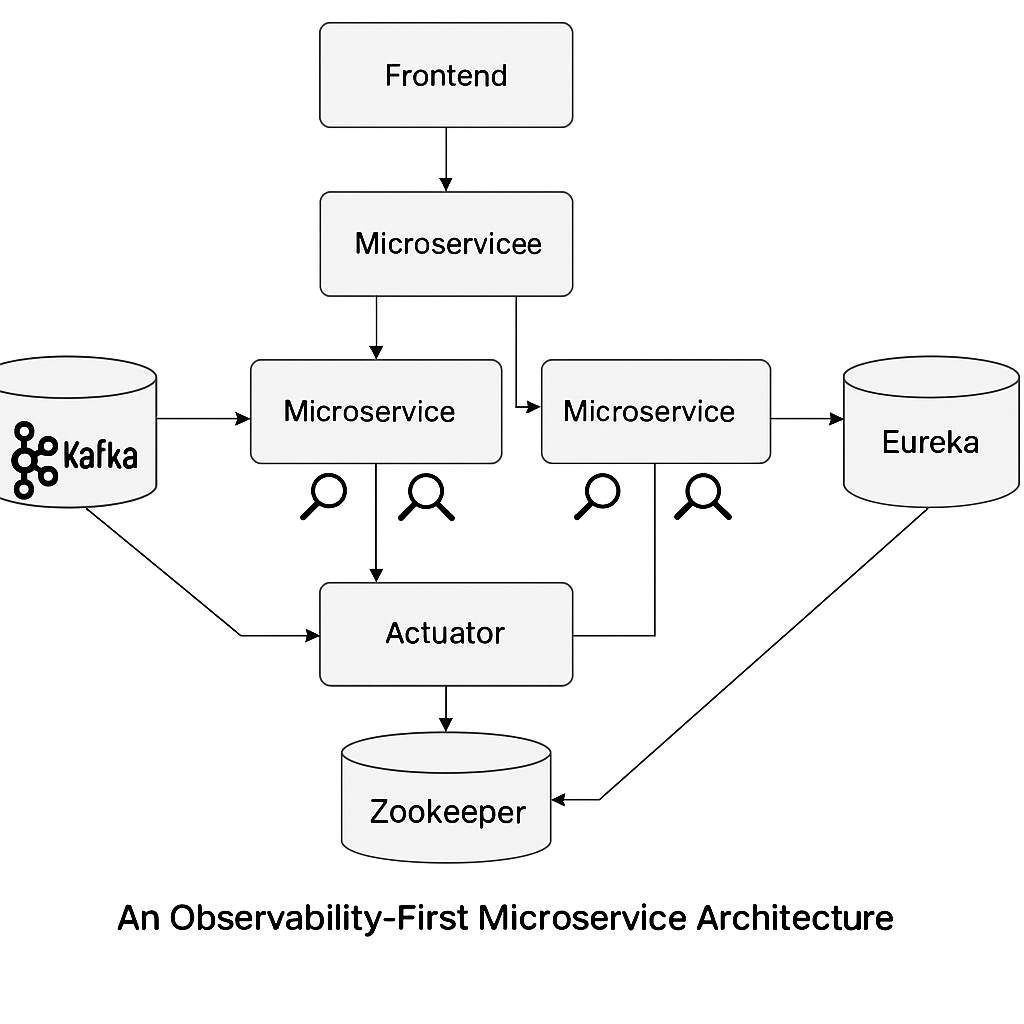
**Scenario:** You're building a critical financial transaction system. You need scalability, reliability, and observability.

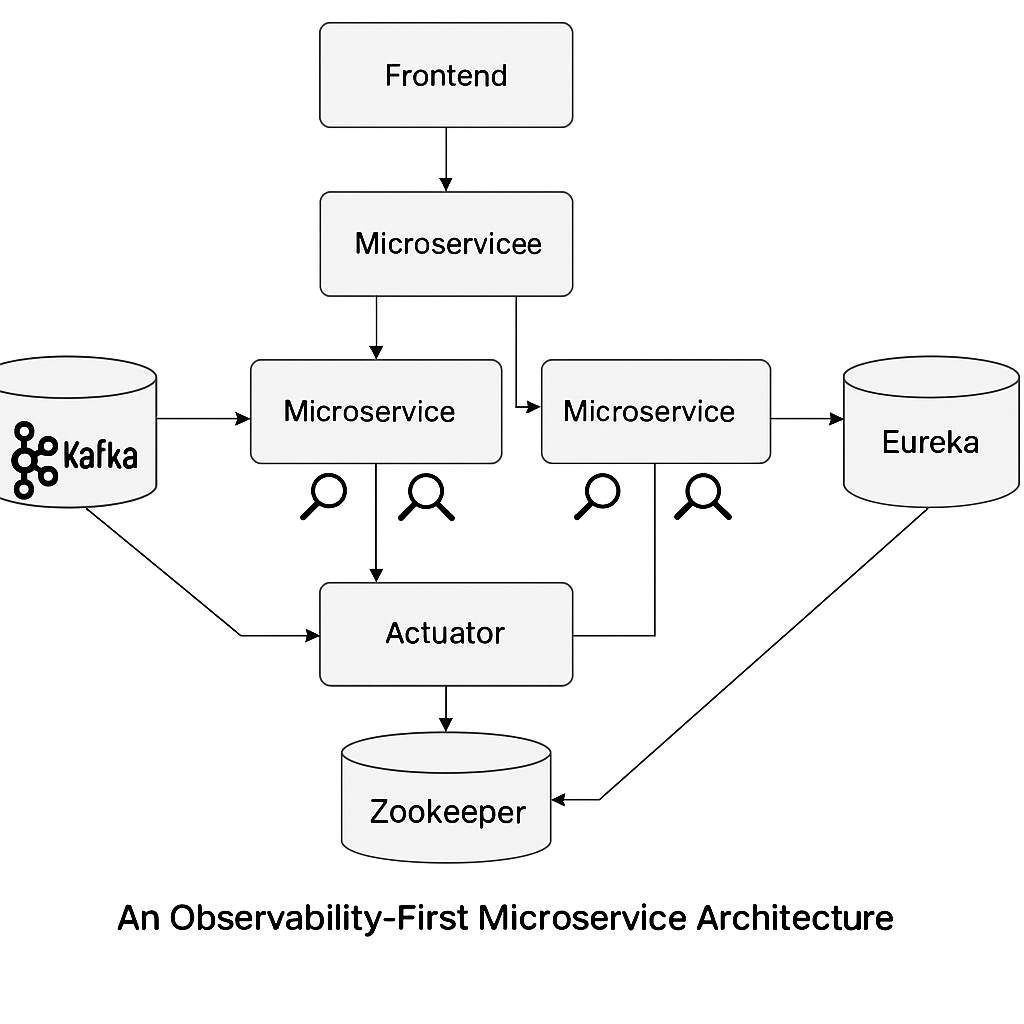
**Design Challenge:**

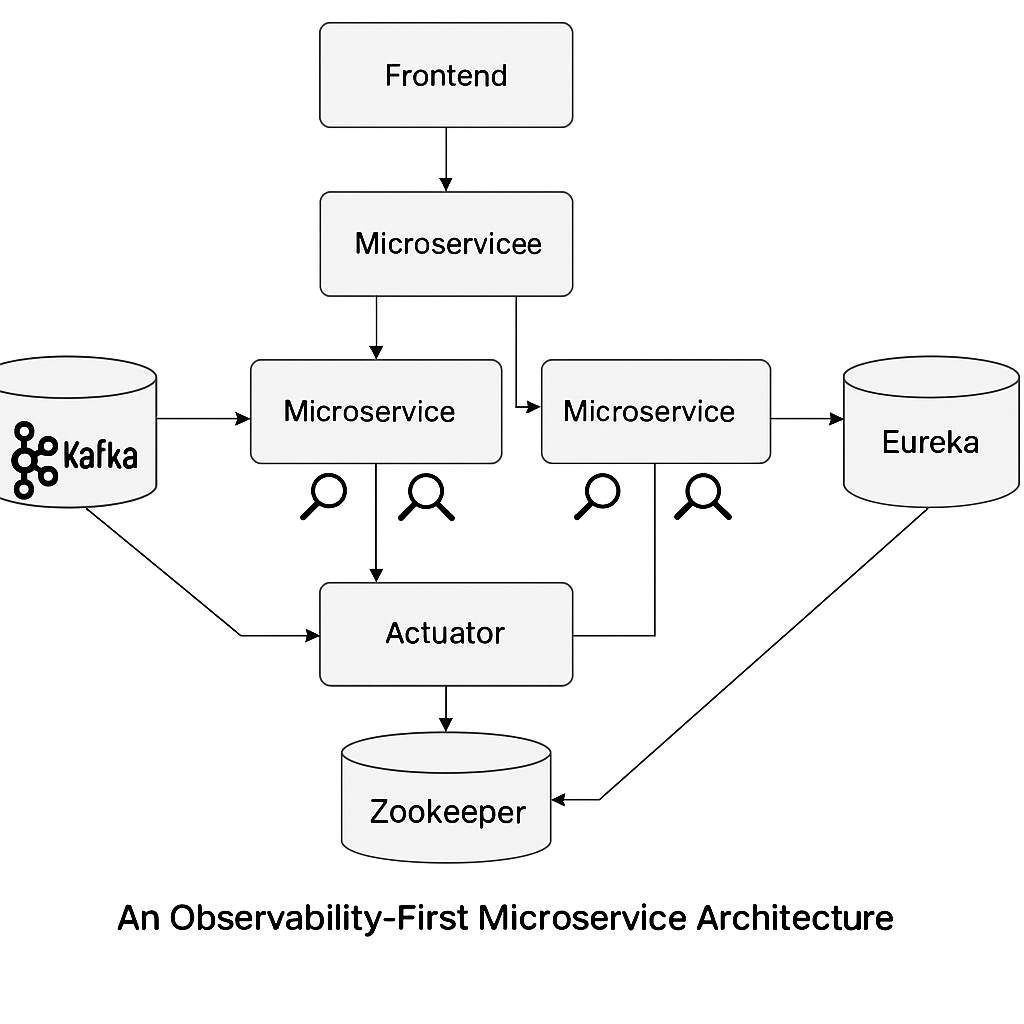
* Use Kafka for async messaging between services.
* Use Eureka for service discovery.
* Use Sleuth + Zipkin for tracing.
* Use Actuator for monitoring and health.
* Use Zookeeper for coordination (e.g., Kafka or leader election).

**Follow-ups:**

* How do you ensure messages are not lost?
* How do you trace a failed transaction from frontend to DB?
* How do you deal with partial failures and retries?







**✅ Design Challenge Solution**

**🔁 1. Kafka – Asynchronous Messaging**

* **Producers (Microservices)** publish domain events (e.g., OrderPlaced) to **Kafka topics**.
* **Consumers (other Microservices)** subscribe and process events independently (e.g., Inventory, Shipping).
* Use **Kafka partitions** for scaling and **consumer groups** for parallelism.
* Messages are persisted until explicitly purged (retention.ms), ensuring durability.

**🧭 2. Eureka – Service Discovery**

* All microservices **register with Eureka Server** on startup.
* They **query Eureka** for locating other services (load-balanced via Ribbon or RestTemplate/WebClient with discovery client).
* Eureka’s **self-preservation mode** avoids flapping during brief outages.

**📊 3. Spring Boot Actuator – Monitoring**

* Expose endpoints like /actuator/health, /metrics, /info, /loggers.
* Integrate with **Prometheus** or **New Relic** for scraping metrics.
* **Liveness/readiness probes** in Kubernetes use Actuator for restarts and traffic routing.

**🔍 4. Sleuth + Zipkin – Tracing**

* **Spring Cloud Sleuth** adds tracing IDs (Trace ID, Span ID) to logs and headers.
* These IDs are automatically **propagated across HTTP and Kafka**.
* **Zipkin collector** stores trace data for visualization.
* Helps visualize full trace from **frontend → API Gateway → microservices → DB**.

**🧠 5. Zookeeper – Coordination**

* Used by **Kafka** for broker coordination, partition leader election, and topic metadata.
* Can also be used for **leader election** in distributed tasks (e.g., batch jobs).
* Ensures only one microservice node runs a specific job at a time.

**🔁 Follow-Up Questions & Solutions**

**⚠️ Q1: How do you ensure messages are not lost?**

**Solution:**

* **Kafka configuration:**
  + acks=all → producer waits for all ISR (in-sync replicas)
  + retries and enable.idempotence=true → prevents duplicate records
  + min.insync.replicas → ensures quorum for durability
* **Consumers:**
  + Use **manual offset commit** after successful processing
  + Store **offsets in Kafka (or DB)** to avoid reprocessing on restart
* **DLQ (Dead Letter Queue):** Failed messages are routed to a DLQ topic for retry or manual handling.

**🔍 Q2: How do you trace a failed transaction from frontend to DB?**

**Solution:**

* **Sleuth** adds Trace-ID and Span-ID in every HTTP/Kafka call.
* **Logs across services** include these IDs; centralized logging (ELK, Loki) allows you to search all logs by trace ID.
* **Zipkin UI** shows full flow of transaction, with breakdown:
  + Frontend → API Gateway → Order Service → Kafka → Inventory Service → DB
  + Latency and failure in any segment are visualized.

**🔄 Q3: How do you deal with partial failures and retries?**

**Solution:**

* **Circuit Breakers (e.g., Resilience4j):**
  + Prevent cascading failures
  + Fall back or fail fast
* **Retries with backoff:**
  + Use @Retryable or Kafka retry logic
* **Idempotent operations:**
  + Ensure repeated messages/requests don't cause side effects
* **Compensation transactions / SAGA pattern:**
  + Reverse operations when part of a distributed transaction fails
* **Timeouts & fallback methods:**
  + Set connection/read timeouts to avoid indefinite hangs