**1. Streaming Architecture**

**Q1. What are the key components of a streaming architecture?**  
**A1.** Input sources (Kafka, socket, files), processing engine (Spark Streaming), and output sinks (databases, dashboards, storage).

**Q2. Difference between micro-batch and real-time processing in Spark Streaming?**  
**A2.** Micro-batch divides data into small time intervals for processing; real-time processes events as they arrive (low latency).

**Q3. Explain processing modes in Spark Structured Streaming.**  
**A3.**

* **Append mode:** Only new rows added.
* **Update mode:** Updates existing rows with late data.
* **Complete mode:** Recomputes the entire result table.

**Q4. What is watermarking in Spark Streaming?**  
**A4.** A technique to handle late-arriving data by specifying how long Spark should wait for late events before dropping them.

**2. Basic Streaming Operations**

**Q5. What’s the difference between stateless and stateful operations?**  
**A5.** Stateless ops work only on the current batch (e.g., map, filter); stateful ops maintain data across batches (e.g., reduceByKeyAndWindow).

**Q6. How do you monitor a Spark Streaming application?**  
**A6.** Using Spark UI, logs, metrics (via Graphite, Prometheus), and external monitoring tools like Ganglia.

**Q7. What are output modes in Structured Streaming?**  
**A7.** Append, Update, Complete.

**Q8. Give an example of a simple streaming operation in Spark.**  
**A8.** Word count from a socket stream:

val lines = spark.readStream.format("socket").option("host","localhost").option("port",9999).load()

val words = lines.as[String].flatMap(\_.split(" "))

val counts = words.groupBy("value").count()

**3. Types of Windows**

**Q9. Explain tumbling windows.**  
**A9.** Fixed-size, non-overlapping windows (e.g., count events every 5 minutes).

**Q10. Explain sliding windows.**  
**A10.** Windows of fixed length that overlap (e.g., 5 min window sliding every 1 min).

**Q11. What is a session window?**  
**A11.** Windows defined by a period of inactivity; ends when no events arrive within a gap duration.

**4. Advanced Streaming (Stateful Ops)**

**Q12. How does Spark handle stateful transformations?**  
**A12.** Using operations like updateStateByKey and mapGroupsWithState, which maintain running aggregates.

**Q13. How do stream-stream joins work in Spark?**  
**A13.** Spark buffers data from both streams (using watermarking) and joins records falling within time boundaries.

**Q14. How do you manage state in Spark Structured Streaming?**  
**A14.** With state stores backed by checkpointing (e.g., RocksDB-like storage), using mapGroupsWithState or flatMapGroupsWithState.

**5. Complex Streaming**

**Q15. How do you handle deduplication in Spark Streaming?**  
**A15.** Use event-time + unique ID combination with dropDuplicates.

**Q16. What techniques are used for late data handling?**  
**A16.** Watermarking, delay thresholds, reprocessing with backfill pipelines.

**Q17. What is checkpointing in Spark Streaming?**  
**A17.** A mechanism to persist metadata and state so the application can recover after failure.

**Q18. What are exactly-once guarantees in Spark Streaming?**  
**A18.** Spark ensures exactly-once output when using idempotent sinks (like Kafka transactional writes) and checkpointing.

**6. Performance Analysis**

**Q19. How do you optimize Spark Streaming jobs?**  
**A19.** Tune batch interval, enable caching, optimize partitions, avoid shuffles, and use broadcast joins.

**Q20. How do you identify bottlenecks in a Spark job?**  
**A20.** Using Spark UI to analyze stages, shuffle, task execution, and GC times.

**Q21. How do you allocate resources for Spark Streaming?**  
**A21.** Configure executors, cores, and memory per workload. Example:

--executor-memory 4G --executor-cores 2 --num-executors 5

**Q22. What role does backpressure play in Spark Streaming?**  
**A22.** It automatically adjusts data ingestion rate based on system processing capacity.

**7. Memory Optimization**

**Q23. How do you debug memory issues in Spark?**  
**A23.** Use Spark UI (storage tab), heap dumps, GC logs, and structured logging.

**Q24. How does caching impact Spark Streaming performance?**  
**A24.** Speeds up iterative computations but must be managed to prevent memory overflow.

**Q25. What’s the difference between serialization methods (Java vs Kryo)?**  
**A25.** Kryo is faster and more memory-efficient than default Java serialization.

**Q26. How do you avoid OOM errors in Spark Streaming?**  
**A26.** Optimize batch size, checkpointing, data pruning, and memory configurations.

**8. Project Development**

**Q27. How would you design a real-time fraud detection pipeline?**  
**A27.** Ingest from Kafka → preprocess in Spark Streaming → apply ML model → store alerts in DB → notify via API.

**Q28. What’s the best practice for error handling in Spark Streaming?**  
**A28.** Implement retry mechanisms, dead-letter queues, checkpointing, and alerting systems.

**Q29. How do you ingest streaming data into Spark from multiple sources?**  
**A29.** Using structured streaming connectors (Kafka, Kinesis, Socket, File) and unioning them.

**Q30. How would you implement real-time analytics dashboards?**  
**A30.** Stream → Spark Aggregation → Push to ElasticSearch / Cassandra → Visualize in Kibana/Grafana.

**9. Architecture Patterns**

**Q31. What is an event-driven architecture?**  
**A31.** A design where services communicate asynchronously via events (e.g., Kafka + Spark).

**Q32. Compare Lambda and Kappa architecture.**  
**A32.** Lambda: batch + stream pipelines; Kappa: only streaming pipeline (simpler, avoids code duplication).

**Q33. When would you prefer batch over streaming?**  
**A33.** Batch is better for historical analysis, large-scale ETL, and when low latency is not required.

**Q34. Real-time vs near real-time processing?**  
**A34.** Real-time: sub-second latency (fraud detection). Near real-time: seconds to minutes (analytics dashboards).

**10. Scenario-Based / Advanced Qs**

**Q35. How would you handle a slow sink in Spark Streaming?**  
**A35.** Use write-ahead logs, buffering, scaling sink, or backpressure.

**Q36. What happens if Spark Streaming job crashes?**  
**A36.** It restarts from checkpoint (recovering offsets, state).

**Q37. How do you ensure idempotent writes in sinks?**  
**A37.** Use unique keys, upserts, or transactional sinks (Kafka transactions, Delta Lake).

**Q38. How do you handle schema evolution in streaming data?**  
**A38.** Use schema registry (Confluent), handle optional fields, and manage backward/forward compatibility.

**Q39. Explain structured streaming triggers.**  
**A39.**

* **Default (micro-batch)**: runs as soon as new data arrives.
* **Fixed interval**: e.g., every 10 sec.
* **Continuous processing**: low-latency mode.

**Q40. How would you implement sessionization in Spark?**  
**A40.** Using session windows with gap duration to group user events.

**11. Deeper Technical Qs**

**Q41. Difference between foreachBatch and foreach in Structured Streaming?**  
**A41.** foreachBatch: operates on each micro-batch DataFrame. foreach: operates on individual records.

**Q42. What is watermark + window combination used for?**  
**A42.** To bound state size while still processing late data.

**Q43. How do you configure parallelism in Spark Streaming?**  
**A43.** Tune spark.streaming.concurrentJobs, number of partitions, and executor cores.

**Q44. How do you scale a Spark Structured Streaming job?**  
**A44.** Add more executors, repartition input data, optimize joins, and enable autoscaling.

**Q45. What is backpressure in Kafka + Spark Streaming?**  
**A45.** Mechanism to control ingestion rate based on consumer lag.

**12. Tricky & Implementation Qs**

**Q46. Can Spark Streaming guarantee zero data loss?**  
**A46.** Yes, with checkpointing + write-ahead logs + reliable sources (like Kafka).

**Q47. How would you implement deduplication across multiple micro-batches?**  
**A47.** Use state store with unique IDs + watermark to clean duplicates.

**Q48. How would you reduce shuffle in Spark Streaming?**  
**A48.** Use partitioning, bucketing, broadcast joins, and map-side aggregation.

**Q49. How does Spark handle unbounded streaming data?**  
**A49.** By breaking it into micro-batches, applying windows, and bounding state with watermarks.

**Q50. What are some common Spark Streaming production issues?**  
**A50.** Out-of-memory errors, checkpoint corruption, skewed partitions, late data, and sink throughput bottlenecks.

**1. Kafka Integration Basics**

1. **Q:** What is Apache Kafka and why is it used in real-time systems?  
   **A:** Kafka is a distributed event streaming platform used for high-throughput, fault-tolerant, and scalable real-time data pipelines. It decouples producers and consumers, making it ideal for streaming analytics.
2. **Q:** Explain Kafka topics, partitions, and offsets.  
   **A:** A topic is a logical category for messages. Each topic is split into partitions (log files) for parallelism. Offsets are sequential IDs within partitions, representing message positions.
3. **Q:** Difference between Kafka Producer and Consumer.  
   **A:** Producer publishes messages to topics, consumers subscribe to topics and process messages. Consumers track offsets to ensure message delivery.
4. **Q:** What is a Consumer Group in Kafka?  
   **A:** A consumer group is a set of consumers that share load for a topic. Each partition is consumed by only one consumer in the group.
5. **Q:** How do you integrate Kafka with Spark Streaming?  
   **A:** By using Spark Structured Streaming + Kafka connector (readStream.format("kafka")). Spark consumes Kafka topics in micro-batches or continuous mode.

**2. Advanced Kafka Features**

1. **Q:** What is Exactly-Once Processing (EOS) in Kafka?  
   **A:** EOS ensures messages are processed once, even during retries. Achieved using **idempotent producers + transactions + offset commits**.
2. **Q:** What is Schema Evolution in Kafka with Avro?  
   **A:** Schema Registry allows backward/forward compatibility. Producers write data with a schema ID, consumers fetch schema to deserialize even if schema evolves.
3. **Q:** Kafka Streams vs Spark Streaming – key differences?  
   **A:** Kafka Streams: library, lightweight, event-at-a-time, tightly coupled with Kafka.  
   Spark Streaming: cluster framework, micro-batch/continuous, supports multiple sources, advanced analytics.
4. **Q:** What is idempotent producer in Kafka?  
   **A:** Ensures no duplicate messages are written during retries. Kafka assigns a Producer ID (PID) and sequence number to guarantee uniqueness.
5. **Q:** How does Kafka handle backpressure?  
   **A:** By controlling consumer poll rates, producer linger.ms, batching, and Spark backpressure mechanism (spark.streaming.backpressure.enabled=true).

**3. Complex Integration Patterns**

1. **Q:** How do you consume from multiple Kafka topics in Spark?  
   **A:** Provide a comma-separated topic list or regex in subscribe option when defining the Kafka source.
2. **Q:** How do you handle errors in Kafka-Spark pipelines?  
   **A:** Retry policies, dead letter queues (DLQ), storing failed events in another Kafka topic or HDFS.
3. **Q:** What is message serialization in Kafka?  
   **A:** Data encoding before sending (JSON, Avro, Protobuf). Kafka requires serializers/deserializers (SerDes) for producers and consumers.
4. **Q:** Difference between Avro, Protobuf, and JSON for Kafka messages?  
   **A:** JSON (human-readable, no schema), Avro (binary, compact, schema-based), Protobuf (binary, language-neutral, strong schema evolution).
5. **Q:** How to ensure compatibility when multiple consumers need different schemas?  
   **A:** Use **Schema Registry** with compatibility rules (backward, forward, full).

**4. Advanced Processing Concepts**

1. **Q:** What is Complex Event Processing (CEP) in streaming?  
   **A:** Detecting meaningful patterns across multiple events (e.g., fraud detection, anomaly detection).
2. **Q:** What is Stream Enrichment in Spark Streaming?  
   **A:** Enhancing Kafka messages by joining with external datasets (e.g., lookup DB, broadcast variables, or APIs).
3. **Q:** Explain pattern detection in streaming data.  
   **A:** Using CEP libraries (Flink CEP, Spark Structured Streaming with stateful operations) to detect sequences or anomalies.
4. **Q:** What are Temporal Operations in Spark Streaming?  
   **A:** Operations based on event-time (not ingestion-time), using watermarks for late-arriving data handling.
5. **Q:** Difference between Event-time and Processing-time in Spark Streaming.  
   **A:** Event-time is when the event occurred; processing-time is when Spark processes it. Event-time + watermarks give accurate aggregations.

**5. Real-Time Analytics**

1. **Q:** How is Machine Learning applied to streaming data?  
   **A:** Models are trained offline, then deployed in streaming jobs for inference (e.g., fraud detection with pre-trained ML model).
2. **Q:** How do you do anomaly detection in real-time streams?  
   **A:** By applying statistical thresholds, clustering, or ML models to detect deviations in data streams.
3. **Q:** How does Spark handle real-time aggregations with Kafka?  
   **A:** Using groupByKey, reduceByKey, or window functions on streaming DataFrames.
4. **Q:** What is a sliding window aggregation?  
   **A:** Continuous computation of aggregations over overlapping time windows (e.g., 10s window sliding every 5s).
5. **Q:** What’s the difference between tumbling and sliding windows?  
   **A:** Tumbling = fixed non-overlapping intervals; Sliding = overlapping intervals for finer granularity.

**6. Production Operations**

1. **Q:** What is Spark Streaming UI used for?  
   **A:** To monitor streaming queries: batch duration, processing latency, event rates, input/output statistics.
2. **Q:** How do you monitor Kafka-Spark pipelines in production?  
   **A:** With custom metrics (Dropwizard, Prometheus, Grafana), logs, Spark UI, and Kafka JMX metrics.
3. **Q:** How do you implement alerting systems in streaming jobs?  
   **A:** By integrating Spark/Kafka with monitoring tools (Prometheus alerts, ELK stack, PagerDuty) to send alerts on threshold breaches.
4. **Q:** How do you analyze logs from streaming systems?  
   **A:** Centralize logs in ELK/EFK stack, apply parsing + alerting rules, and visualize in Kibana/Grafana.
5. **Q:** How do you troubleshoot lag in Kafka consumers?  
   **A:** Use Kafka Consumer Lag monitoring (kafka-consumer-groups.sh), increase partitions, tune consumer throughput, or scale consumer groups.

**7. Scenario-Based & Tricky Questions**

1. **Q:** You see duplicate events in Spark Streaming from Kafka. How do you fix it?  
   **A:** Enable **idempotent producer + transactions**, use checkpointing in Spark, and commit offsets only after successful processing.
2. **Q:** A Kafka consumer is always behind. What could be the reason?  
   **A:** Small batch size, insufficient partitions, slow processing, GC overhead, or network bottleneck.
3. **Q:** How do you guarantee ordering in Kafka consumption?  
   **A:** Kafka guarantees ordering within a partition. To preserve ordering, use only one consumer per partition.
4. **Q:** How do you handle late-arriving events in Spark Structured Streaming?  
   **A:** Use **watermarking** with event-time windows and allow late data up to a threshold.
5. **Q:** Your streaming job keeps failing. How do you make it fault-tolerant?  
   **A:** Enable checkpointing, WAL (write-ahead logs), idempotent sinks, and automatic retries.

**8. Cloud/Deployment-Oriented**

1. **Q:** How would you deploy Spark + Kafka on Kubernetes?  
   **A:** Use Spark operator on K8s, Kafka Helm charts, persistent volumes, and configure service discovery.
2. **Q:** How do you scale Kafka consumers in Spark Streaming?  
   **A:** Increase partitions in Kafka topic, increase Spark executors, and add more tasks for parallelism.
3. **Q:** What are Dead Letter Queues (DLQs) in Kafka?  
   **A:** Special topics for storing unprocessable/invalid messages for later inspection.
4. **Q:** What are transactional writes in Kafka?  
   **A:** Producers can atomically write to multiple partitions + commit offsets together, ensuring EOS.
5. **Q:** How do you checkpoint Kafka offsets in Spark Structured Streaming?  
   **A:** Spark maintains offset metadata in checkpoint directories (HDFS, S3, etc.) for recovery.

**9. Performance & Optimization**

1. **Q:** How do you optimize Spark Structured Streaming performance?  
   **A:** Use proper partitioning, caching, avoid shuffles, optimize batch size, enable backpressure.
2. **Q:** How to tune Kafka producer performance?  
   **A:** Increase batch.size, linger.ms, compression.type (snappy/lz4), and acks=1/0.
3. **Q:** How do you tune Kafka consumer performance?  
   **A:** Increase fetch.min.bytes, fetch.max.wait.ms, parallelism, and tune max.poll.records.
4. **Q:** How do you reduce checkpoint overhead in Spark Streaming?  
   **A:** Use RocksDB state store, reduce checkpoint interval, and compact state.
5. **Q:** What’s the impact of increasing Kafka partitions?  
   **A:** Increases parallelism & throughput, but adds overhead for metadata and rebalancing.

**10. Security & Reliability**

1. **Q:** How do you secure Kafka messages in production?  
   **A:** Use SSL/TLS for encryption, SASL for authentication, and ACLs for authorization.
2. **Q:** How do you implement data masking in Kafka-Spark pipelines?  
   **A:** Apply transformations in Spark to anonymize/mask sensitive fields before downstream usage.
3. **Q:** How do you ensure message durability in Kafka?  
   **A:** Set acks=all, enable replication factor ≥ 3, use min.insync.replicas=2.
4. **Q:** What is Kafka ISR (In-Sync Replica)?  
   **A:** Set of replicas that are fully caught up with the leader. Kafka only commits messages when written to ISR.
5. **Q:** How do you ensure disaster recovery in Kafka clusters?  
   **A:** Multi-DC replication (MirrorMaker 2.0), cross-cluster replication, and snapshots.