Apache Flink — Interview Q&A; Guide (Simple → Complex)

A Light■Theme Illustrated PDF covering core to advanced Apache Flink questions with code examples, including DataStream, SQL, windows, and stateful operations.

A. Core & Basics

Q: What is Apache Flink?

A: A distributed engine for real-time, stateful stream and batch processing with exactly-once guarantees.

Q: Flink vs Spark Streaming?

A: Flink is true streaming (record-at-a-time) with event-time semantics; Spark Structured Streaming uses micro-batches.

Q: Event time vs Processing time?

A: Event time is embedded in records; processing time is wall-clock on the machine.

Q: What is a watermark?

A: A marker that indicates event-time progress and helps close event-time windows.

Q: What is a checkpoint?

A: A distributed snapshot of state for recovery after failure.

Q: What is parallelism?

A: Defines how many operator subtasks run in parallel.

```
env.fromElements("a","b","a")
   .map(x -> Tuple2.of(x,1))
   .keyBy(t -> t.f0)
   .sum(1);
```

B. Windows and Time Semantics

Windows group events by time intervals for aggregation: tumbling, sliding, session, and global windows.

```
stream.assignTimestampsAndWatermarks(wmStrategy)
      .keyBy(e -> e.user)
      .window(TumblingEventTimeWindows.of(Time.seconds(5)))
      .reduce((a,b) -> a.add(b));
```

Watermark Strategy:

```
WatermarkStrategy<Event> wmStrategy =
  WatermarkStrategy.<Event>forBoundedOutOfOrderness(Duration.ofSeconds(5))
    .withTimestampAssigner((e, ts) -> e.eventTimeMillis);
```

C. State and Fault Tolerance

State is the backbone of Flink for exactly-once semantics and recoverability. Supports keyed and operator state.

```
StateTtlConfig ttl = StateTtlConfig
  .newBuilder(Time.hours(1))
  .setUpdateType(StateTtlConfig.UpdateType.OnCreateAndWrite)
  .cleanupFullSnapshot()
  .build();
ValueStateDescriptor<Integer> desc = new ValueStateDescriptor<>("cnt", Integer.class);
desc.enableTimeToLive(ttl);

public class CountFn extends KeyedProcessFunction<String, Event, Tuple2<String,Integer>> {
  private transient ValueState<Integer> count;
  public void open(Configuration c){ count = getRuntimeContext().getState(new ValueStateDe
  public void processElement(Event e, Context ctx, Collector<Tuple2<String,Integer>> out)
    int c = Optional.ofNullable(count.value()).orElse(0)+1;
    count.update(c);
    if (c % 1000 == 0) out.collect(Tuple2.of(ctx.getCurrentKey(), c));
  }
}
```

D. Sources, Sinks, and Connectors

Flink integrates with Kafka, JDBC, Filesystem, and Debezium CDC. Supports transactional (two-phase commit) sinks for exactly-once delivery.

```
KafkaSource<Event> source = KafkaSource.<Event>builder()
  .setBootstrapServers("localhost:9092")
  .setTopics("events")
  .setGroupId("g1")
  .setValueOnlyDeserializer(new EventDeser())
  .build();

env.fromSource(source, wmStrategy, "Kafka Source");
```

E. Table & SQL API

Flink’s Table API and SQL unify batch and streaming with time-based windows, temporal joins, and aggregation functions.

```
SELECT user_id, COUNT(*) AS cnt
FROM TABLE(
  TUMBLE(TABLE clicks, DESCRIPTOR(event_time), INTERVAL '1' MINUTE)
)
GROUP BY user_id, window_start, window_end;
```

Temporal Table Join:

```
SELECT o.order_id, d.price
FROM Orders AS o
JOIN Prices FOR SYSTEM_TIME AS OF o.rowtime AS d
ON o.sku = d.sku;
```

F. Advanced Topics & Best Practices

Covers CEP, backpressure, tuning, and production recommendations.

```
Pattern<Event, ?> pattern = Pattern.<Event>begin("start")
  .where(e -> e.type.equals("login"))
  .next("fail").times(3).within(Time.minutes(5));
```

Best Practices:

• Define event-time and watermarks for consistent processing.

• Use RocksDB for large state; enable incremental checkpoints.

• Use savepoints for safe upgrades.

• Monitor checkpoint latency and backpressure in Web UI.

• Configure GRACE period for late data and two-phase commit sinks.