CHECKLIST

Apache Spark Streaming Production Readiness Checklist

A practical checklist for evaluating Apache Spark Streaming workloads, from fit and latency to checkpointing, sink design, and recovery behavior before production rollout.

Apache Spark Streaming Production Readiness Checklist

Use this checklist to decide whether Apache Spark Streaming is a good fit and to validate the operational details before production use.

---

1) Workload fit

- [ ] The use case needs continuous processing, not delayed batch windows.
- [ ] A latency of a few seconds is acceptable for the business need.
- [ ] The workload benefits from reusing Spark for both batch and streaming.
- [ ] The pipeline must handle distributed inputs from multiple producers.
- [ ] The team needs a unified execution model for historical and incoming data.
- [ ] The workload does not require true sub-second event handling.

2) Data sources and ingestion

- [ ] All source systems are identified and documented.
- [ ] Expected event rates are measured or estimated for each source.

- [ ] Peak input bursts have been tested or realistically modeled.
- [ ] Source delivery semantics are understood (at-most-once, at-least-once, or replayable).
- [ ] The ingestion layer can absorb short-term spikes without data loss.
- [ ] Source schemas, formats, and evolution patterns are documented.
- [ ] Missing, late, or duplicate events are accounted for in the design.

3) Processing design

- [ ] The job logic is split into clear ingestion, transformation, and output stages.
- [ ] Stateless transformations are separated from stateful ones where possible.
- [ ] Keying and partitioning strategy are defined before implementation.
- [ ] Shuffle-heavy operations have been reviewed for performance impact.
- [ ] State retention windows are intentionally chosen and justified.
- [ ] High-cardinality keys have been evaluated for memory and recovery cost.
- [ ] The pipeline can combine streaming data with historical reference data if needed.

4) Latency and throughput

- [ ] The micro-batch interval matches the required freshness target.
- [ ] End-to-end latency has been measured under expected load.
- [ ] Throughput remains stable during sustained traffic, not just short tests.
- [ ] Backpressure behavior has been tested under overload conditions.
- [ ] The cluster has enough executor and memory capacity for peak demand.
- [ ] Processing time stays below the batch interval with safety margin.
- [ ] Recovery time is acceptable after failures or restarts.

5) Checkpointing and recovery

- [ ] Checkpoint storage is durable and highly available.
- [ ] Checkpoints are written frequently enough to limit replay cost.
- [ ] Recovery behavior has been tested from a real failure scenario.
- [ ] The pipeline can resume without manual cleanup after restart.
- [ ] State restoration time has been measured and accepted.
- [ ] Checkpoint data retention matches operational and audit needs.
- [ ] The team understands what is replayed after failure.

6) Sink and delivery semantics

- [ ] Downstream sinks are identified for every output stream.
- [ ] Each sink can tolerate duplicate writes or supports idempotent updates.
- [ ] Output semantics are documented for alerting, storage, and analytics targets.
- [ ] Write failures are handled safely and consistently.
- [ ] Partial writes do not corrupt downstream datasets.
- [ ] The sink design supports replay without creating bad records.
- [ ] Exactly-once behavior is verified rather than assumed.

7) Data quality and correctness

- [ ] Validation rules exist for malformed or incomplete input events.
- [ ] Deduplication strategy is defined where duplicate events are possible.
- [ ] Event ordering assumptions are explicitly documented.
- [ ] Late-arriving data handling is defined.
- [ ] Aggregations and windowing logic have been tested with edge cases.
- [ ] Output records are checked for completeness and correctness.
- [ ] Reference data joins are validated against realistic data volumes.

8) Operational readiness

- [ ] Metrics are in place for input rate, processing time, backlog, and sink writes.
- [ ] Logs are sufficient to trace failures without exposing sensitive data.
- [ ] Alerts exist for checkpoint failures, lag growth, and repeated task retries.
- [ ] Runbooks cover restart, replay, and recovery procedures.
- [ ] Ownership for the pipeline is clearly assigned.
- [ ] On-call responders know how to inspect stream health quickly.
- [ ] Capacity scaling steps are documented and tested.

9) Security and governance

- [ ] Access to source and sink systems follows least privilege.
- [ ] Sensitive telemetry is protected in transit and at rest.
- [ ] Streaming outputs are limited to the minimum required fields.
- [ ] Audit or compliance requirements are reflected in retention and access controls.
- [ ] Data masking or filtering is applied where required.
- [ ] The pipeline does not expose confidential records in logs or debug output.
- [ ] Governance controls are consistent with related batch jobs.

10) Testing before production

- [ ] Load testing has been performed with realistic event volumes.
- [ ] Failure testing has covered executor loss, restart, and checkpoint recovery.
- [ ] Sink retries and duplicate delivery scenarios have been tested.
- [ ] Monitoring dashboards show the right operational signals.
- [ ] The pipeline has been validated end to end in a staging environment.

- [ ] The team has compared expected vs. actual latency and throughput.
- [ ] A rollback plan exists if the stream causes downstream issues.

Go-live decision check

Only move to production when all of the following are true:

- [ ] The workload is a good micro-batch fit.
- [ ] Latency targets are met in testing.
- [ ] Checkpointing and recovery are proven.
- [ ] Sinks can safely handle replay or duplicates.
- [ ] Monitoring and alerting are in place.
- [ ] Security and governance requirements are satisfied.
- [ ] The operational team is ready to support the pipeline.

Quick rule of thumb

Choose Apache Spark Streaming when you need distributed, replayable, operationally manageable processing with acceptable short latency. Avoid it when every event must be acted on immediately or when your sinks cannot safely handle replay behavior.