CHECKLIST

Big Data Production Readiness Checklist

A practical checklist for verifying a big data platform before production use, including architecture, security, data quality, scaling, monitoring, and rollback readiness.

Programming - July 04, 2026

Purpose

A big data platform can look healthy in development and still fail in production because of hidden issues in ingestion, data quality, permissions, cluster sizing, schema evolution, or recovery procedures. This checklist helps you verify whether a big data solution is actually ready to handle production workloads, protect data, and recover from failures without guesswork.

Use this checklist to decide whether the platform, pipelines, and operational controls are ready for live use. By the end, you should be able to confirm applicability, run a practical validation workflow, capture evidence, score readiness, and identify what must be fixed before production use.

How to use this checklist

Treat each phase as a review gate. For every check, collect evidence, assign an owner, and mark the item only when the acceptance criteria are met. If an item cannot be verified, assume it is not ready.

A useful pattern is to review the platform in order: architecture, data movement, security, reliability, performance, operations, and go-live controls. For related validation work on AI data and model pipelines, compare the readiness method with the Learning Implementation Roadmap Checklist and the Learning Checklist for AI and Machine Learning Projects when your big data estate feeds analytics or ML workloads.

1) Scope and platform assumptions

Purpose: Confirm what is being approved, what workloads it supports, and what constraints apply.

- ? Document the production use case, data domains, expected consumers, and latency requirements.
- ? Confirm the platform boundaries, including source systems, storage layers, processing engines, and serving outputs.
- ? Validate that the target data volumes, event rates, and peak concurrency are explicitly stated.
- ? Review which environments, regions, and tenants are in scope and which are excluded.
- ? Assign a named owner for each major component: ingestion, storage, processing, orchestration, security, and observability.

Evidence to capture: architecture diagram, workload profile, RACI or ownership matrix, environment inventory.

Acceptance criteria: the team can state exactly what is being launched, who owns it, and what operational limits it must meet.

Owner: platform architect or data engineering lead.

Review cadence: once before go-live and again after any major scope change.

2) Data sources and ingestion controls

Purpose: Verify that upstream sources can deliver data reliably, completely, and with traceable lineage.

- ? Confirm each source system, delivery method, schedule, and retry behavior.
- ? Validate authentication, network reachability, and secret handling for each ingestion path.
- ? Review whether batch, streaming, CDC, or file-based transfer is used and whether each mode has a defined failure behavior.
- ? Test source-to-landing delivery with representative payloads, including malformed and late-arriving records.
- ? Document how duplicates, partial files, offsets, and replay windows are handled.
- ? Confirm that ingestion lag thresholds and alert conditions are defined for each critical source.

Evidence to capture: ingestion run logs, connection test results, sample payloads, offset or watermark records, failure and retry logs.

Acceptance criteria: every required source can deliver data within the approved window and failure handling is explicit.

Owner: ingestion engineer or data pipeline owner.

Review cadence: before go-live and after any source schema or schedule change.

3) Schema, data quality, and contract validation

Purpose: Make sure upstream changes do not break downstream jobs or silently corrupt datasets.

- ? Validate schema compatibility rules for additions, removals, renames, and type changes.
- ? Review whether schema registry, catalog, or contract enforcement is enabled where required.
- ? Test representative records against parsing, transformation, and validation rules.
- ? Confirm that null handling, default values, and invalid record routing are defined.
- ? Document data quality checks for completeness, uniqueness, freshness, referential integrity, and value ranges.
- ? Assign a quarantine or dead-letter path for rejected records and confirm retention settings.

Evidence to capture: schema comparison output, validation test results, data quality rule set, rejected record samples, lineage records.

Acceptance criteria: breaking changes are detected before production impact, and bad records are isolated rather than silently accepted.

Owner: data quality lead or analytics engineer.

Review cadence: every release and whenever a source schema changes.

4) Storage, partitioning, and lifecycle controls

Purpose: Verify that data can be stored efficiently, queried predictably, and retained safely.

- ? Confirm storage classes, encryption settings, and replication behavior for raw, curated, and serving layers.
- ? Review partitioning, clustering, indexing, or file sizing decisions for the expected query patterns.
- ? Validate that small-file risk, compaction strategy, and file format choices are addressed.
- ? Document retention, archival, legal hold, and deletion rules for each dataset class.
- ? Test whether restore, reprocess, or replay is possible from retained raw data.
- ? Confirm storage capacity alarms and growth forecasts are in place for the expected ingest rate.

Evidence to capture: storage policy configuration, partition design, lifecycle policy, capacity trend report, restore test results.

Acceptance criteria: data is stored in a way that supports expected queries, retention is enforceable, and recovery from retained sources is feasible.

Owner: storage or platform operations lead.

Review cadence: monthly and before large volume growth or retention changes.

5) Processing, orchestration, and dependency handling

Purpose: Ensure pipelines run deterministically and can recover from dependency failures.

- ? Confirm each pipeline has a defined trigger, schedule, and dependency graph.
- ? Validate job idempotency or replay safety for reruns and partial failures.
- ? Review timeout, retry, backoff, and failure escalation settings for critical jobs.
- ? Test rerun behavior after a controlled failure in staging or a preproduction environment.
- ? Document how upstream delays, missing partitions, and late-arriving data affect downstream jobs.
- ? Assign a manual override procedure for urgent recovery when automation fails.

Evidence to capture: workflow definitions, scheduler logs, rerun test output, dependency map, escalation runbook.

Acceptance criteria: the pipeline can be rerun safely, dependencies are explicit, and manual recovery is documented.

Owner: pipeline owner or orchestration engineer.

Review cadence: every release and after scheduler or dependency changes.

6) Security, access, and secrets

Purpose: Confirm that sensitive data and administrative access are controlled at production grade.

- ? Validate authentication methods for operators, services, and integrations.
- ? Review role-based access control, least privilege, and separation of duties.
- ? Confirm encryption in transit and at rest for all sensitive data paths.
- ? Test secret rotation, expiration handling, and failure behavior after credential replacement.
- ? Document which datasets contain regulated, sensitive, or restricted data and how access is audited.
- ? Confirm that privileged actions are logged and reviewable.

Evidence to capture: access review report, encryption settings, secret inventory, audit log samples, privileged access logs.

Acceptance criteria: only approved identities can access production data and administrative operations are auditable.

Owner: security engineer or platform security lead.

Review cadence: before go-live and on a scheduled access review cycle.

7) Network, perimeter, and isolation controls

Purpose: Reduce exposure by verifying that only approved traffic can reach the platform.

- ? Confirm network segmentation between ingestion, processing, storage, and administrative paths.
- ? Review firewall, security group, route, or private endpoint rules for least exposure.
- ? Validate that public access is disabled where it is not required.
- ? Test connectivity from approved clients only and verify blocked paths remain blocked.
- ? Document cross-region, cross-account, or cross-tenant traffic flows and approval points.

Evidence to capture: network diagram, rule export, connection test results, blocked connection attempts, approval records.

Acceptance criteria: the platform is reachable only through approved paths and unwanted access attempts are denied.

Owner: network or cloud infrastructure engineer.

Review cadence: after network changes and during periodic control review.

8) Capacity, scaling, and performance validation

Purpose: Verify the system can absorb expected load without queue growth, failures, or unacceptable delay.

- ? Test representative peak load, not just average load, for ingestion and processing paths.
- ? Validate cluster autoscaling or manual scaling thresholds against measured demand.
- ? Review CPU, memory, disk, shuffle, queue depth, and network saturation under load.

- ? Confirm that query latency, job duration, and backpressure stay within agreed limits.
- ? Document bottlenecks, hot partitions, skew, and noisy-neighbor risks.
- ? Assign scaling triggers and emergency capacity procedures.

Evidence to capture: load test results, utilization graphs, latency distributions, saturation alerts, bottleneck analysis.

Acceptance criteria: the platform meets agreed service targets at expected peak load with a known scaling path.

Owner: performance engineer or platform operations lead.

Review cadence: before go-live and after any material workload increase.

9) Monitoring, logging, and alerting

Purpose: Make sure operators can detect failure early and diagnose it with enough context to act.

- ? Confirm that platform metrics, job metrics, and business data health metrics are collected.
- ? Validate alert thresholds for ingestion lag, failed jobs, missing data, storage growth, and permission errors.
- ? Review log retention, correlation IDs, timestamps, and structured logging consistency.
- ? Test at least one alert path end to end, including notification delivery and acknowledgement.
- ? Document dashboards for daily operations, incident response, and capacity review.

Evidence to capture: dashboard links or exports, alert test results, log samples, on-call routing configuration.

Acceptance criteria: operators can see the right signals quickly and alerts fire before user impact becomes severe.

Owner: observability or SRE lead.

Review cadence: weekly for high-change systems and monthly for stable systems.

10) Backup, restore, and disaster recovery

Purpose: Prove that the platform and critical datasets can be recovered within approved time limits.

- ? Validate backup scope for configuration, metadata, critical state, and required datasets.
- ? Test restore of at least one representative dataset and one operational component.
- ? Review recovery time objective and recovery point objective against business expectations.
- ? Document cross-zone or cross-region recovery behavior where required.
- ? Confirm replay, rehydration, or backfill procedures for data that cannot be restored directly.
- ? Assign a disaster recovery owner and a communication path for recovery events.

Evidence to capture: backup logs, restore test results, RTO/RPO declaration, DR runbook, failed-restore notes.

Acceptance criteria: a real restore has been demonstrated, and recovery objectives are realistic and documented.

Owner: infrastructure or disaster recovery lead.

Review cadence: quarterly and after major architecture changes.

11) Operational handoff and support readiness

Purpose: Ensure the operations team can run the platform without depending on the build team for every issue.

- ? Document runbooks for common incidents, routine maintenance, and escalation paths.
- ? Review support hours, on-call coverage, and handoff responsibilities.
- ? Validate that the team knows how to pause, resume, reprocess, and roll back safely.
- ? Confirm ticket severity definitions and response time expectations.

- ? Test a simulated incident review that uses logs, metrics, and runbook steps to resolve a known issue.
- ? Assign post-go-live ownership for backlog triage and recurring issue management.

Evidence to capture: runbooks, support matrix, incident simulation notes, escalation policy, ownership handoff record.

Acceptance criteria: a responder can operate and recover the system using the documented materials alone.

Owner: operations lead or service owner.

Review cadence: before handoff and after major operational changes.

Common mistakes to avoid

These failures repeatedly create avoidable production incidents:

- Treating successful development runs as proof of production readiness.
- Verifying only average load instead of peak load, backfill load, or replay load.
- Leaving schema evolution undocumented until a source change breaks downstream jobs.
- Ignoring small-file buildup, partition skew, or retention growth until query performance degrades.
- Skipping restore tests and assuming backups are usable.
- Using broad access roles for convenience and postponing least-privilege cleanup.
- Missing alert thresholds for data freshness, which is often the earliest sign of ingestion failure.
- Approving the platform without a clear owner for reruns, incident handling, and escalation.

If these issues are already present, do not compensate with more monitoring alone; fix the underlying control gap first.

Readiness scoring

Use a simple 0?2 score for each phase:

- 0 = not verified or failed
- 1 = partially verified, evidence incomplete, or manual workaround required
- 2 = verified with evidence and acceptable acceptance criteria

Scoring method

Add the scores from the 11 phases for a maximum of 22 points.

- 18?22: production ready with routine monitoring
- 13?17: conditionally ready; launch only after named gaps are remediated or risk-accepted
- 0?12: not ready for production use

Pass/fail decision rule

Use pass/fail gates in addition to the score:

- Fail immediately if restore cannot be demonstrated, critical access controls are missing, or ingestion failure would cause silent data loss.
- Fail immediately if peak-load performance has not been validated for the intended workload.
- Pass conditionally only when every failed item has a named owner, remediation date, and retest plan.

Review log template

Capture the result of each phase in a consistent format so the review can be audited later.

Final check before production use

Before approving the platform, confirm that every critical workflow has evidence, every exception has an owner, and every recovery path has been tested. A big data system is production ready only when it can ingest, process, secure, monitor, and recover data at the required scale without relying on assumptions.

Use this guidance together with JavaScript checklist and python checklist to connect the workflow with related operational context already available on the site.

Related guides in this cluster

- Big Data Security Checklist for Hadoop Cluster Hardening
- Troubleshoot Apache Spark Shuffle Failures in Big Data Jobs

Related guides in this cluster

- Optimizing Distributed SQL Queries for Large-Scale Data Processing

Related guides in this cluster

- Implementing Apache Kafka Exactly-Once Processing in Big Data Pipelines

Related guides in this cluster

- Optimizing Spark Job Performance with Query Tuning Techniques
- Optimizing Apache Spark Query Performance with Partition Pruning

Related guides in this cluster

- Detecting Anomalies in Big Data Pipelines with Apache Spark

Related guides in this cluster

- Optimizing Apache Spark Jobs for Large-Scale Data Processing

Related guides in this cluster

- Distributed Big Data Processing with Apache Spark Streaming

Related guides in this cluster

- Optimizing Apache Spark Performance for Large-Scale ETL Pipelines

Related guides in this cluster

- Implementing Secure Data Pipelines in Big Data Systems
- How to Optimize Spark Streaming Performance for Large Data Pipelines

Related guides in this cluster

- How to Secure Big Data Pipelines with Apache Kafka Streams

Related guides in this cluster

- Optimizing Apache Spark Performance with Memory Tuning Tips

Related guides in this cluster

- How to Build Scalable Big Data Pipelines with Apache Spark