

Apache Spark smoke tests on kubernetes

Julien Laurenceau | Published 2026-01-12 | Updated 2026-09-30

<https://pepitedata.com/benchmark-spark-data-access-tpcx-hs-kubernetes/>

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



Introduction

Benchmarking remains a critical (and often underestimated) tool when designing or validating large-scale data platforms. While many teams rely on synthetic workloads or production replays, standardized benchmarks still play a key role when comparing architectures, tuning clusters, or validating infrastructure choices.

TPCx-HS is one of those benchmarks: designed to stress **big data systems at scale**, with a focus on throughput, storage, and distributed processing.

However, the original reference implementations were designed for **older Hadoop-centric stacks**, making them increasingly difficult to run in modern environments built around **Apache Spark 2.x / 3.x** and **Kubernetes**.

This is why I decided to **modernize TPCx-HS**:

- Make it runnable on **current Spark versions**
- Make it **Kubernetes-native**
- Keep it **close to the spirit of the original benchmark**, without over-engineering

The result is an open-source project you can find here:

-> <https://github.com/julienlau/tpcx-hs>

Why TPCx-HS Still Matters

TPCx-HS still matters because it stresses the three things that break first on Kubernetes with object storage: shuffle performance, executor sizing and storage throughput under sustained load. It is a large-scale data generation, distributed sort and end-to-end throughput benchmark, formerly TeraSort-based, and unlike most suites it exercises the data path rather than the query layer.

TPCx-HS (formerly TeraSort-based benchmarks) is designed to evaluate:

- Large-scale data generation
- Distributed sort performance
- End-to-end system throughput (CPU, network, storage)

In practice, it's a **great stress test** for:

- Object storage vs HDFS
- Shuffle performance
- Executor sizing and memory pressure
- Scheduler behavior under load

Yet, many teams avoid it today because:

- Legacy scripts assume **YARN**
- Spark versions are outdated
- Kubernetes is not supported out of the box

That gap is exactly what this project addresses.

What Was Changed

The modernization touches four things: the Spark version range it runs on, the resource manager, where data lives, and what the harness refuses to do. Each change removes a YARN-era assumption or a shared-filesystem dependency so the benchmark runs against today's stack.

1. Spark 2.x and 3.x Compatibility

The codebase was refactored to:

- Run on **Spark 2.x and Spark 3.x**
- Avoid deprecated APIs
- Use configuration patterns that work across versions

This makes it usable on:

- On-prem clusters
- Cloud managed Spark
- Spark-on-Kubernetes distributions

2. Kubernetes-First Design

Running Spark on Kubernetes introduces very different constraints compared to YARN:

- No shared filesystem by default
- Ephemeral executors
- Externalized storage (S3, MinIO, etc.)
- Explicit resource isolation

The project was adapted to:

- Work cleanly with **Spark on Kubernetes**
- Be compatible with **object storage backends**
- Avoid assumptions tied to HDFS or static nodes

This makes it suitable for **modern cloud-native data platforms**.

3. Minimalism Over Reinvention

This is **not** a full benchmarking framework.

Deliberate choices:

- No custom orchestration layer
- No vendor-specific tuning
- No attempt to “optimize the benchmark itself”

The goal is simple:

Provide a **clean, runnable, understandable TPCx-HS implementation** for modern Spark platforms.

You are expected to tune **your cluster**, not the benchmark.

Typical Use Cases

Typical uses cluster around comparison and capacity: comparing Spark on Kubernetes against a legacy YARN estate, stressing object storage throughput, validating executor sizing and shuffle behavior, reproducing a performance issue under controlled load, and holding a known baseline before a production rollout.

This implementation is useful if you want to:

- Compare **Spark-on-K8s vs legacy YARN**
- Stress test **object storage throughput**
- Validate executor sizing and shuffle behavior
- Reproduce performance issues under controlled load
- Have a **known baseline** before production rollout

It is intentionally infrastructure-agnostic.

What This Is Not

To set expectations clearly, three exclusions apply. It is not a certified TPC submission, not a vendor benchmark and not an auto-tuning tool. The harness stays deliberately minimal so the numbers describe your cluster rather than the benchmark's own engineering. Tune your cluster, not the benchmark.

- [x] Not a certified TPC submission
- [x] Not a vendor benchmark
- [x] Not an auto-tuning solution

It is a **practical engineering tool**, not a marketing benchmark.

Who This Is For

The project targets data platform teams who must defend a Spark-on-Kubernetes decision with measurements. Platform engineers sizing a migration, SREs reproducing a performance regression, and architects who need one reproducible baseline before production all get a runnable harness from it.

This project will be most useful to:

- Data platform engineers
- SREs working on Spark infrastructure
- Architects migrating from Hadoop/YARN to Kubernetes
- Anyone who needs **realistic, heavy Spark workloads** without production data

If you've ever asked:

"Can my Spark platform really handle this scale?"

This benchmark helps you answer it.

Related: once you have measured how much CPU your Spark executors actually use, the next problem is that Kubernetes reserves far more than that. I cover the fix in [CPU overcommit for Spark on Kubernetes](#).

Repository

[package] Source code and documentation:

-> <https://github.com/julienlau/tpcx-hs>

Contributions, feedback, and performance reports are welcome.

[Linkedin post](#)

<https://pepitedata.com/expert-call/>

Update 2026-09-29. This post is now part of [Performance and Cost Engineering for Data Platforms](#), the field guide to performance and cost engineering.

Related posts

- [Performance and Cost Engineering for Data Platforms](#)
- [Spark on Kubernetes reserves CPU it never uses](#)
- [Benchmark Best Practices, Part 1: Amdahl's law caps your scaling claim](#)
- [Sail: when Apache Spark meets Rust](#)

PDF snapshot of the public article. The online version is authoritative; links remain clickable.