# Apache Spark on Kubernetes Workshop

## Introduction

Apache Spark is a popular distributed data processing engine, and **Kubernetes** is a powerful container orchestration platform. In this workshop, we’ll explore **why containerization and Kubernetes are important** for running Spark applications, and how to deploy Spark on Kubernetes. The goal is to help data engineers (who may not be DevOps experts) understand the benefits of containerizing Spark jobs, using Kubernetes for orchestration, and the **pros and cons** of Spark on Kubernetes. We’ll also walk through a **hands-on demo** of running a Spark job on a local Kubernetes cluster, so you can see it in action.

## Why Containerize Spark Applications?

Containerization means packaging an application with its dependencies into a lightweight, portable container image. This approach offers several benefits for Spark applications:

- **Environment Consistency:** A container image encapsulates all dependencies (libraries, Spark version, etc.), ensuring the application runs the same everywhere. This avoids the “it works on my machine” problem and guarantees consistency across dev, test, and prod[[1]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Here%20are%20some%20few%20key,benefits).

- **Isolation:** Each Spark job can run in its own container, isolated from other jobs and the host system. This prevents dependency conflicts and allows different jobs to have different Spark or Python versions without interference[[2]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,a%20Spark%20application%20if%20needed).

- **Reproducibility & Agility:** Containers enable repeatable and reliable build workflows. You build the image once and run it anywhere (locally or at scale)[[3]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=This%20is%20the%20main%20motivation,iterate%20on%20your%20code%20faster). This speeds up iteration: developers have reported that using Docker with Spark cut their development cycle from minutes to seconds[[4]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=1,to%20less%20than%2030%20seconds).

- **Portability:** A Spark job packaged in a Docker container can be deployed on any environment that supports containers (on-premises or cloud). This makes it easier to move Spark workloads between environments or cloud providers without compatibility issues[[5]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=4).

- **Dependency Management:** Managing PySpark dependencies can be notoriously painful. With containers, you can bake necessary libraries into the image, avoiding flaky init scripts and ensuring all executors have the needed packages[[6]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Our%20favorite%20benefit%20is%20definitely,libraries%20on%20each%20application%20launch).

In short, containerizing Spark jobs makes them **more portable, consistent, and easier to deploy**, while reducing the manual work needed to configure environments[[3]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=This%20is%20the%20main%20motivation,iterate%20on%20your%20code%20faster).

## Why Use Kubernetes for Spark?

If containers package our Spark applications, **Kubernetes (K8s)** is the platform that runs and manages those containers across a cluster of machines. Kubernetes provides an “operating system” for clusters, automating deployment, scheduling, scaling, and recovery of containerized applications[[7]](https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b?gi=0bec52493821" \l ":~:text=Kubernetes%20in%20a%20nutshell%20is,containerized%20applications%20of%20varying%20sizes). Here’s why Kubernetes is a great fit for Spark:

* **Orchestration & Scheduling:** Kubernetes will automatically schedule Spark driver and executor containers onto cluster nodes, handle restarts on failure, and even **auto-scale** if needed. This takes away a lot of manual ops work. Essentially, Kubernetes acts as Spark’s cluster manager (replacing YARN or standalone mode) and efficiently **deploys, auto-scales, and heals** the Spark pods in the cluster[[8]](https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b?gi=0bec52493821" \l ":~:text=Figure%201%E2%80%931%3A%20Deploy%2C%20auto,Kubernetes%20from%20What%20is%20Kubernetes).
* **Resource Efficiency:** Running Spark on Kubernetes can improve cluster utilization. Traditionally with Hadoop YARN, many teams ran a separate cluster per job or would spin up a transient cluster for each Spark application, incurring start-up overhead and under-utilizing resources. Kubernetes encourages a **shared cluster** where multiple Spark jobs run side by side in namespaces, with Kubernetes dynamically allocating and reclaiming resources per job[[9]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=2,big%20cost%20savings)[[10]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=As%20a%20consequence%2C%20many%20platforms,compute%20resources%20with%20this%20approach). This means no more 10-minute YARN startup delays for each job and better bin-packing of workloads on the cluster nodes[[11]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=running%20transient%20clusters%20for%20production,compute%20resources%20with%20this%20approach). In fact, companies have found that using one Kubernetes cluster for many Spark jobs yields big cost savings, thanks to *full isolation* of each job plus shared resource pools[[12]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=On%20other%20cluster,have%20to%20compromise%20on%20isolation)[[13]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Image%20%20100Spark,managing%20clusters%2C%20just%20run%20containers).
* **Replacing YARN’s Limitations:** Hadoop YARN (a common Spark scheduler) can be complex and often requires static partitioning of resources. It also forces cluster teardown for each job in many cases, wasting time and compute[[14]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=There%E2%80%99s%20no%20Spark%20vs%20Kubernetes,results%20in%20inefficient%20task%20management). Kubernetes, on the other hand, has become ubiquitous in modern infrastructure and doesn’t have those limitations. Recent Spark versions have closed the performance gap with YARN, and Kubernetes is poised to **replace YARN as the main big-data cluster manager** in many organizations[[15]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=On%20the%20other%20hand%2C%20Kubernetes,main%20big%20data%20processing%20engine)[[16]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Second%2C%20recent%20Spark%20versions%20,performance%20improvement%20for%20Spark%20workloads). For example, Amazon’s tests showed Spark on Kubernetes was ~5% faster than YARN, and Amazon EKS (Kubernetes) could reduce Spark costs by up to 61% while improving performance[[17]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Second%2C%20recent%20Spark%20versions%20,performance%20improvement%20for%20Spark%20workloads).
* **Unified Infrastructure & Ecosystem:** If your company already uses Kubernetes for microservices, deploying Spark on the same platform unifies your stack. You can leverage Kubernetes features like namespaces (for multi-tenancy and quotas), role-based access control (for security), and reuse common tools for logging and monitoring across all applications[[18]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Deploying%C2%A0Spark%20on%20Kubernetes%C2%A0gives%20you%20powerful,grained%20security%20and%20data%20access). Kubernetes’ rich ecosystem (Prometheus/Grafana monitoring, Fluentd/ELK logging, etc.) can be directly applied to Spark workloads[[19]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=6,Ecosystem)[[20]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,Jenkins%20X%20or%20GitLab%20CI). This **cloud-native integration** means less vendor lock-in (Spark becomes cloud-agnostic, not tied to Hadoop vendor distributions)[[21]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=If%20you%20have%20a%20need,in).
* **Scalability and Flexibility:** Kubernetes makes it easier to scale Spark applications horizontally. It also allows running Spark alongside other workloads (like web services or machine learning models) on the same cluster, orchestrated by one system. This simplifies resource sharing between batch analytics and other services, maximizing utilization of every node[[22]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=5,Other%20Applications).

In summary, we **need Kubernetes** because once Spark jobs are containerized, we require a robust way to run many containers across many machines. Kubernetes provides the automation and infrastructure-as-code needed to deploy Spark reliably at scale, with the added benefit of integrating Spark into the broader IT ecosystem.

## How Spark Runs on Kubernetes (Architecture)

*Spark on Kubernetes Architecture:* In Spark’s Kubernetes mode, the Kubernetes cluster acts as the **cluster manager** for Spark, taking on the role that YARN or Spark’s standalone master would play. Here’s a high-level look at what happens when you run a Spark job on K8s:

* **Spark Driver Pod:** When you submit a Spark application with Kubernetes as the master, Kubernetes will launch the Spark *driver* inside a Kubernetes Pod (a pod is the basic deployable unit in K8s)[[23]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=You%20can%20submit%20Spark%20apps,and%20the%20Spark%20configuration). The driver pod contains the Spark driver process, which is the main controller of the Spark application.
* **Executor Pods:** The Spark driver then requests executor pods from Kubernetes for distributed computation[[24]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Kubernetes%20takes%20this%20request%20and,pools%20to%20meet%20the%20requirements). Each Spark executor will run in its own pod. The driver uses the Kubernetes API to create these pods dynamically, and Kubernetes schedules them onto nodes. Executors connect back to the driver and start processing tasks.
* **Dynamic Lifecycle:** Executors can be scaled up and down during runtime if Spark’s dynamic allocation is enabled (requiring Spark 3.x)[[25]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=,anticipate%20spot%20kills%20and%20gracefully). When the Spark job finishes, the executor pods terminate and free their resources[[26]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,collected%20or%20manually%20cleaned%20up). The driver pod remains in a “completed” state for a while (so you can fetch logs or Spark UI data) until Kubernetes garbage-collects it[[27]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=When%20you%20run%20Spark%20on,and%20implement%20the%20application%20code). Importantly, in the completed state the driver pod isn’t using CPU/Memory, so finished apps don’t hold resources unnecessarily[[28]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,collected%20or%20manually%20cleaned%20up).
* **Kubernetes as Scheduler:** Kubernetes handles placing pods on available nodes, accounting for resource requirements. It ensures each Spark pod gets the requested CPU/memory and can recover pods on failures. Spark’s built-in Kubernetes scheduler backend works on a **desired state principle**, where the driver asks for a certain number of executors and Kubernetes continually works to maintain that state (adding or removing executors)[[29]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=dependencies%20to%20run%20,and%20the%20Spark%20configuration)[[30]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=To%20go%20a%20little%20deeper%2C,lives%20in%20the%20Spark%20driver).
* **Networking and Storage:** The driver and executors communicate over the Kubernetes network (they’re all in the cluster network). Service discovery is handled natively by K8s. For storage, Spark can use hostPath volumes or PVs if it needs to mount external storage, but usually data is read from external systems (HDFS, cloud storage, etc.) over the network. Kubernetes will also manage other aspects like config files or secrets (e.g., passing credentials) via its primitives.

In essence, Spark-on-Kubernetes replaces the need to manually manage Spark clusters. **You don’t “launch a Spark cluster” ahead of time** – you simply submit a job to K8s, and it takes care of launching the needed Spark driver/executors as pods on the fly, then cleans them up when done[[27]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=When%20you%20run%20Spark%20on,and%20implement%20the%20application%20code)[[31]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Businesses%20are%20embracing%20a%20Spark,processes%20and%20shorten%20deployment%20time). This yields a more elastic and efficient usage of resources compared to statically allocated clusters.

## Pros of Running Spark on Kubernetes

Running Spark on Kubernetes can offer many advantages compared to traditional deployments. Here are some of the key **pros**:

* **Portable, Consistent Environments:** As discussed, Kubernetes requires containerizing Spark apps, which yields portable and reproducible environments. You build a Spark Docker image with all dependencies and can run it anywhere – local, on-prem, or any cloud – with consistent behavior[[3]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=This%20is%20the%20main%20motivation,iterate%20on%20your%20code%20faster)[[32]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=4). This eliminates the configuration drift issues common in bare-metal or VM deployments of Spark.
* **Dependency Isolation:** Each Spark application runs fully isolated in its own set of containers. There’s no need to have a global Spark or Python version on the cluster – different jobs can use different versions or dependency sets without conflict[[33]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=On%20other%20cluster,have%20to%20compromise%20on%20isolation)[[2]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,a%20Spark%20application%20if%20needed). This isolation helps in multi-tenant environments where multiple teams run Spark jobs on the same cluster.
* **Dynamic Resource Sharing = Efficiency & Cost Savings:** Kubernetes allows a **shared pool of resources** for all Spark jobs, which drives up utilization. Spark on K8s supports fine-grained dynamic allocation of executors, so resources are used only when needed and returned to the pool afterward[[34]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Kubernetes%20excels%20at%20managing%20cluster,When%20running%20Spark%20on%20Kubernetes)[[35]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,efficiently%2C%20improving%20overall%20cluster%20utilization). This contrasts with having dedicated Spark clusters per team or per job that sit idle. The result is significant cost efficiency. For example, one case study showed **65% cost savings** by migrating from EMR (Spark on YARN) to Spark on Kubernetes due to better sharing and using spot instances[[36]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Despite%20the%20raw%20performance%20of,Kubernetes%20%28Data%20Mechanics). Kubernetes can bin-pack executors tightly and quickly recycle resources for the next job, avoiding the waste of long cluster spin-up times[[11]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=running%20transient%20clusters%20for%20production,compute%20resources%20with%20this%20approach)[[37]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=How%20does%20this%20cost%20reduction,resources%20to%20be%20moved%20elsewhere).
* **Scalability and Auto-Scaling:** Kubernetes is built to scale workloads horizontally. Spark can leverage this by launching many executors in parallel across nodes, and you can also integrate with the Kubernetes cluster autoscaler. For instance, if a big Spark job needs more nodes, Kubernetes can auto-provision new VMs (in cloud environments) to meet the demand and then scale them down later. This on-demand elasticity is harder to achieve with static Hadoop clusters.
* **Unified Platform & Ecosystem:** Deploying Spark on Kubernetes means you can use the same infrastructure and tools for Spark as for other applications. You can run Spark alongside streaming services, web apps, etc., on one Kubernetes cluster – all managed through a common interface. Monitoring and logging can be unified: e.g., use **Prometheus/Grafana** to monitor Spark metrics as well as cluster health, and use a centralized logging stack for all pods[[19]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=6,Ecosystem)[[20]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,Jenkins%20X%20or%20GitLab%20CI). You also benefit from Kubernetes features like namespaces for multi-tenancy and **RBAC for access control**, which can restrict who can submit Spark jobs or access certain data[[18]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Deploying%C2%A0Spark%20on%20Kubernetes%C2%A0gives%20you%20powerful,grained%20security%20and%20data%20access)[[38]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=7). Overall, Spark becomes a first-class citizen in the cloud-native ecosystem, enabling things like GitOps CI/CD for Spark jobs, service meshes for networking, and so on[[19]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=6,Ecosystem)[[39]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=or%20the%20ELK%20,Jenkins%20X%20or%20GitLab%20CI).
* **Cloud-Agnostic and No Vendor Lock-In:** Because Kubernetes is supported across all major clouds and on-prem, running Spark on Kubernetes makes your data pipeline more cloud-agnostic. You’re not tied to a specific Hadoop/YARN distro or cloud-specific Spark service. This flexibility can prevent lock-in and allow easier migration or hybrid-cloud setups[[21]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=If%20you%20have%20a%20need,in).
* **Performance Parity (with Newer Spark):** As of Spark 3.x, Spark on Kubernetes offers performance on par with traditional Spark on YARN[[40]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=The%20Neutral%20%E2%80%93%20Spark%20performance,is%20the%20same). Early on, Kubernetes support was experimental, but now it’s **production-ready** (Spark 3.1+ officially GA for Kubernetes) and has resolved prior issues[[17]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Second%2C%20recent%20Spark%20versions%20,performance%20improvement%20for%20Spark%20workloads)[[25]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=,anticipate%20spot%20kills%20and%20gracefully). So you generally don’t sacrifice performance; in fact, with better resource management, you might gain throughput and certainly gain in throughput per dollar spent.

## Cons and Challenges of Spark on Kubernetes

Despite the benefits, there are some **drawbacks and challenges** to be aware of when deploying Spark on Kubernetes:

* **Steep Learning Curve:** Kubernetes is a complex system with its own concepts (pods, services, yaml configs, etc.). Data engineers not already familiar with K8s may face a learning curve. The new abstractions and tools can be “frightening” at first and take you away from core data engineering work[[41]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=1.%C2%A0Making%20Spark,requires%20build%20time%20and%20expertise). In industry surveys, over half of professionals noted the upskilling challenge (DevOps, operations, security knowledge) as a barrier to K8s adoption[[42]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Initial%20deployment%20can%20be%20among,including%20Spark%20on%20K8s%20adoption)[[43]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Pepperdata%E2%80%99s%20State%20of%20Kubernetes%202023,including%20Spark%20on%20K8s%20adoption). So teams might need to invest in training or new expertise.
* **Cluster Setup and Maintenance Overhead:** Unlike using a managed Spark service, running Spark on Kubernetes means **managing a lot of infrastructure** yourself (unless you use a managed K8s service, which handles the control plane but you still manage configurations). You have to create and configure the K8s cluster (and possibly manage many nodes/VMs), handle networking and storage integration, and possibly set up Kubernetes add-ons like the cluster autoscaler. Additionally, to get the most out of Spark on K8s, you often need to set up things like the Spark Kubernetes Operator (to easily manage Spark applications), a Docker registry for your Spark images, a Spark History Server to view past job UIs, and monitoring/logging solutions[[44]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=introduces%20can%20be%20frightening%20and,away%20from%20your%20core%20mission)[[45]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=,your%20notebooks%20and%2For%20your%20scheduler). All this requires DevOps effort and time to build a robust production-ready environment.
* **Migrating Existing Workloads:** If you have a lot of Spark jobs running on YARN or other platforms, migrating them to Kubernetes isn’t plug-and-play. You need a solid migration strategy, including adapting jobs to use container images and potentially refactoring how they handle data I/O or security. Organizations without in-house K8s skills can struggle during this transition, and ignoring underlying performance tuning (e.g., how to handle Spark shuffle on K8s) can lead to issues[[46]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Moving%20to%20Kubernetes%20can%20put,defined%20plan%20and%20implementation%20strategy)[[47]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Additionally%2C%C2%A0many%20enterprises%20lack%20the%20prerequisite,reliability%2C%20can%20cause%20migration%20challenges). In short, adopting Spark on K8s should be done with careful planning.
* **Monitoring & Debugging Complexity:** Kubernetes adds another layer where things can go wrong (container runtime issues, scheduling delays, etc.), so monitoring Spark jobs on K8s can be more complex than on a standalone Spark cluster. Traditional Spark monitoring (Spark UI, logs) is still available, but you also need to monitor the Kubernetes side (pod health, cluster resource usage). Generic monitoring tools might not suffice for big data workloads[[48]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Monitoring%20and%20Alerting). It’s often necessary to employ specialized observability tools or dashboards to get insights into Spark-on-K8s performance. Debugging can involve both Spark logs and kubectl commands to inspect pods, which is a bit more involved than debugging on a single VM.
* **Security and Configuration Details:** By bringing Spark into Kubernetes, you inherit Kubernetes security concerns (need to set up proper RBAC, consider pod security policies, manage image vulnerabilities, etc.). For example, allowing Spark to use hostPath volumes or run with certain privileges can pose security risks if not managed[[49]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=Volume%20Mounts). There’s also more configuration to get right: e.g., giving the Spark driver the correct service account permissions to create pods[[50]](https://thulasitharan-gt96.medium.com/running-spark-on-kubernetes-k8s-9e613d64ae1d" \l ":~:text=Service%20account%20for%20spark%20%3A), ensuring Kubernetes DNS is configured for Spark executors to register, etc. Misconfigurations can cause Spark applications to fail to start.
* **Spark Version Constraints:** Kubernetes support in Spark became stable in Spark 3.1 (2021). Using older Spark versions (< 3.0) is not recommended for Kubernetes mode because critical features (like dynamic allocation and shuffle file tracking) were missing or unstable[[51]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Initial%20support%20for%20Spark%20on,We%20highly%20recommend%20using). So teams might need to upgrade Spark versions to leverage Kubernetes, which in itself can be a big change if you’re on a legacy version. The upside is that if you’re adopting Kubernetes, you’re likely also adopting the latest Spark improvements.

Despite these challenges, many organizations find that the benefits outweigh the cons, especially with the right tooling (for example, using the open-source Spark Operator or managed services to simplify the deployment). Over time, the ecosystem is addressing these pain points, making Spark-on-K8s easier to use.

### Deploying Apache Spark monolithic vs. Kubernetes

Below are **reference-grade sizing cheat-sheets** you can drop straight into your workshop deck or notes. They show rough-order-of-magnitude numbers that work well for most teams and point out where Kubernetes adds (or hides!) extra cost. Treat the figures as starting baselines—always profile and tune for your actual workload. Rules of thumb you can quote:

* **Memory:** Aim for executors ≤ 32 GiB; JVMs > 200 GiB behave poorly. Keep driver smaller than the largest executor. ([Apache Spark](https://spark.apache.org/docs/latest/hardware-provisioning.html))
* **CPU per node:** provision **8–16 cores min**; Spark scales almost linearly up to tens of cores. ([Apache Spark](https://spark.apache.org/docs/latest/hardware-provisioning.html))
* **Disk:** 4-8 local SSDs per node, mounted separately, no RAID, noatime option. ([Apache Spark](https://spark.apache.org/docs/latest/hardware-provisioning.html))
* **K8s overhead budgeting:** for managed services reserve **~6 % CPU of first core + 0.25 % above 4 cores** and **~5-25 % RAM on small nodes**; bigger nodes waste less. ([LearnKube](https://learnkube.com/allocatable-resources))
* **Minikube demo:** start with minikube start --cpus 3 --memory 4096 or Spark won’t fit even one executor. ([Apache Spark](https://spark.apache.org/docs/latest/running-on-kubernetes.html))

Use these tables as quick-reference slides, then dive into tuning demos during your video. They’ll help your data-engineering audience understand how much to ask for—and why Kubernetes’ numbers never line up one-for-one with their old YARN clusters.

### Baseline cluster sizing — monolithic vs. Kubernetes

| Work-load tier (≈ data processed / day) | Monolithic Spark (stand-alone or YARN) | Spark on Kubernetes\* | Why the difference? |
| --- | --- | --- | --- |
| **Small / Dev (< 100 GB)** | • 1 node (laptop/VM) **4 vCPU / 8 GiB** runs local[4]• For a “real” mini-cluster: **3 × (4 vCPU / 16 GiB)** workers | • minikube or kind single-node cluster: **3 vCPU + 4 GiB** minimum to launch driver + 1 executor ([Apache Spark](https://spark.apache.org/docs/latest/running-on-kubernetes.html)) | K8s needs a little extra for control-plane pods plus **pod overhead** (≈ 250 mCPU + 120 Mi) per Spark pod when RuntimeClass overhead is enabled ([Kubernetes](https://kubernetes.io/docs/concepts/scheduling-eviction/pod-overhead/)) |
| **Medium (0.1 – 1 TB)** | **10 × (16 vCPU / 64 GiB)** workersDriver 4 core / 8 GiBExecutors --num-executors 20 --executor-cores 4 --executor-memory 16g | Same 10 nodes, but K8s reserves ~**6–25 %** of RAM & 6 % of the 1st core, then 1 % of cores > 1 for system daemons (GKE/EKS rule-of-thumb) ([LearnKube](https://learnkube.com/allocatable-resources)) | Pods declare **requests & limits**; scheduler must fit requests + overhead onto allocatable node resources. |
| **Large (1 – 10 TB)** | **30 × (32 vCPU / 128 GiB)** workersDriver 8 core / 16 GiBExecutors ≈ 100 pods, 4 core / 32 GiB each | Same raw HW; K8s reserve on big nodes drops to ~**4 %** memory and ~0.25 % CPU/core ([LearnKube](https://learnkube.com/allocatable-resources)) | Larger nodes amortise K8s overhead; cluster-autoscaler can add/remove nodes elastically. |

\* Figures assume the Spark 4.0 official image and default JVM workloads; non-JVM executors (Python, GPU, etc.) need higher memoryOverhead (40 % default from Spark 3.3+) ([Apache Spark](https://spark.apache.org/docs/latest/configuration.html)).

### “Golden” Spark submit knobs

| Parameter | ****Typical start-point**** | K8s-specific note | Source |
| --- | --- | --- | --- |
| --executor-cores | **3 – 5** cores per executor (keeps GC & context-switch overhead down) | You can override with spark.kubernetes.executor.request.cores / limit.cores for fine-grained QoS. | ([Medium](https://medium.com/%40krthiak/apache-spark-is-powerful-but-like-any-high-performance-engine-it-needs-the-right-fuel-88ba4c5749f8)) |
| --executor-memory | **8 – 16 g** (scale with data; stay ≤ 32 g to avoid huge-page GC penalties) | Same value; container limit = memory + overhead. | ([Medium](https://medium.com/%40krthiak/apache-spark-is-powerful-but-like-any-high-performance-engine-it-needs-the-right-fuel-88ba4c5749f8), [Apache Spark](https://spark.apache.org/docs/latest/configuration.html)) |
| spark.executor.memoryOverhead | max(10 % of executor-mem, 384m) (Spark default) | **Must** be sized or pods OOM-kill; non-JVM images default to 40 %. | ([Apache Spark](https://spark.apache.org/docs/latest/configuration.html)) |
| --num-executors / spark.executor.instances | Small 2-4 • Med 20 • Large 100 (or enable dynamic allocation) | Most users enable **Dynamic Allocation** + K8s Cluster-Autoscaler. | ([Medium](https://medium.com/%40krthiak/apache-spark-is-powerful-but-like-any-high-performance-engine-it-needs-the-right-fuel-88ba4c5749f8)) |
| --driver-memory | Small 4 g • Med 8 g • Large 16 g | Add spark.kubernetes.authenticate.driver.serviceAccountName=<sa> so driver can create executor pods. | ([Apache Spark](https://spark.apache.org/docs/latest/running-on-kubernetes.html)) |
| K8s pod overhead | n/a | RuntimeClass-defined or default (~120 Mi + 250 mCPU/pod, counted by scheduler) | ([Kubernetes](https://kubernetes.io/docs/concepts/scheduling-eviction/pod-overhead/)) |
| Node reserve (K8s only) | n/a | GKE example: 25 % of first 4 GiB, 20 % next 4 GiB, etc. | ([LearnKube](https://learnkube.com/allocatable-resources)) |

### Common gotchas & cures

| Symptom | Where it shows up | Root cause | Quick fix / prevention |
| --- | --- | --- | --- |
| MemoryOverhead Exceeded **→ executor pod OOM-Killed** | Kubernetes | default 10 % overhead too small for large shuffles or Python + JVM mix | bump spark.executor.memoryOverhead (e.g. 4096m) or set higher memoryOverheadFactor; monitor with kubectl describe pod ([Stack Overflow](https://stackoverflow.com/questions/77121410/solving-oom-issue-for-a-spark-job?utm_source=chatgpt.com)) |
| ImagePullBackOff **/ pod stuck** Pending | Kubernetes | wrong image tag, private registry auth, or local build not pushed | verify image:, run kubectl describe pod, add imagePullSecrets, or use a local registry for dev ([Lumigo](https://lumigo.io/kubernetes-troubleshooting/kubernetes-imagepullbackoff/?utm_source=chatgpt.com)) |
| Pod Pending – no node has resources | Kubernetes | requests + overhead larger than allocatable (node reserve!) | lower executor mem/cores, enable Cluster-Autoscaler, or pick bigger node type ◆ |
| Straggler tasks / long GC pauses | Both | too many cores per executor or heap too large | keep executor ≤ 5 cores, ≤ 32 GiB; switch to G1GC (spark.executor.extraJavaOptions=-XX:+UseG1GC) ([Dev Genius](https://blog.devgenius.io/from-out-of-memory-to-optimized-handling-java-heap-space-gc-overhead-limit-exceeded-issues-in-61f35bb253da?utm_source=chatgpt.com)) |
| GC-overhead / java.lang.OutOfMemoryError | Both | storage/execution memory not balanced | raise executor memory, tune spark.memory.fraction or cache less data; prefer **AQE** for skew joins ([Stack Overflow](https://stackoverflow.com/questions/27462061/why-does-spark-fail-with-java-lang-outofmemoryerror-gc-overhead-limit-exceeded?utm_source=chatgpt.com)) |
| Cluster under-utilised (monolithic) | Stand-alone / YARN | static partitioning & long cluster startup | enable **dynamic allocation** or migrate to Kubernetes shared pool |
| Shuffle network bottleneck | Both (large jobs) | disks slow or 1 GbE network | use SSD temp space (spark.local.dir on NVMe), ensure 10 GbE+ NICs; on K8s, consider push-based shuffle service |

## Hands-On Demonstration: Running Spark on Kubernetes

Now let’s outline a simple **hands-on demo** to illustrate Spark on Kubernetes. The idea is to run a Spark job on a local Kubernetes cluster (using open-source tools) that you can also translate to a production setup conceptually. We’ll use **Minikube** (which is a local single-node Kubernetes) for the demo, since it’s free and self-contained.

**1. Set up a Local Kubernetes Cluster:** Install Minikube and kubectl on your machine. Minikube runs a single-node Kubernetes cluster locally (you can also use Docker Desktop’s built-in K8s or KinD). Start Minikube with enough resources for Spark – the default may be too low. For example, allocate at least 3 CPUs and 4GB of RAM to Minikube to run a simple Spark job[[52]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,list%2C%20create%2C%20edit%20and%20delete):

minikube start --cpus 3 --memory 4096

This will spin up the K8s cluster. Verify it’s running with kubectl cluster-info and ensure you can connect (kubectl get nodes should show the Minikube node)[[53]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=To%20verify%20the%20cluster%20is,up%20and%20running%2C%20use)[[54]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Also%2C%20make%20sure%20kubectl%20is,to%20communicate%20with%20your%20cluster).

**2. Prepare a Spark Docker Image:** Spark on K8s requires a container image for the Spark driver and executors that includes Spark itself. You have a few options: - **Build your own Spark image:** The Apache Spark distribution comes with Dockerfiles and a script to build images. For instance, in Spark’s kubernetes/dockerfiles directory, there’s a docker-image-tool.sh that you can use to build an image with Spark binaries[[55]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=Docker%20Images)[[56]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=Spark%20also%20ships%20with%20a,use%20with%20the%20Kubernetes%20backend). You might run:

./bin/docker-image-tool.sh -t spark-k8s-demo -r <your\_dockerhub\_username> build

which packages the Spark distribution into an image tagged spark-k8s-demo. (Make sure Docker is installed and running on your machine for this step.) - **Use a pre-built image:** To save time, you can use an existing Spark image from a repository. For example, the Spark Operator project provides ready-to-use images for Spark 3.x (e.g., gcr.io/spark-operator/spark-py:v3.1.1 for PySpark, or an equivalent Scala image). Using a pre-built image means you don’t have to build one from scratch for the demo.

For our demo, let’s assume we use a pre-built image that matches our Spark version. Either way, note the image name/tag – we’ll need it when submitting the job.

**3. Configure Kubernetes Permissions:** By default, your Spark driver pod will run in the **default namespace** (or you can choose another) and use the default service account. Kubernetes, however, **restricts permissions** by default. We need to ensure the Spark driver pod can create other pods (executors) and manage them. The simplest approach in a local setup is to give the default service account extended privileges, or create a dedicated service account for Spark: - Create a dedicated namespace (e.g., spark) and a service account in it:

kubectl create namespace spark   
kubectl create serviceaccount spark -n spark

- Bind a role to this service account that allows creating pods and services. For example, you can give it the built-in edit role on the namespace, which is often used for applications. A quick way is:

kubectl create clusterrolebinding spark-role --clusterrole=edit \  
 --serviceaccount=spark:spark --namespace=spark

This grants the “spark” service account permission to create/edit resources in the “spark” namespace[[57]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=apiVersion%3A%20v1%20kind%3A%20ServiceAccount%20metadata%3A,io%2Fv1%20kind%3A%20ClusterRoleBinding%20metadata)[[58]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=name%3A%20spark,io). (In a real production cluster, you’d tighten these permissions.) - If you choose to use the default namespace and default service account, you could similarly give it a ClusterRoleBinding to allow pod creation. But isolating in a separate namespace is cleaner.

**4. Submit a Spark Job to Kubernetes:** With the cluster up, image built, and permissions set, we can run a Spark job. We’ll use Spark’s built-in **Spark Pi** example for simplicity. This job computes an approximation of π. Use spark-submit from your Spark installation, specifying the Kubernetes master URL and the image:

```bash

$SPARK\_HOME/bin/spark-submit \ --master k8s://https://<MINIKUBE-CLUSTER-IP>:8443 \ --deploy-mode cluster \ --name spark-pi-demo \ --class org.apache.spark.examples.SparkPi \ --conf spark.executor.instances=2 \ --conf spark.kubernetes.container.image=<your-spark-image> \ --conf spark.kubernetes.namespace=spark \ --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \ local:///opt/spark/examples/jars/spark-examples\_2.12-3.1.1.jar

```

Let’s break down what this does:  
 - `--master k8s://https://...` points Spark to the Kubernetes API server. (You can get the exact URL from `kubectl cluster-info`.)[59] For Minikube, it’s usually something like `https://127.0.0.1:xxxxx` or a similar address.  
 - `--deploy-mode cluster` means the Spark driver itself will run on Kubernetes (not on your local machine).  
 - We give the app a name and the main class for SparkPi (which is included in Spark’s examples jar).  
 - We specify `spark.executor.instances=2` to run with 2 executors (you can adjust as resources allow).  
 - The key config is `spark.kubernetes.container.image=<image>` – this is the Docker image we built or chose, which has Spark in it. Both the driver and executor pods will use this image (unless you specify separate images for driver/executor).  
 - We also specify the namespace and service account so that the driver uses the correct namespace and permissions[60][57].  
 - The application jar is referenced as `local:///opt/spark/examples/jars/...` – note that `local://` means “inside the image”. We know the Spark examples jar is included in the Spark image at that path. (If running a custom app, you’d build your app jar and either bake it into the image or use a remote location like a cloud storage URL.)  
  
Once you run this command, Spark will contact Kubernetes to start the driver pod. If all goes well, Kubernetes will create a pod named “spark-pi-demo-driver” (or similar) in the `spark` namespace and soon after spawn two executor pods.  
  
**5. Monitor the Spark Application**: You can observe what’s happening using Kubernetes commands:  
- List the pods to see the driver and executors coming up:   
 ```bash  
 kubectl get pods -n spark  
 ```   
 You should see something like `spark-pi-demo-driver` and `spark-pi-demo-executor-1` (and -2) with their statuses. Initially they might be ContainerCreating, then Running[61].  
- To follow the progress, you can check the driver pod’s logs:   
 ```bash  
 kubectl logs -f spark-pi-demo-driver -n spark  
 ```   
 This will stream the Spark driver output. In there you’ll see the typical Spark logs. Eventually, you should see the Spark Pi output, e.g., a line with `Pi is roughly 3.14...`.  
- If you enabled it, you could also access the Spark UI. By default in cluster mode, the driver’s Spark UI is port-forwardable. For example, you could do:

```bash  
 kubectl port-forward svc/spark-pi-demo-ui-svc 4040:4040 -n spark  
 ```   
 (This assumes Spark’s driver exposed a UI service – the Spark Operator does this automatically via a service. If using vanilla spark-submit, you might need to set up port-forward to the driver pod’s UI port manually.) Then you could open http://localhost:4040 and view the Spark UI for the running job[62]. For a quick demo, however, checking logs is usually enough.  
  
The Spark Pi job is short, so within a minute the job should finish. The executors will exit and their pods will go to Completed status then disappear. The driver pod will likely stay in Completed status for a while (so you can still fetch its logs even after finish).  
  
**6. Cleanup:** For a local demo, cleanup is straightforward. You can delete the driver pod (it will eventually be removed anyway by Kubernetes GC):

```bash  
kubectl delete pod <driver-pod-name> -n spark

```

If you created a namespace for Spark, you can delete the whole namespace to clean up everything in one go:

```bash

kubectl delete namespace spark

```

And finally, you can stop Minikube (minikube stop`) if you’re done. In production, cleanup would involve terminating jobs or allowing Kubernetes to garbage collect completed pods, and perhaps cleaning up any persistent resources you created.

**Note:** This demonstration was done on a single-node cluster for simplicity. In a production environment, you would use a multi-node Kubernetes cluster (possibly a managed service like Amazon EKS, Google GKE, or Azure AKS). The steps conceptually remain the same – you’d build a Spark image, ensure the cluster has the needed RBAC for Spark, and submit jobs. For production usage, you might also deploy the **Spark Operator** (an open-source Kubernetes operator by Google) which allows you to create a SparkApplication YAML (as shown in some tutorials) to run Spark jobs more natively in Kubernetes[[63]](https://medium.com/@kerrache.massipssa/deploy-apache-spark-with-kubernetes-k8s-fb73a748c09d" \l ":~:text=In%20the%20example%20below%2C%20we,the%20YAML%20manifest%20file%20below)[[64]](https://medium.com/@kerrache.massipssa/deploy-apache-spark-with-kubernetes-k8s-fb73a748c09d" \l ":~:text=mode%3A%20cluster%20image%3A%20%22gcr.io%2Fspark,OnFailure%20onFailureRetries%3A%203%20onFailureRetryInterval%3A%2010). The operator can simplify submission and manage retries, monitoring, etc., but it’s an advanced tool beyond this initial demo.

## Conclusion

In this workshop, we covered how containerization and Kubernetes can modernize the way you run Apache Spark: - Containers solve the deployment headaches by packaging Spark jobs into portable, consistent units that run anywhere. - Kubernetes provides the automation to deploy and manage these containers at scale, bringing benefits of better resource utilization, flexibility, and integration into cloud-native ecosystems. - Running Spark on Kubernetes offers clear **advantages** in efficiency, isolation, and operational consistency, which is why many organizations are adopting it. Studies and industry trends show Kubernetes is likely to become the default resource manager for Spark in the cloud[[15]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=On%20the%20other%20hand%2C%20Kubernetes,main%20big%20data%20processing%20engine)[[65]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Additional%20benefits%20of%20running%20Spark,scale%20data%20analytics). - However, there are also non-trivial **challenges** – primarily the added complexity of Kubernetes itself. Data engineers may need to collaborate with DevOps teams or acquire new skills to successfully use Spark on K8s. Good planning, tooling, and possibly managed services can help bridge that gap.

By walking through a hands-on example, you’ve seen the end-to-end process of deploying a Spark job on Kubernetes using open-source tools (Minikube, Docker, etc.). With this knowledge, you can experiment further – for instance, try running your own Spark application on Kubernetes, or explore the Spark Operator for a more production-ready approach.

Spark on Kubernetes is a powerful combination that aligns with modern DevOps practices: it enables **DataOps** teams to deliver big data processing with greater agility and consistency. As cloud-native technologies continue to evolve, mastering Spark on K8s will be a valuable skill for the next generation of data engineers. Happy Sparking on K8s!

**Sources**

The information and best practices above were gathered from Apache Spark documentation and community experts[[52]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,list%2C%20create%2C%20edit%20and%20delete)[[3]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=This%20is%20the%20main%20motivation,iterate%20on%20your%20code%20faster)[[10]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=As%20a%20consequence%2C%20many%20platforms,compute%20resources%20with%20this%20approach)[[14]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=There%E2%80%99s%20no%20Spark%20vs%20Kubernetes,results%20in%20inefficient%20task%20management)[[27]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=When%20you%20run%20Spark%20on,and%20implement%20the%20application%20code)[[42]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Initial%20deployment%20can%20be%20among,including%20Spark%20on%20K8s%20adoption)[[66]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=%24SPARK_HOME%2Fbin%2Fspark,conf%20spark.kubernetes.namespace%3Dspark), as well as real-world case studies on Spark-on-Kubernetes deployments[[17]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Second%2C%20recent%20Spark%20versions%20,performance%20improvement%20for%20Spark%20workloads)[[12]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=On%20other%20cluster,have%20to%20compromise%20on%20isolation).

[[1]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Here%20are%20some%20few%20key,benefits) [[2]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,a%20Spark%20application%20if%20needed) [[5]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=4) [[19]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=6,Ecosystem) [[20]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,Jenkins%20X%20or%20GitLab%20CI) [[22]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=5,Other%20Applications) [[32]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=4) [[34]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Kubernetes%20excels%20at%20managing%20cluster,When%20running%20Spark%20on%20Kubernetes) [[35]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=,efficiently%2C%20improving%20overall%20cluster%20utilization) [[38]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=7) [[39]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=or%20the%20ELK%20,Jenkins%20X%20or%20GitLab%20CI) [[53]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=To%20verify%20the%20cluster%20is,up%20and%20running%2C%20use) [[54]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Also%2C%20make%20sure%20kubectl%20is,to%20communicate%20with%20your%20cluster) [[57]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=apiVersion%3A%20v1%20kind%3A%20ServiceAccount%20metadata%3A,io%2Fv1%20kind%3A%20ClusterRoleBinding%20metadata) [[58]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=name%3A%20spark,io) [[60]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=spark.master%20k8s%3A%2F%2Fhttps%3A%2F%2F%3Ckubernetes,serviceAccountName%20spark) [[61]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=Run%20the%20command%20below%20to,the%20details%20about%20your%20pods) [[66]](https://www.chaosgenius.io/blog/spark-on-kubernetes/" \l ":~:text=%24SPARK_HOME%2Fbin%2Fspark,conf%20spark.kubernetes.namespace%3Dspark) HOW TO: Configure and Run Apache Spark on Kubernetes (2025)

<https://www.chaosgenius.io/blog/spark-on-kubernetes/>

[[3]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=This%20is%20the%20main%20motivation,iterate%20on%20your%20code%20faster) [[4]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=1,to%20less%20than%2030%20seconds) [[6]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Our%20favorite%20benefit%20is%20definitely,libraries%20on%20each%20application%20launch) [[9]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=2,big%20cost%20savings) [[10]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=As%20a%20consequence%2C%20many%20platforms,compute%20resources%20with%20this%20approach) [[11]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=running%20transient%20clusters%20for%20production,compute%20resources%20with%20this%20approach) [[12]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=On%20other%20cluster,have%20to%20compromise%20on%20isolation) [[13]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Image%20%20100Spark,managing%20clusters%2C%20just%20run%20containers) [[18]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Deploying%C2%A0Spark%20on%20Kubernetes%C2%A0gives%20you%20powerful,grained%20security%20and%20data%20access) [[21]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=If%20you%20have%20a%20need,in) [[23]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=You%20can%20submit%20Spark%20apps,and%20the%20Spark%20configuration) [[24]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Kubernetes%20takes%20this%20request%20and,pools%20to%20meet%20the%20requirements) [[25]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=,anticipate%20spot%20kills%20and%20gracefully) [[29]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=dependencies%20to%20run%20,and%20the%20Spark%20configuration) [[30]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=To%20go%20a%20little%20deeper%2C,lives%20in%20the%20Spark%20driver) [[33]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=On%20other%20cluster,have%20to%20compromise%20on%20isolation) [[36]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Despite%20the%20raw%20performance%20of,Kubernetes%20%28Data%20Mechanics) [[40]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=The%20Neutral%20%E2%80%93%20Spark%20performance,is%20the%20same) [[41]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=1.%C2%A0Making%20Spark,requires%20build%20time%20and%20expertise) [[44]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=introduces%20can%20be%20frightening%20and,away%20from%20your%20core%20mission) [[45]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=,your%20notebooks%20and%2For%20your%20scheduler) [[51]](https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/" \l ":~:text=Initial%20support%20for%20Spark%20on,We%20highly%20recommend%20using) The Pros and Cons of Running Apache Spark on Kubernetes | Spot.io

<https://spot.io/blog/the-pros-and-cons-of-running-apache-spark-on-kubernetes/>

[[7]](https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b?gi=0bec52493821" \l ":~:text=Kubernetes%20in%20a%20nutshell%20is,containerized%20applications%20of%20varying%20sizes) [[8]](https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b?gi=0bec52493821" \l ":~:text=Figure%201%E2%80%931%3A%20Deploy%2C%20auto,Kubernetes%20from%20What%20is%20Kubernetes) Spark on Kubernetes. A Hands-On Introduction: Getting Up and… | by Scott Haines | Level Up Coding

<https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b?gi=0bec52493821>

[[14]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=There%E2%80%99s%20no%20Spark%20vs%20Kubernetes,results%20in%20inefficient%20task%20management) [[15]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=On%20the%20other%20hand%2C%20Kubernetes,main%20big%20data%20processing%20engine) [[16]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Second%2C%20recent%20Spark%20versions%20,performance%20improvement%20for%20Spark%20workloads) [[17]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Second%2C%20recent%20Spark%20versions%20,performance%20improvement%20for%20Spark%20workloads) [[27]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=When%20you%20run%20Spark%20on,and%20implement%20the%20application%20code) [[31]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Businesses%20are%20embracing%20a%20Spark,processes%20and%20shorten%20deployment%20time) [[37]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=How%20does%20this%20cost%20reduction,resources%20to%20be%20moved%20elsewhere) [[42]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Initial%20deployment%20can%20be%20among,including%20Spark%20on%20K8s%20adoption) [[43]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Pepperdata%E2%80%99s%20State%20of%20Kubernetes%202023,including%20Spark%20on%20K8s%20adoption) [[46]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Moving%20to%20Kubernetes%20can%20put,defined%20plan%20and%20implementation%20strategy) [[47]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Additionally%2C%C2%A0many%20enterprises%20lack%20the%20prerequisite,reliability%2C%20can%20cause%20migration%20challenges) [[48]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Monitoring%20and%20Alerting) [[65]](https://www.pepperdata.com/blog/spark-on-kubernetes/" \l ":~:text=Additional%20benefits%20of%20running%20Spark,scale%20data%20analytics) Spark on Kubernetes: A Quick Guide to Get Started | Pepperdata

<https://www.pepperdata.com/blog/spark-on-kubernetes/>

[[26]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,collected%20or%20manually%20cleaned%20up) [[28]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,collected%20or%20manually%20cleaned%20up) [[49]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=Volume%20Mounts) [[52]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=,list%2C%20create%2C%20edit%20and%20delete) [[55]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=Docker%20Images) [[56]](https://spark.apache.org/docs/latest/running-on-kubernetes.html" \l ":~:text=Spark%20also%20ships%20with%20a,use%20with%20the%20Kubernetes%20backend) Running Spark on Kubernetes - Spark 4.0.0 Documentation

<https://spark.apache.org/docs/latest/running-on-kubernetes.html>

[[50]](https://thulasitharan-gt96.medium.com/running-spark-on-kubernetes-k8s-9e613d64ae1d" \l ":~:text=Service%20account%20for%20spark%20%3A) [[59]](https://thulasitharan-gt96.medium.com/running-spark-on-kubernetes-k8s-9e613d64ae1d" \l ":~:text=Getting%20k8s%20URL%3A) Running spark on kubernetes (k8s) | by Thulasitharan Govindaraj | Medium

<https://thulasitharan-gt96.medium.com/running-spark-on-kubernetes-k8s-9e613d64ae1d>

[[62]](https://medium.com/@kerrache.massipssa/deploy-apache-spark-with-kubernetes-k8s-fb73a748c09d" \l ":~:text=After%20deploying%20the%20application%2C%20a,port%20to%20your%20local%20environment) [[63]](https://medium.com/@kerrache.massipssa/deploy-apache-spark-with-kubernetes-k8s-fb73a748c09d" \l ":~:text=In%20the%20example%20below%2C%20we,the%20YAML%20manifest%20file%20below) [[64]](https://medium.com/@kerrache.massipssa/deploy-apache-spark-with-kubernetes-k8s-fb73a748c09d" \l ":~:text=mode%3A%20cluster%20image%3A%20%22gcr.io%2Fspark,OnFailure%20onFailureRetries%3A%203%20onFailureRetryInterval%3A%2010) Deploy Apache Spark with Kubernetes (K8s) | by Kerrache Massipssa | Medium

<https://medium.com/@kerrache.massipssa/deploy-apache-spark-with-kubernetes-k8s-fb73a748c09d>