# DevOps for Data: Spark Delivery on Containers

### 30‑minute workshop deck (15 min talk • 15 min demo)

**Presenter:** Muhammad Danyal “Sage” Khan  
**For:** CD Foundation / Linux Foundation Workshops  
**Date:** Aug 28

## Agenda & Timeboxes

* **0–2 min** Hook & goals
* **2–6 min** Why CD for data (not “just K8s”)
* **6–9 min** Containers 101 for Spark
* **9–12 min** Why Kubernetes for Spark
* **12–15 min** Architecture & CD blueprint
* **15–30 min** **Live demo**: commit→build→test→deploy→logs→(rollback)

Speaker note: keep tempo brisk; save questions for the last 2 minutes or Slack.

## Opening Hook (Poll)

**Which is your biggest pain today?**  
- A) “It works on my laptop” drift  
- B) Dependency hell (Py/Spark/native libs)  
- C) No promotion/rollback story  
- D) Costs & idle clusters

**Promise:** after this session, you’ll have a **repeatable CD pattern** for Spark that fixes A–D.

## Learning Objectives (What you’ll take away)

* A **DevOps blueprint** for Spark: commit → artifact → checks → promotion → safe rollout
* How to **package** Spark apps as **immutable images**
* How to add **quality gates** (code **and** data)
* How to **orchestrate** on containers (Kubernetes concrete)
* A **local demo** you can reproduce, map to prod later

## Why CD for Data (not “just Kubernetes”)

* Spark jobs are **releases** that deserve versioning, promotion & rollback
* Data pipelines need **traceability** (who/what ran, with which data + deps)
* CD brings **auditability, speed, and safety** to analytics just like to apps

**Takeaway:** K8s is the runtime; **CD is the operating model**.

## Containers 101 for Spark (quick)

* **Bundle once**: Spark + Py deps + configs ⇒ **one image** per job
* **Run anywhere**: laptop → CI → cluster (same bits)
* **Isolation**: different jobs, different versions, no conflicts

Visual: width:900

## Why Kubernetes for Spark (in one slide)

* **Scheduling/Auto‑healing** of driver/executors (pods)
* **Shared pool** → higher utilization vs. per‑job clusters
* **RBAC/Namespaces** for multi‑tenancy & quotas
* **Ecosystem**: Prometheus/Grafana, ELK, GitOps

Analogy: K8s is the **warehouse manager**; containers are the **boxes**.

## Diagram: Spark on Kubernetes (mental model)

+------------------ Kubernetes Cluster ------------------+  
| Namespace: spark |  
| |  
| [Job] spark-pi ---> [Driver Pod] <--- service acct |  
| | |  
| +----------------+----------------+ |  
| | | |  
| [Executor Pod] [Executor Pod] |  
| (cores/mem) (cores/mem) |  
| |  
+-------------------------------------------------------+

## CD Blueprint (diagram)

Git push  
 │  
 ├──▶ CI: build image (Spark+deps)  
 │ ├─ unit tests  
 │ ├─ data tests (pandera/GE)  
 │ └─ SBOM & (opt) sign  
 │  
 └──▶ CD: apply manifest (Job/CronJob/SparkApplication)  
 └─ Observe (logs/metrics) & rollback

**CDF fit:** Tekton/Jenkins pipelines • CDEvents for triggers • SBOM/provenance inventory

## Facts & Figures (cheat‑sheet)

| Tier | Driver (vCPU/RAM) | Executor (vCPU/RAM) | Count | Notes |
| --- | --- | --- | --- | --- |
| **Dev (<100 GB)** | 1 / 2 GiB | 1 / 2 GiB | 2–3 | kind/minikube ≥3 vCPU, 4 GiB; pod overhead applies |
| **Medium (0.1–1 TB)** | 2–4 / 4–8 GiB | 3–4 / 8–16 GiB | 10–20 | ≤5 cores/executor; memoryOverhead ≥10–25% |
| **Large (1–10 TB)** | 4–8 / 8–16 GiB | 4–5 / 24–32 GiB | 50–100 | SSD local dirs; dynamic allocation + autoscaler |

**Rules:** executors ≤32 GiB • ≤5 cores/executor • budget for K8s node reserve & pod overhead

## Golden Spark Knobs (start points)

* --executor-cores: **3–5**
* --executor-memory: **8–16g** (keep ≤32g)
* spark.executor.memoryOverhead: **≥10%** (25–40% for Py/shuffle heavy)
* spark.executor.instances: small **2–4**, med **20, large 100**
* --driver-memory: small **4g**, med **8g**, large **16g**

Tip: prefer **G1GC**; avoid giant heaps; watch shuffle temp space.

## Risks & Fixes (visual table)

| Symptom | Root cause | Quick fix |
| --- | --- | --- |
| Pods Pending | requests + overhead > allocatable | Right‑size mem/cores; autoscale; bigger nodes |
| Executor OOMKilled | overhead too low; Py/shuffle heavy | Raise memoryOverhead; ≤5 cores/executor |
| Slow start | cold images | Pre‑pull; slimmer base |
| Stragglers/GC | too many cores/heap | ≤5 cores; ≤32 GiB; G1GC |

## Demo Plan (15 minutes)

**Goal:** end‑to‑end locally, offline‑friendly

1. **Cluster:** make kind-up
2. **Build:** make build (Spark app image)
3. **(Optional tests):** pip install -r requirements-dev.txt && make test
4. **Load image:** make load (no registry)
5. **Deploy:** make deploy-local
6. **Observe:** kubectl logs job/spark-pi -n spark -f
7. **Cleanup:** make kind-down

Fallback: Minikube path make -f Makefile.minikube deploy-local.

## Demo: What to Show on Screen

* kubectl get pods -n spark -w (driver → executors lifecycle)
* kubectl describe job spark-pi -n spark (conditions)
* **Log line milestone:** Pi is roughly 3.14...
* (Optional) tweak spark.executor.instances from 2 → 3 and re‑deploy

## Observability (quick)

* Spark **metrics → Prometheus** (JMX exporter)
* **Logs** to ELK/Cloud (correlated by job name/run id)
* **SLOs:** job latency, success %, cost/run

Diagram: width:950

## Platform Guardrails (for big orgs)

* Namespaces & **quotas**, LimitRange, admission policies
* **Secrets**: Vault/ESO, least‑privilege SAs
* **Cost**: requests/limits, VPA (drivers), autoscaler, TTL
* **Promotion**: dev→stage→prod via GitOps; CDEvents trail

## Mapping to Production

Local: kind + Job + embedded spark-submit  
Prod: EKS/GKE/AKS + Spark Operator **(SparkApplication)** or Argo Workflows  
Add: CI gates (tests, SBOM, signing), GitOps, SSO/RBAC, autoscaling

## Recap — What you can do today

* Clone the **starter repo**
* Run the **local demo**
* Add **data tests** (pandera / GE)
* Wire CI/CD (Tekton/Jenkins or GitHub Actions)
* Plan **guardrails** with your platform team

## Q&A (2 minutes)

Share your org’s top blocker; we’ll map it to the blueprint.

## Backup: Command Sheet (copy & paste)

make kind-up  
make build && make load && make deploy-local  
kubectl get pods -n spark -w  
kubectl logs job/spark-pi -n spark -f  
make kind-down

## Backup: Manifest Snippet (Job)

apiVersion: batch/v1  
kind: Job  
metadata: {name: spark-pi, namespace: spark}  
spec:  
 template:  
 spec:  
 restartPolicy: Never  
 serviceAccountName: spark-runner  
 containers:  
 - name: driver  
 image: spark-app:dev  
 command: ["/opt/spark/bin/spark-submit"]  
 args: ["--master","k8s://https://kubernetes.default.svc",  
 "--deploy-mode","cluster",  
 "--conf","spark.kubernetes.container.image=spark-app:dev",  
 "--conf","spark.executor.instances=2",  
 "local:///opt/app/app.py"]