# **Elementos Básicos do Kubernetes**

## **Módulo 3: Spark-as-a-Services [SpaaS] & Spark no Kubernetes [SPOK]**

### **Introdução**

Em 2023, 89% das empresas Fortune 500 executam Apache Spark em ambientes Kubernetes, representando um crescimento de 300% em apenas 3 anos. Esta mudança massiva demonstra a transformação do Spark de uma ferramenta on-premises para uma solução cloud-native.

Para arquitetos de dados modernos, dominar Kubernetes tornou-se essencial para gerenciar workloads de Big Data em escala. A habilidade de orquestrar aplicações Spark em Kubernetes não é mais um diferencial - é um requisito fundamental.

No contexto do UberEats, o processamento de 10 milhões de pedidos diários exigia uma solução que oferecesse:

* Escalabilidade automática durante horários de pico
* Redução de 60% nos custos de infraestrutura
* Tempo de deployment reduzido de horas para minutos
* Resiliência para zero downtime

**Objetivos de Aprendizado:**

* Compreender a arquitetura fundamental do Kubernetes para workloads Spark
* Configurar e otimizar ambientes Spark em Kubernetes para produção
* Implementar estratégias de scaling e resiliência
* Aplicar best practices para operações em escala

### **Contexto Cloud-Native**

A evolução do Spark reflete a transformação geral da indústria:

* **2014-2016**: Spark dominava clusters on-premises (Hadoop/YARN)
* **2017-2019**: Migração inicial para cloud com EMR/Dataproc
* **2020-2023**: Kubernetes emerge como padrão para orquestração
* **2024**: 70% dos novos deployments Spark usam Kubernetes

**Forças que impulsionam a adoção:**

* Elasticidade: Scale de 0 a 1000 nós em minutos
* Custo: Pay-per-use reduz TCO em até 75%
* Portabilidade: Mesmo código roda em AWS/Azure/GCP
* Automação: Infrastructure as Code simplifica operações

### **Fundamentos Arquiteturais**

**Anatomia de uma solução Spark-on-K8s:**

1. **Control Plane**
   * API Server: Interface central para todas as operações
   * etcd: Armazena estado do cluster
   * Scheduler: Aloca pods aos nós
   * Controller Manager: Mantém estado desejado
2. **Data Plane**
   * Worker Nodes: Executam containers Spark
   * Kubelet: Agente em cada nó
   * Kube-proxy: Gerencia networking
   * Container Runtime: Docker/containerd
3. **Spark Components**
   * Driver Pod: Coordena execução
   * Executor Pods: Processam dados
   * Service Accounts: Gerenciam permissões
   * ConfigMaps/Secrets: Configurações e credenciais

### **Estratégias de Implementação**

**Arquitetura de Referência para Produção:**

apiVersion: v1

kind: ServiceAccount

metadata:

name: spark-service-account

---

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: spark-role

rules:

- apiGroups: [""]

resources: ["pods", "services"]

verbs: ["get", "list", "create", "delete"]

**Decisões Críticas de Design:**

* Node Pools dedicados vs compartilhados
* Persistent volumes para shuffle data
* Network policies para segurança
* Resource quotas por namespace

**Configurações para Performance:**

spark.kubernetes.executor.request.cores: "4"

spark.kubernetes.executor.limit.cores: "8"

spark.kubernetes.memoryOverheadFactor: "0.1"

spark.dynamicAllocation.enabled: "true"

### **Operações e Observabilidade**

**Stack de Monitoramento:**

* Prometheus: Coleta métricas
* Grafana: Visualização em real-time
* Elasticsearch: Indexação de logs
* Jaeger: Distributed tracing

**Métricas Essenciais:**

* CPU/Memory utilization por pod
* Shuffle read/write metrics
* Task completion rates
* Network I/O patterns

**Estratégia de Alertas:**

* Driver failures: Critical (PagerDuty)
* Resource exhaustion: Warning (Slack)
* Performance degradation: Info (Email)

### **Aplicação ao UberEats**

**Pipeline de Processamento de Pedidos:**

1. **Ingestão em Tempo Real**
   * Kafka consumers em pods dedicados
   * Auto-scaling baseado em lag
   * 100k eventos/segundo
2. **Processamento Spark**
   * Streaming jobs para status updates
   * Batch jobs para analytics
   * ML pipelines para ETA prediction
3. **Otimização de Custos**
   * Spot instances para executors
   * Scheduled scaling para horários de pico
   * Resource sharing entre ambientes

### **Escalabilidade e Performance**

**Padrões de Auto-scaling:**

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: spark-executor-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: spark-executor

minReplicas: 3

maxReplicas: 100

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

**Benchmarks de Referência:**

* Startup time: 30s (vs 5min on-premises)
* Scale-out: 100 nodes em 2 minutos
* Cost efficiency: $0.05/GB processado

### **Comparação de Plataformas**

| **Critério** | **AWS EKS** | **Azure AKS** | **GCP GKE** |
| --- | --- | --- | --- |
| Spark Integration | EMR on EKS | HDInsight | Dataproc |
| Managed Services | ✓ | ✓ | ✓ |
| Spot Support | ✓ | ✓ | ✓ |
| Auto-scaling | HPA/VPA | KEDA | HPA/VPA |
| Cost | $$ | $$ | $ |
| Performance | High | Medium | High |

**Critérios de Seleção:**

* Existing cloud footprint
* Regional availability
* Support requirements
* TCO analysis

### **Governança e DevOps**

**CI/CD Pipeline:**

graph LR

A[Git Push] --> B[Build Image]

B --> C[Run Tests]

C --> D[Security Scan]

D --> E[Deploy to K8s]

E --> F[Smoke Tests]

F --> G[Production]

**GitOps com ArgoCD:**

* Declarative deployments
* Automated rollbacks
* Multi-cluster support
* Audit trail

### **Evolução e Tendências**

**Serverless Spark (2024-2025):**

* AWS EMR Serverless
* Azure Synapse Serverless
* GCP Dataproc Serverless

**Tecnologias Emergentes:**

* eBPF para observability
* WebAssembly para portability
* Service Mesh integration
* GPU scheduling improvements

### **Conclusão e Reflexão**

**Conceitos-chave para Produção:**

* Kubernetes abstrai complexidade de infraestrutura
* Auto-scaling é fundamental para cost efficiency
* Observability previne problemas antes que impactem
* GitOps garante deployments consistentes

**Checklist de Implementação:**

* [ ] Service accounts e RBAC configurados
* [ ] Resource limits definidos
* [ ] Monitoring stack deployado
* [ ] Auto-scaling policies testadas
* [ ] Backup e disaster recovery planejados
* [ ] Security policies implementadas

### **Recursos Complementares**

**Documentação Oficial:**

* kubernetes.io/docs
* spark.apache.org/docs/latest/running-on-kubernetes.html
* cloud.google.com/kubernetes-engine/docs

**Ferramentas Recomendadas:**

* Helm: Package manager para K8s
* Kustomize: Customização de configs
* Kubecost: Análise de custos
* Lens: IDE para Kubernetes

**Comunidades:**

* CNCF Slack (#spark-on-k8s)
* Stack Overflow (tags: apache-spark, kubernetes)
* Reddit r/kubernetes
* Meetups locais de Kubernetes