3. Hands-On Spark Operator

Tutorial Com ArgoCD

Download do Helm Chart para o Spark Operator

Link: https://github.com/kubeflow/spark-operator/tree/master

1. Adicionar o repositório helm
    1. helm repo add spark-operator https://kubeflow.github.io/spark-operator
    2. helm repo update
2. Criar um folder dentro da pasta dos helm-charts e colocar os arquivos baixados
3. O values.yaml pode manter padrão, alterando somente o namespace padrão dos jobs de default para processing

Criação da Aplicação Yaml do ArgoCD

1. Construir arquivo manifesto apontando para a pasta configurada

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: spark-operator-datawaybr
  namespace: argocd
spec:
  project: default
  source:
    repoURL: 'git@github.com:Alexno9/k8s-argo-minio.git'
    path: helm-charts/spark-operator
    targetRevision: main
    helm:
      valueFiles:
        - values.yaml
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: processing
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

2. Realizar apply do manifesto: `kubectl apply -f arquivo.yaml -n argocd`
3. Acompanhar a subida do operador via argocd

Tutorial via Helm Manual

Instalação do Helm Chart Padrão

1. helm install spark-operator spark-operator/spark-operator -n processing --wait
    1. Instalação default sem modificações na versão mais recente do Helm-Chart
2. Irá instalar dois pods: controller e webhook
    1. Ambos controlam os submits de aplicações + gestão de recursos

Criar Imagem Docker PySpark + Delta

```
1. # Escolha uma imagem base oficial do Spark com suporte ao Hadoop e Python
FROM bitnami/spark:3.4.1

# Defina as variáveis de ambiente necessárias
ENV DELTA_CORE_VERSION=2.4.0 \
    JAVA_HOME=/opt/bitnami/java \
    PATH=$PATH:/opt/bitnami/java/bin

# Instale dependências de sistema e Python
USER root
RUN apt-get update && apt-get install -y --no-install-recommends \
    python3-pip \
    python3-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# Instale bibliotecas Python necessárias
# COPY requirements.txt /tmp/requirements.txt
RUN pip3 install --no-cache-dir delta-spark==${DELTA_CORE_VERSION}

# Copie os scripts para o contêiner
WORKDIR /opt/spark/work-dir
COPY notebooks/delta.py .
COPY data/dados_fake.csv .

# Altere permissões (necessário para execução com usuários não root)
RUN chmod -R 775 /opt/spark/work-dir

# Usuário padrão para execução
USER 1001
```

2. docker build -t nome-imagem -f Dockerfile.jobcontainer .
3. docker tag nome-imagem hub/reponame
4. docker push hub/reponame

Construir Manifesto para o Job Spark

Configs Principais

- sparkConf:
  - Delta Dependencies devem ser anexadas ao spark manifest, se não, independente de ter uma imagem com Delta, ela não sera executada
  - Isso se deve em conta da estrutura de execução dos jobs via spark-submit
- volumes:
  - É possivel criar um ConfigMap do seu script e cria-lo dentro do cluster e referenciar via mountpoint dentro do manifesto spark
- macros.datetime:
  - Não funciona para deploy local via kubectl
  - Somente com chamada via Airflow
- sparkVersion:
  - Deve ser igual a imagem do dockerfile
- dynamicAllocation: Alocação de replica de containers conforme demanda dos Executores
  - enabled: true
  - initialExecutors: 2
  - maxExecutors: 5
  - minExecutors: 1

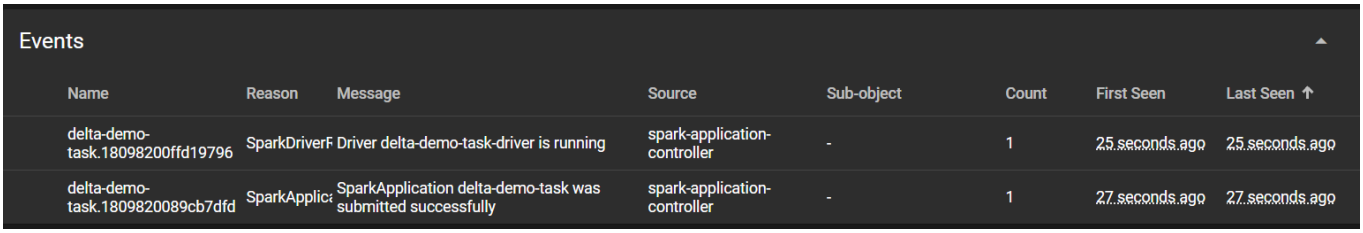
O Manifesto

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: delta-faker-app-{{ macros.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
}}
  namespace: default
spec:
  timeToLiveSeconds: 3600
  type: Python
  pythonVersion: "3"
  mode: cluster
  image: alexno9/spark-job-container:v01
  imagePullPolicy: IfNotPresent
```

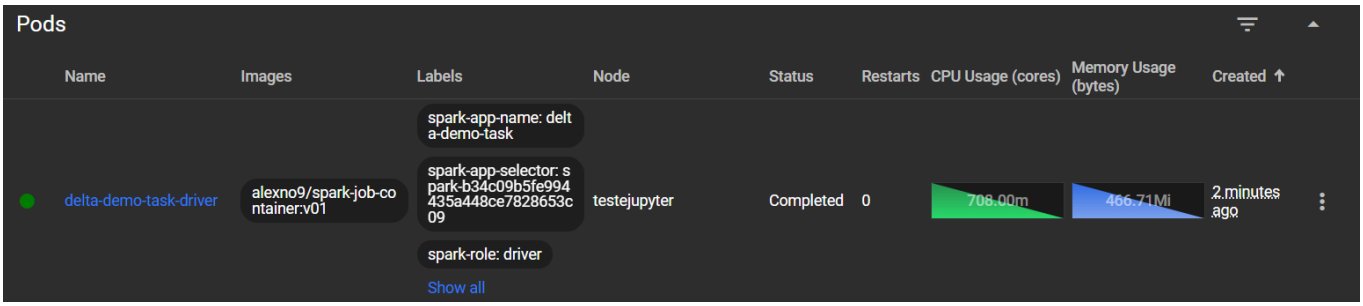
```
  mainApplicationFile: "local:///opt/spark/work-dir/delta.py"
  sparkVersion: 3.4.1
  sparkConf:
    "spark.jars.packages": "io.delta:delta-core_2.12:2.4.0"
    "spark.sql.extensions": "io.delta.sql.DeltaSparkSessionExtension"
    "spark.sql.catalog.spark_catalog":
"org.apache.spark.sql.delta.catalog.DeltaCatalog"
  restartPolicy:
    type: Never
  driver:
    cores: 1
    coreRequest: "0.5"
    coreLimit: 800m
    memory: 512m
    labels:
      version: 3.4.1
    serviceAccount: spark-operator-spark
  executor:
    instances: 1
    coreRequest: "1200m"
    coreLimit: 1500m
    memory: 512m
    labels:
      version: 3.4.1
```

Execução de Um Manifesto Via Spark Operator

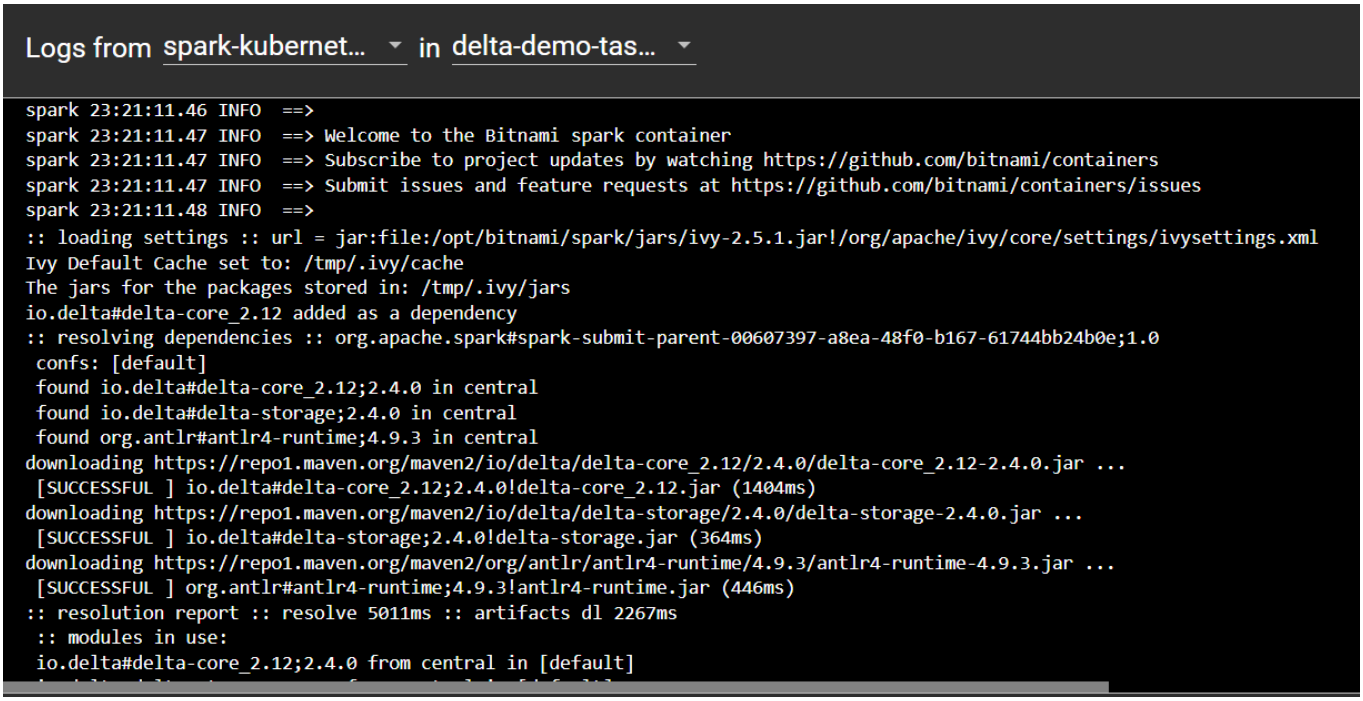
O CRD `spark-application` acompanha todas as requisições de aplicação. Quando executamos um apply no manifesto, é recebida uma requisição de aplicação contendo as instruções que passamos no yaml.



Um pod é criado dentro do namespace selecionado para o driver



Log de Execução da Aplicação



Demo Test com Sample spark-pi

- kubectl apply -f https://raw.githubusercontent.com/kubeflow/spark-operator/refs/heads/master/examples/spark-pi.yaml -n processing

```yaml
apiVersion: sparkoperator.k8s.io/v1beta2
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: spark:3.5.3
  imagePullPolicy: IfNotPresent
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: local:///opt/spark/examples/jars/spark-examples.jar
  arguments:
  - "5000"
  sparkVersion: 3.5.3
  driver:
    labels:
      version: 3.5.3
    cores: 1
    memory: 512m
    serviceAccount: spark-operator-spark
  executor:
```

```
  labels:
    version: 3.5.3
  instances: 1
  cores: 1
  memory: 512m
```