Spark su Docker e Kubernetes

Fonti

- how-to-deploy-a-jump-pod-on-kubernetes
- getting-started-with-spark-on-kubernetes
- kubedex.com
- spark-scheduling-in-kubernetes
- spark-on-k8s-operator
- how-to-manage-monitor-spark-on-kubernetes-introduction-spark-submit-kubernetes-operator

Per poter eseguire Kafka si deve deployare un cluster che si componga di:

- Nodo Master
- Nodi Worker

Quindi si utilizza uno script chiamato spark-submit per far eseguire delle applicazioni custom.

Per poterlo utilizzare all'interno del contesto di Kubernetes abbiamo due alternative

Modalità preferita: Kubernetes Operator For ApacheSpark

Modalità di utilizzo

Spark Master e Cluster configurati su un Pod Standalone - spark-submit attraverso un pod gestito manualmente

Versione richiesta di Spark: 2.3.0+

Prerequisiti

- Configurazione manuale di un cluster Spark su Kubernetes. L'idea è di configurare un immagine Docker in maniera tale che all'avvio si possano richiamare gli script bin/start-master e bin/start-worker. Il deployment dovrà deployare prima il master quindi N nodi worker.

Installazione

Una volta che su kubernetes sarà stato predisposto il cluster sarà sufficente lanciare da un'altra macchina il seguente comando

```
bin/spark-submit \
    --master spark://https://<master-ip>:7077 \
    --deploy-mode cluster \
    --conf spark.executor.instances=1 \
    --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark
\
    --conf
spark.kubernetes.container.image=trow.kube-public:31000/docfly-spark:lat
est \
    --class org.apache.spark.examples.SparkPi \
    --name spark-pi \
    local:///opt/spark/examples/jars/spark-examples_2.11-2.4.4.jar
```

Considerazioni

Pro

- Accesso perenne alla dashboard masterUI (mostra informazioni su risorse utilizzate, consumi, disco, worker attivi e log etcetc
- Permette un maggior controllo del numero di esecutori, del numero di worker e dei carichi di lavoro. Aumentando le repliche del pod dei worker è possibile concedere a spark una maggiore potenza.

Contro

- Configurazione più manuale
- Non sfruttiamo le API degli Operator di K8s

Spark Master e Cluster configurati su Kubernetes Cluster - spark-submit attraverso un pod gestito manualmente

Prerequisiti

- Versione richiesta di Spark: 2.3.0+
- Versione richiesta di Kubernetes: 1.4+
- Kubectl command line configurata per puntare il cluster Kubernetes che si vuole utilizzare

Installazione

Le ultime versione di Spark e Kubernetes vengono fornite di base di strumenti per poter deployare utilizzando direttamente tramite Operator di Kubernetes.

```
kubectl cluster-info // per recuperare kubernetes-ip e port
bin/spark-submit \
    --master k8s://https://<kubernetes-ip>:<kubernetes-port> \
    --deploy-mode cluster \
    --conf spark.executor.instances=1 \
    --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark
\
    --conf
spark.kubernetes.container.image=trow.kube-public:31000/docfly-spark:lat
est \
    --class org.apache.spark.examples.SparkPi \
    --name spark-pi \
    local:///opt/spark/examples/jars/spark-examples_2.11-2.4.4.jar
```

Considerazioni

Pro

- Sfruttiamo al massimo le funzionalità K8s
- Spark si occupa in maniera autonoma di gestire le risorse che deve utilizzare

Contro

- Perdiamo la dashboard sulla masterUI, ciascun `master`, chiamato DRIVER vive fin tanto che almeno un suo worker è in utilizzo.

Kubernetes Operator For Apache Spark - spark-submit attraverso lo SparkOperator

"The Kubernetes Operator for Apache Spark aims to make specifying and running Spark applications as easy and idiomatic as running other workloads on Kubernetes. It uses Kubernetes custom resources for specifying, running, and surfacing status of Spark applications."

La funzionalità degli Spark Operator non è nativa all'interno di K8s attualmente, dovrebbe diventarlo nei successivi rilasci di K8s v 1.16.

È però possibile installarlo a partire dalla versione 1.13 e rende molti dei passaggi precedenti completamente trasparenti.

Prerequisiti

- Helm (attualmente unico metodo testato)

Installazione

Installazione SparkOperator

```
<Installazione dipendente da sistema in cui stiamo lavorando,
l'operazione è stata testata su MacOs, Windows 10, Ubuntu Server
18.04.3>
helm init
helm repo add incubator
http://storage.googleapis.com/kubernetes-charts-incubator
helm install incubator/sparkoperator --namespace techitalia --set
enableWebhook=true --generate-name
```

Una volta installato lo sparkoperator, dall'incubator di google è sufficente applicare uno yaml

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: docfly-spark
  namespace: spark
spec:
  volumes:
    - name: parquet
      persistentVolumeClaim:
        claimName: parquet-pv-claim
  type: Java
  mode: cluster
  image: "localhost:5000/docfly-spark:latest"
  imagePullPolicy: Always
  mainClass:
it.arubapec.esecurity.mongostreamspark.SpringKafkaApplication
  mainApplicationFile:
"local:///opt/spark/examples/jars/app.jar/app.jar"
  sparkVersion: "2.4.4"
  restartPolicy:
    type: Never
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 2.4.4
    serviceAccount: spark
    volumeMounts:
      - name: parquet
        mountPath: "/mnt/parquet" # Deve coincidere con il configurato
all'interno dell'immagine docker!
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 2.4.4
```

Pro

- Risparmia un sacco di configurazioni, la gestione manuale dei driver e delle schedulazioni.

Contro

- Se si necessita di configurazioni particolari risulta più difficile applicarle, principalmente perché manca una vera documentazione.

Apache Zeppelin

https://zeppelin.apache.org/

Apache Zeppelin on K8S

- https://zeppelin.apache.org/docs/0.9.0-SNAPSHOT/quickstart/kubernetes.html

Requisiti per l'installazione

- Docker installato su pc
- Kubectl installato su pc
- Git installato su pc

Installazione

Non esiste attualmente una versione completamente stabile di Apache Zeppeling che possa runnare out of the box su kubernetes.
Le linee guida per poterlo utilizzare richiedono di eseguire una build di maven dei sorgenti e, partendo da questi, costruirsi la propria immagine Docker.
Inoltre il Dockerfile che hanno predisposto nelle sorgenti della versione master su github presenta un paio di imprecisazioni che rendono l'installazione non immediata.
L'immagine di docker finale raggiunge la dimensione di 5 GB e prove ripetute rischiano di saturare sia la repository docker su K8S che quella locale del pc.
È un problema noto che hanno deciso di risolvere successivamente nella loro prossima, ma non schedulata, release stabile.

Scaricare i sorgenti di Zeppelin e buildarli con Maven

```
git clone https://github.com/apache/zeppelin.git
cd zeppelin
mvn package -DskipTests -Pbuild-distr <your flags> // i flag dovrebbero permettere di specificare le versioni dei vari linguaggi e o librerie di cui si necessita, io ho eseguito la procedura digitando solamente mvn package -DskipTests -Pbuild-distr
```

L'operazione richiederà un bel pò di tempo, e genererà un jar di circa 1.4 GB.

NB: La build ha funzionato sia con Java 12 che con Java 8, ma non con Java 11. In teoria loro consigliano Java 7.

NB: Parte della build richiede l'installazione di node.js per la build della parte frontend. Su Windows ci sono stati dei problemi con i delimitatori di caratteri, non appena possibile il tar.gz buildato verrà caricato online.

Spostare il pacchetto nella contesto Docker e fixare i problemi nel dockerfile

```
mv zeppelin-distribution/target/zeppelin-*.tar.gz
scripts/docker/zeppelin/bin/
vi scripts/docker/zeppelin/bin/Dockerfile


...
# Cercare le seguenti righe verso la fine dello script e commentarle //
occhio a spostare a se stante la riga con il chown! Vedi sotto!
# RUN echo "$LOG_TAG Download Zeppelin binary" && \
#    wget -O /tmp/zeppelin-${Z_VERSION}-bin-all.tgz
http://archive.apache.org/dist/zeppelin/zeppelin-${Z_VERSION}/zeppelin-$
{Z_VERSION}-bin-all.tgz && \
#    tar -zxvf /tmp/zeppelin-${Z_VERSION}-bin-all.tgz && \
#    rm -rf /tmp/zeppelin-${Z_VERSION}-bin-all.tgz && \
#    mv /zeppelin-${Z_VERSION}-bin-all ${Z_HOME}

# E aggiungere le seguenti righe
RUN chown -R zeppelin:zeppelin ${Z_HOME}
ADD zeppelin-${Z_VERSION}.tar.gz /
RUN ln -s /zeppelin-${Z_VERSION} /zeppelin
...
```

NB: Nel Dockerfile è presente un altro bug, lo script crea la cartella `zeppelin` sotto la root / come home folder dell'utente `zeppelin`, quindi prova in queste ultime righe a ricrearla come link simbolico verso i sorgenti che abbiamo inserito nell'immagine. Questo crea un risultato in atteso che fa fallire la build. La cosa migliore da fare è cambiare tutti i riferimenti all'utente zeppelin rinominandolo in altro modo.

Buildare l'immagine di docker e pusharla nel proprio repository

```
cd scripts/docker/zeppelin/bin
docker build -t zeppelin-home:0.9.0-SNAPSHOT .            # ci vorrà un
bel pò di tempo, prt 1
docker tag zeppelin-home:0.0.9-SNAPSHOT
localhost:5000/zeppelin-home:0.9.0-SNAPSHOT
docker push localhost:5000/zeppelin-home:0.9.0-SNAPSHOT          # ci
vorrà un bel pò di tempo, prt 2
```

Deployare un POD su K8S

```
curl -s -O
https://raw.githubusercontent.com/apache/zeppelin/master/k8s/zeppelin-se
rver.yaml
vi zeppelin-server.yaml
# Cambiare le seguenti righe:
...
spec:
  automountServiceAccountToken: true
  containers:
  - name: zeppelin-server
    image: localhost:5000/zeppelin-home:0.9.0-SNAPSHOT   <-----
    command: ["sh", "-c", "$(ZEPPELIN_HOME)/bin/zeppelin.sh"]
    lifecycle:
      preStop:
        exec:
          # SIGTERM triggers a quick exit; gracefully terminate instead
          command: ["sh", "-c", "ps -ef | grep
org.apache.zeppelin.server.ZeppelinServer | grep -v grep | awk '{print
$2}' | xargs kill"]
    env:
    - name: ZEPPELIN_K8S_CONTAINER_IMAGE
      value: localhost:5000/zeppelin-home:0.9.0-SNAPSHOT   <-----
    - name: ZEPPELIN_HOME
      value: /zeppelin            <-----
 volumeMounts:
    - name: parquet
   mountPath: /zeppelin/k8s-custom
[...]
 volumes:
  - name: zeppelin-server-conf-volume
    configMap:
      name: zeppelin-server-conf
      items:
      - key: nginx.conf
        path: nginx.conf
      - key: serviceDomain
        path: serviceDomain
  - name: parquet            <-----
    persistentVolumeClaim:          <-----
      claimName: parquet-pv-claim        <-----
...


kubectl apply -f zeppelin-server.yaml
kubectl port-forward zeppelin-server 8080:80
kubectl delete -f zeppelin-server.yaml
```

A questo punto sarà possibile connettersi alla pagina web localhost:8080 per accedere al front end di zeppelin e poter eseguire i Job attraverso gli interpreti.

Se il master di Spark sarà all'interno di questo cluster K8S sarà raggiungibile di default, altrimenti andrà cambiata la seguente property nello zeppelin-server.yaml

```
[...]
    - name: MASTER   # default value of master property for spark
interpreter.
      value: k8s://https://kubernetes.default.svc
[...]
```

I volume mount sono stati aggiunti per poter mettere a fattor comune il filystem dove Spark elabora i dati!

~~Connettersi ad un POD spark pre esistente~~

Eseguire elaborazioni sui dati generati da un job SPARK

Zeppelin è un gestore di interpreti, non andrà mai a invocare un pod pre esistente, anzi, si interfaccerà direttamente con Kubernetes per creare nuovi nodi esecutori.

La questione principale è come dargli accesso ai dati elaborati in precedenza da un nodo Spark, ci sono due casistiche da prendere in esame:

Spark elabora i dati e li inserisce all'interno di un database (SQL/NoSQL)

DA TESTARE

Dato che all'interno di Kubernetes le rotte tra i pod sono generalmente sempre accessibili dovrebbe essere necessario creare da Zeppelin un Interpreter custom della famiglia SQL/NoSQL e mettere, al posto dell'ip, il nome del service corretto, con relativa porta.

Una volta creata però la configurazione non sopravviverebbe ad un eventuale riavvio del pod, per poter far permanere i notebook e gli interpreter è necessario configurare le seguenti proprietà all'interno del file zeppelin-server.yaml:

zeppelin-server.yaml

Spark elabora i dati e li inserisce all'interno di uno dei propri filesystem (hive/hdfs/parquet)

All'interno del sorgente di Zeppelin è presente un file `100-interpreter-spec.yaml`, al path /zeppelin-VERSION/k8s/interpreter.

Questo file di configurazione viene utilizzato da Zeppelin per invocare l'esecuzione di nuovi esecutori, necessita due modifiche, una per mettere a fattor comune il pvc dove vengono salvati i dati di elaborazione ed una per dare l'istruzione alla spark-submit di montare quel path nel nodo esecutore.

Nel dettaglio:

100-interpreter-spec.yaml - VolumeMounts

```
[...]
{% if zeppelin.k8s.interpreter.group.name == "spark" %}
    volumeMounts:
    - name: spark-home
      mountPath: /spark
    - name: zeppelin-server-custom-k8s  <---
      mountPath: "/zeppelin/k8s-custom"  <---
  initContainers:
  - name: spark-home-init
    image: {{zeppelin.k8s.spark.container.image}}
    command: ["sh", "-c", "cp -r /opt/spark/* /spark/"]
    volumeMounts:
    - name: spark-home
      mountPath: /spark
    - name: zeppelin-server-custom-k8s  <---
      mountPath: "/zeppelin/k8s-custom"  <---
  volumes:
  - name: spark-home
    emptyDir: {}
  - name: zeppelin-server-custom-k8s  <---
    persistentVolumeClaim:     <---
      claimName: parquet-pv-claim   <---
{% endif %}
[...]
```

100-interpreter-spec.yam - SparkSubmit

```
[...]
env:
  {% for key, value in zeppelin.k8s.envs.items() %}
  - name: {{key}}
 value: {{value}}
  {% endfor %}
  - name: SPARK_SUBMIT_OPTIONS                        <---
    value: "--conf
spark.kubernetes.executor.volumes.persistentVolumeClaim.parquet-pvc.moun
t.path=/zeppelin/k8s-custom"  <---
[...]
```

Mentre la prima modifica è vincolante, la seconda è una shortcut, nel caso in cui fossero necessarie più parametri e configurazioni è possibile configurare un file chiamato zeppelin-env.sh che si limita a fare l'export delle properties desiderate nelle variabili d'ambiente prima di lanciare l'esecuzione della submit.

In questo caso ci siamo limitati ad inserire un'unica configurazione all'interno della variabile d'ambiente "SPARK_SUBMIT_OPTIONS".

```
#

# Licensed to the Apache Software Foundation (ASF) under one or more

# contributor license agreements.  See the NOTICE file distributed with

# this work for additional information regarding copyright ownership.

# The ASF licenses this file to You under the Apache License, Version 2.0

# (the "License"); you may not use this file except in compliance with

```

```
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
apiVersion: v1
kind: ConfigMap
metadata:
name: zeppelin-server-conf
namespace: techitalia
data:
# 'serviceDomain' is a Domain name to use for accessing Zeppelin UI.
# Should point IP address of 'zeppelin-server' service.
#
# Wildcard subdomain need to be point the same IP address to access service
inside of Pod (such as SparkUI).
# i.e. if service domain is 'local.zeppelin-project.org', DNS configuration
should make 'local.zeppelin-project.org' and '*.local.zeppelin-project.org'
point the same address.
#
# Default value is 'local.zeppelin-project.org' while it points 127.0.0.1 and
`kubectl port-forward zeppelin-server` will give localhost to connects.
# If you have your ingress controller configured to connect to
`zeppelin-server` service and have a domain name for it (with wildcard
subdomain point the same address), you can replace serviceDomain field with
your own domain.
serviceDomain: local.zeppelin-project.org:8080
sparkContainerImage: luciferino/spark:2.4.4
nginx.conf: |
daemon off;
worker_processes auto;
events {
worker_connections 1024;
}
http {
map $http_upgrade $connection_upgrade {
```

```
default upgrade;
'' close;
}

# first server block will be default. Proxy zeppelin server.
server {
listen 80;
location / {
proxy_pass http://localhost:8080;
proxy_set_header Host $host;
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection $connection_upgrade;
proxy_redirect http://localhost $scheme://SERVICE_DOMAIN;
}
}

# match request domain [port]-[service].[serviceDomain]
# proxy extra service such as spark-ui
server {
listen 80;
server_name "~(?<svc_port>[0-9]+)-(?<svc_name>[^.]*)\.(.*)";
location / {
resolver 127.0.0.1:53 ipv6=off;
proxy_pass http://$svc_name.NAMESPACE.svc:$svc_port;
proxy_set_header Host $host;
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection $connection_upgrade;
proxy_redirect http://localhost $scheme://SERVICE_DOMAIN;

# redirect rule for spark ui. 302 redirect response misses port number of
service domain
proxy_redirect ~(http:[/]+[0-9]+[-][^-]+[-][^.]+)[^/]+(\/jobs.*)
$1.SERVICE_DOMAIN$2;
}
}
}
---
```

```
kind: Pod
apiVersion: v1
metadata:
name: zeppelin-server
namespace: techitalia
labels:
app: zeppelin-server
spec:
automountServiceAccountToken: true
containers:
- name: zeppelin-server
image: luciferino/zeppelin:0.9.0-SNAPSHOT
command: ["sh", "-c", "$(ZEPPELIN_HOME)/bin/zeppelin.sh"]
lifecycle:
preStop:
exec:
# SIGTERM triggers a quick exit; gracefully terminate instead
command: ["sh", "-c", "ps -ef | grep org.apache.zeppelin.server.ZeppelinServer
| grep -v grep | awk '{print $2}' | xargs kill"]
env:
- name: ZEPPELIN_K8S_CONTAINER_IMAGE
value: luciferino/zeppelin:0.9.0-SNAPSHOT
- name: ZEPPELIN_HOME
value: /zeppelin
- name: ZEPPELIN_SERVER_RPC_PORTRANGE
value: 12320:12320
- name: POD_UID
valueFrom:
fieldRef:
apiVersion: v1
fieldPath: metadata.uid
- name: POD_NAME
valueFrom:
fieldRef:
apiVersion: v1
fieldPath: metadata.name
- name: ZEPPELIN_K8S_SPARK_CONTAINER_IMAGE
valueFrom:
configMapKeyRef:
```

```
name: zeppelin-server-conf
key: sparkContainerImage
- name: SERVICE_DOMAIN
valueFrom:
configMapKeyRef:
name: zeppelin-server-conf
key: serviceDomain
- name: MASTER   # default value of master property for spark interpreter.
value: k8s://https://kubernetes.default.svc
volumeMounts:
- name: parquet
mountPath: "/zeppelin/k8s-custom"
# volumeMounts:
#  - name: zeppelin-server-notebook-volume     # configure this to persist
notebook
#    mountPath: /zeppelin/notebook
#  - name: zeppelin-server-conf                # configure this to persist
Zeppelin configuration
#    mountPath: /zeppelin/conf
#  - name: zeppelin-server-custom-k8s          # configure this to mount
customized Kubernetes spec for interpreter
#    mountPath: /zeppelin/k8s
- name: zeppelin-server-gateway
image: nginx:1.14.0
command: ["/bin/sh", "-c"]
args:
- cp -f /tmp/conf/nginx.conf /etc/nginx/nginx.conf;
sed -i -e "s/SERVICE_DOMAIN/$(cat /tmp/conf/serviceDomain)/g"
/etc/nginx/nginx.conf;
sed -i -e "s/NAMESPACE/$(cat
/var/run/secrets/kubernetes.io/serviceaccount/namespace)/g"
/etc/nginx/nginx.conf;
cat /etc/nginx/nginx.conf;
/usr/sbin/nginx
volumeMounts:
- name: zeppelin-server-conf-volume
mountPath: /tmp/conf
lifecycle:
preStop:
exec:
# SIGTERM triggers a quick exit; gracefully terminate instead
```

```
command: ["/usr/sbin/nginx", "-s", "quit"]

- name: dnsmasq  # nginx requires dns resolver for dynamic dns resolution
image: "janeczku/go-dnsmasq:release-1.0.5"
args:
- --listen
- "127.0.0.1:53"
- --default-resolver
- --append-search-domains
- --hostsfile=/etc/hosts
- --verbose
volumes:
- name: zeppelin-server-conf-volume
configMap:
name: zeppelin-server-conf
items:
- key: nginx.conf
path: nginx.conf
- key: serviceDomain
path: serviceDomain
- name: parquet
persistentVolumeClaim:
claimName: parquet-pv-claim
---
kind: Service
apiVersion: v1
metadata:
name: zeppelin-server
namespace: techitalia
spec:
ports:
- name: http
protocol: TCP
port: 80
targetPort: 8080
- name: rpc
protocol: TCP
port: 12320
targetPort: 12320
```

```
  selector:
    app: zeppelin-server
  type: LoadBalancer
  sessionAffinity: None
  externalTrafficPolicy: Cluster
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: zeppelin-server-role
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["create", "get", "update", "patch", "list", "delete", "watch"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["roles", "rolebindings"]
  verbs: ["bind", "create", "get", "update", "patch", "list", "delete", "watch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: zeppelin-server-role-binding
subjects:
- kind: ServiceAccount
  name: techitalia
roleRef:
  kind: Role
  name: zeppelin-server-role
  apiGroup: rbac.authorization.k8s.io
```