

spark on k8s

一. 安全验证

二. 环境准备

硬性需求:

1. spark 版本要求2.3.x 及以上
2. k8s 集群 1.6 版本及以上
3. 当前用户有 list/create/edit/delete pods的权限

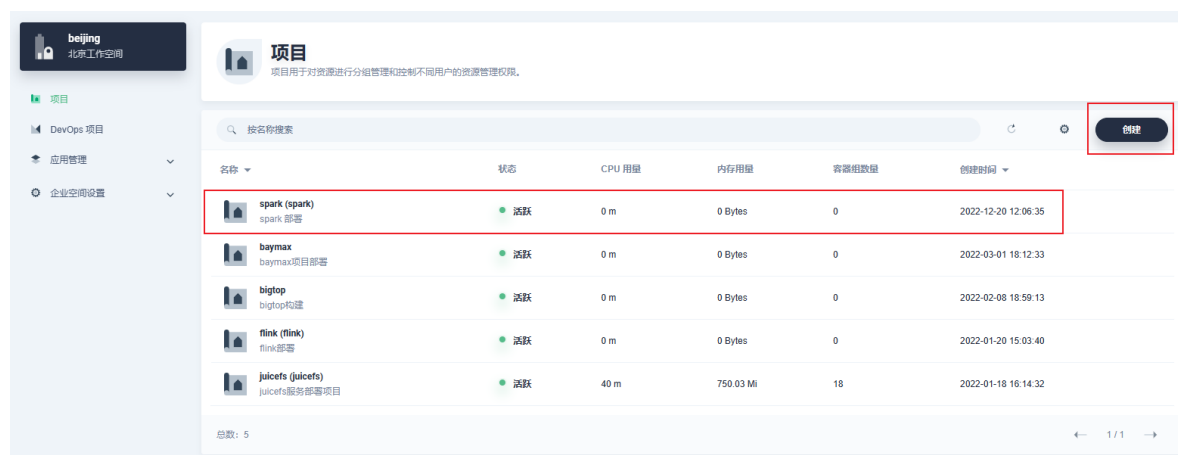
(1) `kubectl auth can-i <list|create|edit|delete> pods`

(2) driver 所在pods 可以创建pods/services/configmaps

4. 集群中必须配置k8s DNS

环境配置

1. 创建命名空间

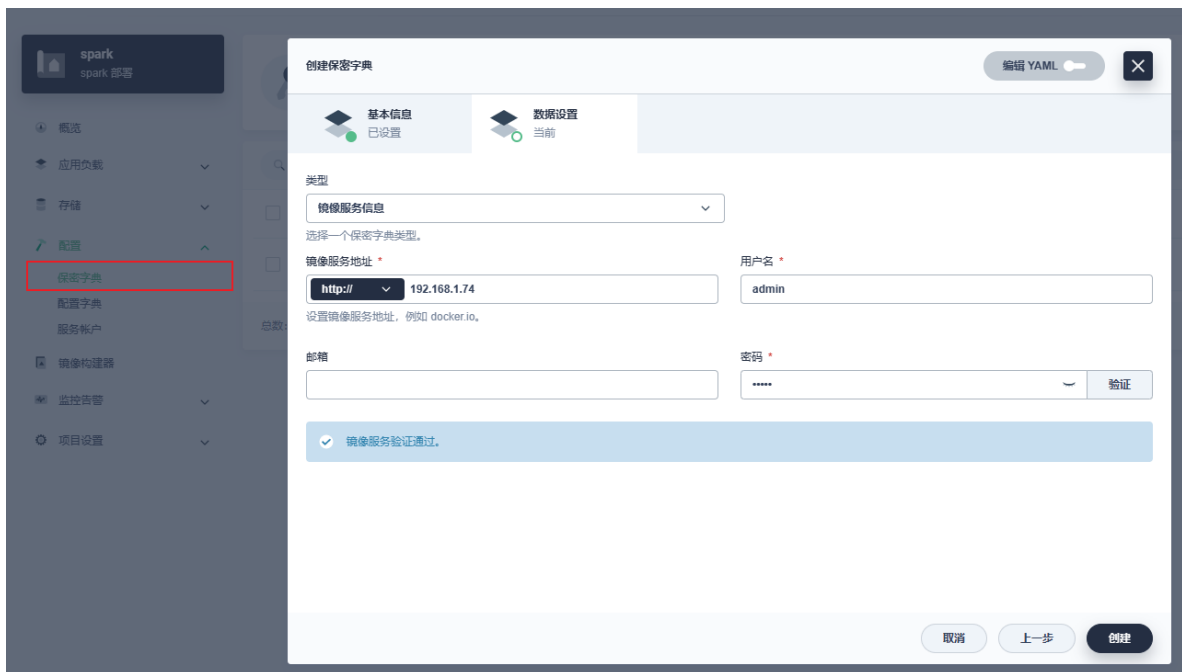


The screenshot shows the Kubernetes dashboard interface. On the left, there is a sidebar with navigation options: '项目' (Project), 'DevOps 项目', '应用管理' (Application Management), and '企业空间设置' (Enterprise Space Settings). The main area displays a table of namespaces. The first row, 'spark (spark)', is highlighted with a red box. The table columns are: 名称 (Name), 状态 (Status), CPU 用量 (CPU Usage), 内存用量 (Memory Usage), 容器组数量 (Number of Pods), and 创建时间 (Creation Time). The 'spark (spark)' namespace is listed with a status of '活跃' (Active), 0 m CPU usage, 0 Bytes memory usage, 0 pods, and a creation time of 2022-12-20 12:06:35. Other namespaces listed include baymax, bigtop, flink, and juicifs. A '创建' (Create) button is visible in the top right corner of the table area.

名称	状态	CPU 用量	内存用量	容器组数量	创建时间
spark (spark)	活跃	0 m	0 Bytes	0	2022-12-20 12:06:35
baymax	活跃	0 m	0 Bytes	0	2022-03-01 18:12:33
bigtop	活跃	0 m	0 Bytes	0	2022-02-08 18:59:13
flink (flink)	活跃	0 m	0 Bytes	0	2022-01-20 15:03:40
juicifs (juicifs)	活跃	40 m	750.03 Mi	18	2022-01-18 16:14:32

2. 创建保密字典

访问Harbor的认证



3. 创建服务账户

spark driver将通过该服务账户访问 k8s api, 创建监视 executor pod. 因此该账户必须具有管理资源的权限, 以便driver 能够正常执行. 默认情况下, 如果创建pod时未指定则使用default账户.

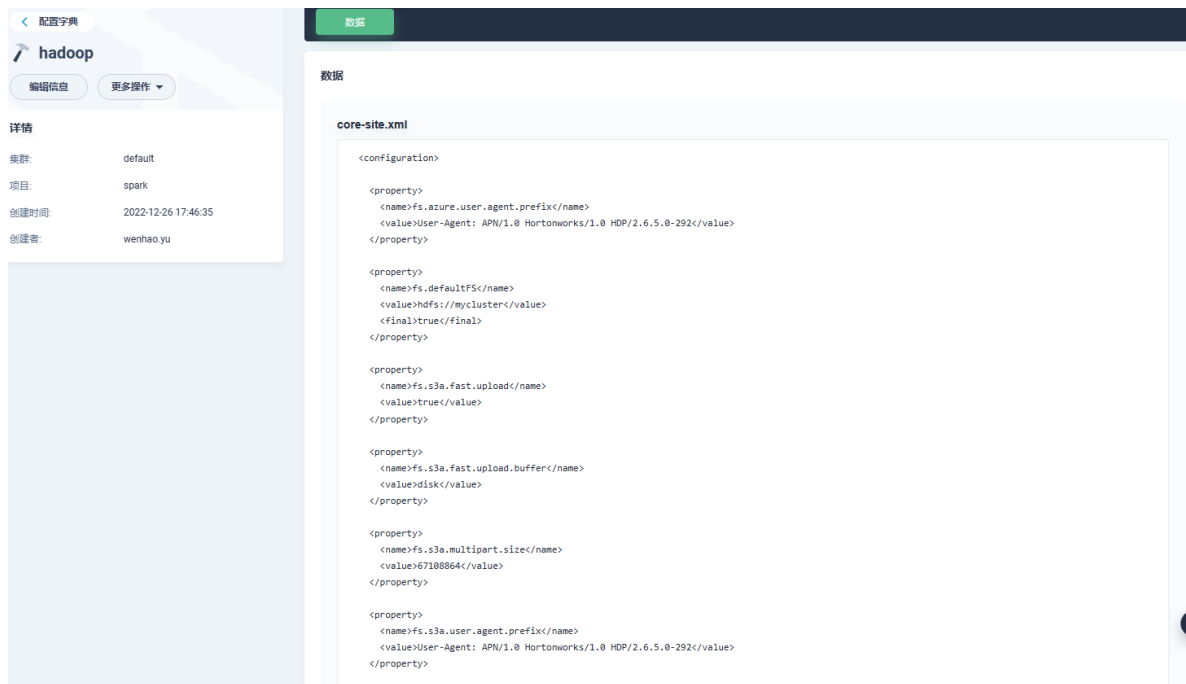


4. 创建配置字典

应用程序可以通过加载配置字典, 来获取相关信息.

如下图所示, 创建了名为 hadoop 的配置字典, key值 core-site.xml ; value值 文件所有参数.

该配置字典可以解决 pod 内解析 hadoop 域名的问题.



5. 创建PV

(1) 通过 kubesphere UI 创建

(2) 通过 yaml 文件创建

创建前我们先在master节点 mkdir /nfs/spark;

创建一个spark子目录供pv使用

vim spark-pv.yaml

```
1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: spark-pv
5    namespace: spark
6    labels:
7      pv: spark-pv
8  spec:
9    capacity:
10     storage: 5Gi
11    accessModes:
12     - ReadWriteMany
13    persistentVolumeReclaimPolicy: Retain
14    storageClassName: nfs-storage
15    nfs:
16     server: 192.168.1.149
17     path: "/nfs/spark" #NFS目录，需要该目录在NFS上存在
```

然后执行 kubectl apply -f spark-pv.yaml

```
spark
spark-pv.yaml
spc-test01-spc-spark-cl-pvc-spc-spark01-0-pvc-f52228d0-d806-419a-bd31-1e22821e954e
test0124-volum-pvc-f082e6aa-bdf1-44ce-8bb8-07462d4c1953
zookeeper
[root@master nfs]# kubectl apply -f spark-pv.yaml
persistentvolume/spark-pv created
[root@master nfs]#
```

```
[root@master nfs]# kubectl get pv spark-pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM          STORAGECLASS  REASON  AGE
spark-pv      5Gi       RWX           Retain          Available  spark/spark-300-pvc  nfs-storage              15s
[root@master nfs]#
```

PV 的访问模式 (accessModes) 有三种:

ReadWriteOnce (RWO) : 是最基本的方式, 可读可写, 但只支持被单个 Pod 挂载。

ReadOnlyMany (ROX) : 可以以只读的方式被多个 Pod 挂载。

ReadWriteMany (RWX) : 这种存储可以以读写的方式被多个 Pod 共享。

PV 的回收策略 (persistentVolumeReclaimPolicy, 即 PVC 释放卷的时候 PV 该如何操作) 也有三种:

Retain, 不清理, 保留 Volume (需要手动清理)

Recycle, 删除数据, 即 `rm -rf /volume/*` (只有 NFS 和 HostPath 支持)

Delete, 删除存储资源, 比如删除 AWS EBS 卷 (只有 AWS EBS, GCE PD, Azure Disk 和 Cinder 支持)

PVC 释放卷是指用户删除一个 PVC 对象时, 那么与该 PVC 对象绑定的 PV 就会被释放。

```
[root@master conf]# kubectl get pv pvc-2f855911-bb68-4bc1-86d6-218c0c16ff8c
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM          STORAGECLASS  REASON  AGE
pvc-2f855911-bb68-4bc1-86d6-218c0c16ff8c  5Gi       RWO,ROX,RWX   Delete          Bound     spark/spark-300-pvc  nfs-storage              5h48m
[root@master conf]# kubectl get pv pvc-2f855911-bb68-4bc1-86d6-218c0c16ff8c
Error from server (NotFound): persistentvolumes "pvc-2f855911-bb68-4bc1-86d6-218c0c16ff8c" not found
[root@master conf]#
```

PersistentVolume 有四种状态:

Available: 可用状态

Bound: 绑定到 PVC

Released: PVC 被删掉, 但是尚未回收


Failed : 自动回收失败

6. 创建 pvc

`vim spark-pvc.yaml`

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: spark-pvc
5    namespace: spark
6  spec:
7    accessModes:
8      - ReadWriteMany
9    resources:
10     requests:
11       storage: 500Mi #容量
12   selector:
13     matchLabels:
14       pv: spark-pv #关联pv 的label,key/value要一致
```

执行 `kubectl apply -f spark-pvc.yaml`

<input type="checkbox"/>	名称	状态	持久卷	访问模式	挂载状态	创建时间	
<input type="checkbox"/>	 spark-pvc nfs-storage	● 已绑定	spark-pv	RWX	已挂载	2022-12-20 14:49	⋮
总数: 1							← 1 / 1 →

7. 在挂载目录下创建子目录

```
[root@master spark]# ll
总用量 24
drwxr-xr-x 2 root root  91 1月 10 15:25 extraConf
drwxr-xr-x 4 root root  27 1月  4 20:30 hadoop
drwxr-xr-x 2 root root 12288 1月  9 19:23 jars
drwxr-xr-x 2 root root 4096 1月 10 14:22 job
drwxrwxrwx 2 root root  154 1月 10 15:57 spark-logs
drwxr-xr-x 2 root root 4096 1月  5 15:27 tmp
```

extraConf/: 配置文件

hadoop/: 存放 hadoop 相关依赖文件

jars/: 存放 spark jar, 平台 jar 包

job/: 存放job相关依赖.

spark-logs/: 日志

8. 下载 spark.tgz 包到服务器并解压

<https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.7.tgz>

使用 spark-3.x 来配置, 因为spark-2.x 缺少很多设置, 相当不灵活, 有些配置只能在镜像初始化时添加进去. spark-3.x之后支持了Pod templates. 可以在模板文件中设置相关参数. 并且支持driver, executor 分别指定不同的模板文件.

9. 构建 docker 镜像

需要在 k8s 先部署一个spark. spark从2.3.x开始自带Dockerfile. 一般放在 kubernetes/dockerfiles 下. 而且bin 目录下自带 docker-image-tool.sh 脚本用来 build 跟 push.

```
1 $ ./bin/docker-image-tool.sh -r <repo> -t my-tag build
2 $ ./bin/docker-image-tool.sh -r <repo> -t my-tag push
```

```
1 build      Build image. Requires a repository address to be provided if
              the image will be pushed to a different registry.
2 push      Push a pre-built image to a registry. Requires a repository
              address to be provided.
3 -f file    Dockerfile to build for JVM based Jobs. By default builds the
              Dockerfile shipped with Spark.
4 -p file    Dockerfile to build for PySpark Jobs. Builds Python
              dependencies and ships with Spark.
5 -R file    Dockerfile to build for SparkR Jobs. Builds R dependencies and
              ships with Spark.
6 -r repo    Repository address.
7 -t tag     Tag to apply to the built image, or to identify the image to be
              pushed.
8 -m        Use minikube's Docker daemon.
9 -n        Build docker image with --no-cache
10 -b arg     Build arg to build or push the image. For multiple build args,
              this option needs to be used separately for each build arg.
```

制作镜像

```
1 cd /opt/spark
2 ./bin/docker-image-tool.sh -r 192.168.1.74 -t spark-v3.0.0 build
```

等上边命令运行完成后，查看本地生成的镜像

```
1 [root@info spark]# docker images |grep spark
2 192.168.1.74/spark          spark-v3.0.0          98ca52eee4d4   8 hours ago
   501MB
```

给新生成的192.168.1.74/spark镜像重新打上tag和版本号

```
1 [root@info spark]# docker tag 192.168.1.74/spark:spark-v3.0.0
  192.168.1.74/spark/spark3:v3.0.0
2 [root@info spark]# docker images |grep 192.168.1.74/spark/spark3
3 192.168.1.74/spark/spark3   v3.0.0   98ca52eee4d4   10 hours ago   501MB
```

将打好tag的镜像推送到Harbor仓库的spark下：

```
1 docker push 192.168.1.74/spark/spark3:v3.0.0
```

10. 配置本地spark

(1) 将 core-site.xml , hdfs-site.xml 放到 conf/ 下

(2) 创建日志存放目录

hdfs dfs -mkdir /spark3.0-history

(3) 修改 spark-default.conf

```
1 # log
2 spark.eventLog.enabled=true
3 spark.history.fs.logDirectory=hdfs://mycluster/spark3.0-history      #spark
   history 监控
4 spark.eventLog.dir=hdfs://mycluster/spark3.0-history      #event log 记录位置
5 spark.eventLog.compress=false                               # 是否压缩
6
7 # k8s
8 spark.kubernetes.namespace=spark
9 spark.kubernetes.authenticate.driver.serviceAccountName=spark-ywh
10 spark.kubernetes.file.upload.path=hdfs://mycluster/spark3      #依赖文件上传路径
11 spark.kubernetes.hadoop.configMapName=hadoop                  #解决hadoop集群映
   射
12 spark.kubernetes.driver.podTemplateFile=/nfs/spark/job/driver-pod-
   template.yaml
13 spark.kubernetes.executor.podTemplateFile=/nfs/spark/job/executor-pod-
   template.yaml
```

(5) 修改 log4j.properties

```
1 log4j.rootCategory=DEBUG, console, logfile
2 log4j.appender.console=org.apache.log4j.ConsoleAppender
3 log4j.appender.console.target=System.err
4 log4j.appender.console.layout=org.apache.log4j.PatternLayout
5 log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p
   %c{1}: %m%n
6
7 #set everything to be logged to the logfile
8 log4j.appender.logfile=org.apache.log4j.RollingFileAppender
```

```

9 log4j.appender.logfile.Encoding=UTF-8
10 log4j.appender.logfile.File=/nfs/spark/spark-logs/spark-flow.log
11 log4j.appender.logfile.MaxFileSize=10MB
12 log4j.appender.logfile.MaxBackupIndex=3
13 log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
14 log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %F
    %p %m%n

```

11. 创建 podTemplateFile

创建 pod 时会以此为模板. 会被提交命令, 或代码中的设置覆盖.

```

1 metadata:
2   name: driver-pod-template
3   namespace: spark
4 spec:
5   volumes:
6     - name: spark-driver-1
7       persistentVolumeClaim:
8         claimName: spark-pvc
9   containers:
10    - name: spark-kubernetes-driver
11      env:
12        - name: HADOOP_USER_NAME
13          value: merce
14      volumeMounts:
15        - name: spark-driver-1
16          mountPath: /opt/spark/jars
17          subPath: jars
18        - name: spark-driver-1
19          mountPath: /opt/spark/spark-logs
20          subPath: spark-logs
21        - name: spark-driver-1
22          mountPath: /opt/spark/extraConf
23          subPath: extraConf
24   hostAliases:
25     - ip: "192.168.1.81"
26       hostnames:
27         - "info1"
28     - ip: "192.168.1.82"
29       hostnames:
30         - "info2"
31     - ip: "192.168.1.83"
32       hostnames:
33         - "info3"
34     - ip: "192.168.1.84"
35       hostnames:
36         - "info4"
37     - ip: "192.168.1.85"
38       hostnames:
39         - "info5"

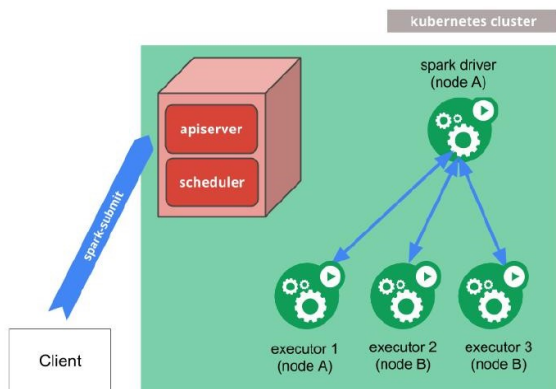
```

12. 在 Job 目录中放入测试jar包

```
[root@master job]# ls
driver-pod-template.yaml    flow_1670482638336_0016.json  woven-dataflow-spark-1.2.4.jar
executor-pod-template.yaml  flow_1670482638336_0016.res
flow_1670482638336_0016.conf  spark-examples_2.12-3.0.0.jar
```

三. 运行机制

spark 可以通过使用 spark-submit 直接提交一个spark 应用到 k8s集群. 运行流程如图:



- Spark 2.3加入了K8s原生的operator
- Spark应用以定制化K8s Controller的形式在K8s中运行
- spark-submit向K8s API Server发送请求, 创建Spark Driver Pod
- Spark Driver Pod创建Executor Pod并进行调度
- 程序结束后Executor Pod被清理回收
- Driver Pod进入COMPLETED状态

1. 首先 k8s 会先拉起一个driver pod
2. 然后driver pod会创建 executor pods 并与它们保持连接, 并执行应用代码.
3. 最后任务结束后所有executor pods 会被终止并清除. driver pod会保留日志并被标记完成状态直到被 GC 回收或清理.

注: 被标记 完成 态的 driver不会占用任何计算和资源.

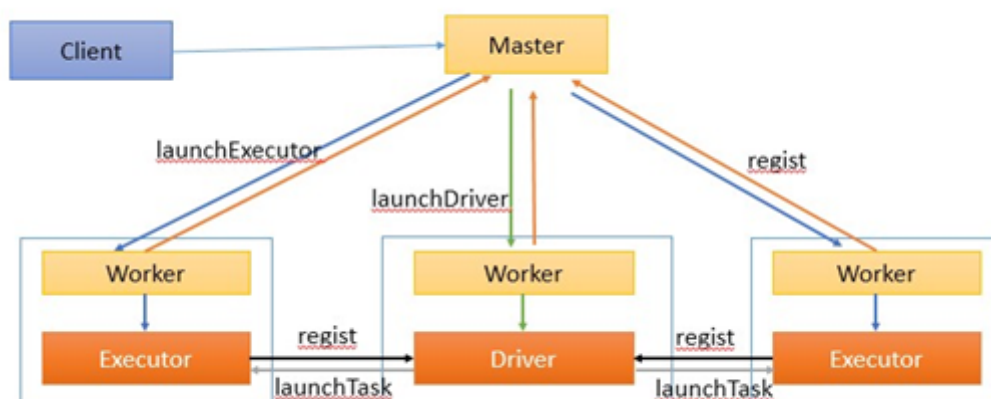
driver 跟 executor pod 的调度由k8s 来控制. 我们可以通过 node selector 的配置, 来选择合适的节点去调度driver跟 executor.

四. 提交应用

7. 测试案例

1. Cluster Mode

(1) 工作流程图



(2) 提交命令


```

1 bin/spark-submit \
2 --master k8s://https://1b.kubesphere.local:6443 \
3 --deploy-mode cluster \
4 --name spark-pi \
5 --class org.apache.spark.examples.SparkPi \
6 --conf spark.executor.instances=1 \           #executor 的实例数
7 --conf spark.executor.request.cores=1 \       #executor 的核数
8 --conf spark.executor.request.memory=100M \   #executor 的内存数
9 --conf spark.kubernetes.namespace=spark \     #命名空间
10 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-ywh \
    #服务账户名称
11 --conf spark.kubernetes.container.image=192.168.1.74/spark/spark3:v3.0.0 \
    #镜像地址
12 --conf spark.kubernetes.container.image.pullSecrets=spark-secret \
    #保密字典
13 --conf spark.kubernetes.driver.pod.name=ywhdriver \
    # pods
14 --conf spark.kubernetes.driver.volumes.persistentVolumeClaim.spark-driver-
    1.mount.path=/spark_driver \                #driver pod 容器内目录 -- 映射的是pvc绑
    定的pv目录
15 --conf spark.kubernetes.driver.volumes.persistentVolumeClaim.spark-driver-
    1.options.claimName=spark-pvc \            #声明 driver 绑定的 pvc
16 --conf spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-
    executor-1.mount.path=/spark_executor \    #executor pod容器内目录 -- 映
    射的是pvc绑定的pv目录
17 --conf spark.kubernetes.executor.volumes.persistentVolumeClaim.spark-
    executor-1.options.claimName=spark-pvc \   #声明 executor 绑定的 pvc
18 /spark_driver/spark-examples_2.11-2.4.5.jar # 容器目录

```

--name: 应用程序名称, 该名称用于命名 k8s 资源如 driver/executor 等. 名称必须由小写字母数字字符、-和组成。并且必须以字母数字字符开头和结尾。

k8s 集群建立之后可以通过执行命令 `kubectl cluster-info`, 来调出 apiserver URL.

```

1 $ kubectl cluster-info
2 Kubernetes master is running at https://1b.kubesphere.local:6443

```

此外也可以使用 `kubectl proxy` 来与 k8s api 通信.

启动方式:

```

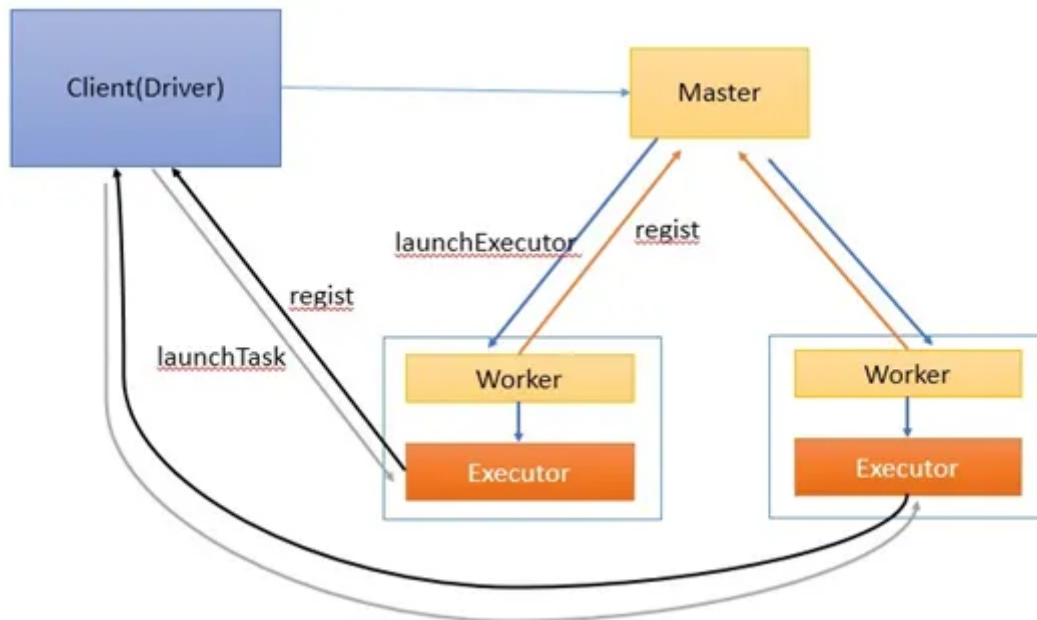
1 $ kubectl proxy

```

如果代理URL为 localhost:8081则只需将参数设置为 --master k8s://<http://127.0.0.1:8001>

2. Client Mode

(1) 工作流程图



(2) 网络

Spark executors 必须能够通过配置的路由的主机名和端口连接到Spark driver。Spark在 Client 模式下工作所需的具体网络配置因设置而异。如果在Kubernetes pod中运行 driver，可以使用一个 headless service来允许的 driver pod 通过一个主机名被访问到。当部署 headless 时，确保 service 的标签选择器只匹配driver，而不匹配其他 pod；建议为您的driver pod分配一个足够唯一的标签，并在headless service 的标签选择器中使用该标签。通过spark.driver.host指定driver的主机名，并将spark driver的端口指定为spark.driver.port。

(3) 垃圾回收

如果您在pod中运行Spark driver，强烈建议将Spark.kubernetes.driver.pod.name设置为该pod的名称。当设置此属性时，Spark scheduler将使用OwnerReference部署executor pod，这反过来将确保一旦driver pod从集群中删除，所有应用程序的executor pod也将被删除。driver 将在spark.kubernetes.namespace指定的命名空间中查找具有给定名称的pod，并且指向该pod的OwnerReference将被添加到每个executor pod的OwnerReferences列表中。注意避免将OwnerReference设置为实际上不是该driver pod的一个pod，否则当删除错误的pod时，driver可能会提前终止。

如果您的应用程序没有在pod中运行，或者当您的应用程序实际在pod中运行时，没有设置spark.kubernetes.driver.pod.name，请记住，当应用程序退出时，executor pod可能无法正确地删除。Spark调度器尝试删除这些pod，但如果网络请求API服务器由于任何原因失败，这些pod将保留在集群中。executor 在无法到达driver时应该退出，因此在应用程序退出后，executor pod不应该消耗集群中的计算资源(cpu和内存)。

(4) 提交命令

```

1 ./bin/spark-submit \
2 --master k8s://https://1b.kubesphere.local:6443 \
3 --deploy-mode client \
4 --name spark-pi \
5 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
6 --conf spark.kubernetes.container.image=192.168.1.74/spark/spark:v2.4.5 \
7 --conf spark.driver.host=192.168.1.149 \           # driver route
8 --conf spark.driver.port=8040 \
9 --conf spark.executor.instances=1 \
10 --conf spark.executor.request.cores=1 \
11 --conf spark.executor.request.memory=100M \

```

```
12 --conf spark.kubernetes.namespace=spark \  
13 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-ywh \  
14 --conf spark.kubernetes.container.image=192.168.1.74/spark/spark:v2.4.5 \  
15 --conf spark.kubernetes.container.image.pullSecrets=spark-secret \  
16 --class org.apache.spark.examples.JavaSparkPi \  
17 /nfs/spark/spark-examples_2.11-2.4.5.jar
```

五. 调试日志

日志可以通过Kubernetes API和kubectl CLI访问。当Spark应用程序正在运行时，可以使用以下方法从应用程序中传输日志：

```
1 | $ kubectl -n=<namespace> logs -f <driver-pod-name>
```

可以使用kubectl端口转发在本地访问与任何应用程序关联的UI。

```
1 | $ kubectl port-forward <driver-pod-name> 4040:4040
```

可能有几种类型的失败。如果Kubernetes API服务器拒绝spark-submit发出的请求，或者由于其他原因拒绝连接，则提交逻辑应该指出遇到的错误。但是，如果在应用程序运行期间出现错误，通常，最好的调查方法可能是通过Kubernetes CLI。

要获得有关driver pod调度的一些基本信息，可以运行：

```
1 | $ kubectl describe pod <spark-driver-pod>
```

如果pod遇到运行时错误，可以使用以下方法进一步探测状态：

```
1 | $ kubectl logs <spark-driver-pod>
```

失败的executor pod的状态和日志可以用类似的方式进行检查。最后，删除driver pod将清理整个spark应用程序，包括所有的executor，相关的service等。driver pod可以看作是Spark应用程序的Kubernetes表示。

1. 日志配置

在\${spark_home}/conf/spark-default.conf 中添加几项日志参数

(1) 事件日志需要配置 spark-default.conf

```
spark.eventLog.enabled=true  
spark.yarn.historyServer.address=http://192.168.1.83:18080  
spark.yarn.historyServer.allowTracking=true  
spark.history.fs.logDirectory=hdfs://192.168.1.81:8020/spark2.4-history  
spark.eventLog.dir=hdfs://192.168.1.81:8020/spark2.4-history  
spark.eventLog.compress=false  
spark.hadoop.yarn.timeline-service.enabled=false
```

(2) 客户端日志需要配置 本地log4j.properties

```
log4j.rootCategory=DEBUG, console, logfile  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.console.target=System.err  
log4j.appender.console.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
```

```
#set everything to be logged to the logfile
```

```
log4j.appender.logfile=org.apache.log4j.RollingFileAppender
```

```
log4j.appender.logfile.Encoding=UTF-8
```

```
log4j.appender.logfile.File=/nfs/spark/spark-logs/spark-flow.log
```

```
log4j.appender.logfile.MaxFileSize=10MB
```

```
log4j.appender.logfile.MaxBackupIndex=3
```

```
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %F %p %m%n
```

(3) 容器内日志需要在挂载目录 extraConf/ 下配置 log4j-driver.properties 跟 log4j-executor.properties

```
log4j.rootCategory=DEBUG, console, logfile
```

```
log4j.appender.console=org.apache.log4j.ConsoleAppender
```

```
log4j.appender.console.target=System.err
```

```
log4j.appender.console.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
```

```
#set everything to be logged to the logfile
```

```
log4j.appender.logfile=org.apache.log4j.RollingFileAppender
```

```
log4j.appender.logfile.Encoding=UTF-8
```

```
log4j.appender.logfile.File=/opt/spark/spark-logs/driver-spark-flow.log
```

```
log4j.appender.logfile.MaxFileSize=10MB
```

```
log4j.appender.logfile.MaxBackupIndex=3
```

```
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %F %p %m%n
```

注:

① extraConf/ 目录需要 +w 权限;

② 命令增加

```
1 --driver-java-options "-Dlog4j.configuration=file:/opt/spark/extraConf/log4j-
  driver.properties" \
2 --conf spark.executor.extraJavaOptions="-
  Dlog4j.configuration=file:/opt/spark/extraConf/log4j-executor.properties" \
```

(4) 保存不同任务容器内的日志

① 升级log4j -> log4j2,

导入log4j-1.2-api-2.12.1.jar, log4j-api-2.12.1.jar, log4j-core-2.12.1.jar, log4j-slf4j-impl-2.12.1.jar

② 在/nfs/spark/extraConf 下添加log4j2.xml,

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration status="WARN" monitorInterval="30">
3   <Properties>
4     <Property name="logName"><![CDATA[${sys:logName}]]></Property>
5     <Property name="logDir"><![CDATA[${sys:logDir}]]></Property>
6     <Property name="LOG_FILE"><![CDATA[${logDir}/${logName}.log]]>
7   </Property>
8     <Property name="LOG_LEVEL_PATTERN"><![CDATA[%5p]]></Property>
9     <Property name="LOG_PATTERN">
```

```

9      <![CDATA[%d{yyyy-MM-dd HH:mm:ss.SSS}] %X{pid} %5p [%t] ---
%{c{1}}: %m%n %throwable{1000}]]></Property>
10    </Properties>
11    <appenders>
12      <console name="Console" target="SYSTEM_OUT">
13        <PatternLayout pattern="${LOG_PATTERN}"/>
14      </console>
15      <RollingFile name="RollingFileInfo" fileName="${LOG_FILE}"
16        filePattern="${logDir}/${date:yyyy-MM}/info-%d{yyyy-
MM-dd}-%i.log">
17        <ThresholdFilter level="INFO" onMatch="ACCEPT"
onMismatch="DENY"/>
18        <PatternLayout pattern="${LOG_PATTERN}"/>
19        <Policies>
20          <TimeBasedTriggeringPolicy/>
21          <SizeBasedTriggeringPolicy size="250 MB"/>
22        </Policies>
23      </RollingFile>
24    </appenders>
25    <loggers>
26      <root level="all">
27        <appender-ref ref="Console"/>
28        <appender-ref ref="RollingFileInfo"/>
29      </root>
30    </loggers>
31  </configuration>
32

```

③ 命令行中增加参数

```

1  --driver-java-options "-DlogName=flow_xxxxx -DlogDir=/opt/spark/logs" \
2  --driver-class-path "/opt/spark/extraConf"

```

注: 原 `log4j.configuration` 可以弃用, log4j2会自动扫描classPath下 `log4j2.xml` 的文件;

`logName` 及 `logDir` 是log4j2.xml中参数变量, 用来指定文件名及输出路径;

`logDir` 需要配置成挂载目录, 用以保存到客户端中.

六. 配置项

<https://spark.apache.org/docs/3.0.0/running-on-kubernetes.html#spark-properties>

<https://spark.apache.org/docs/3.0.0/configuration.html>

七. 注意

1. 异地构建镜像流程

如果本地集群网络IO较差需要异地构建镜像, 可通过以下流程进行操作

(1) 将 spark 打成 tar 包, 传到异地端并解压

(2) 修改 `${spark-path}/kubernetes/dockerfiles/spark/Dockerfile` (本地集群 Build 镜像不需要)

倒第二行给脚本权限.

RUN `chmod -R 777 /opt/entrypoint.sh`

否则最后构建的镜像无法运行, 会报错 `spark ENTRYPOINT ["/opt/entrypoint.sh"] permission denied: unknown;`

(3) `cd ${spark-path}` . 运行 `docker images | grep spark` 确认无之前构建的镜像,

如果之前打过, 执行 `docker rmi [-f] <镜像ID>` 删除镜像.

(4) 执行 `./bin/docker-image-tool.sh -r 192.168.1.74 -t spark-v3.0.0 build` 构建 r-image

该过程需等待很长时间

(5) 执行 `docker images | grep spark` 查看生成镜像 并复制 <镜像id>;

(6) 保存镜像 `docker save 镜像ID > ./spark3.tar` ; 并传到本地集群中.

(7) 恢复镜像 `docker load < ./spark3.tar`

(8) 执行 `docker images` 查看镜像ID

(9) 修改镜像标签 `docker tag 镜像ID 192.168.1.74/spark/spark3:v3.0.0`

(10) 推送镜像 `docker push 192.168.1.74/spark/spark3:v3.0.0`

2. 断点调试的流程

首先在 `spark-default.conf` 中增加参数, 提交任务之后, driver pod 会卡住并等待监听连接.

```
1 | spark.kubernetes.driverEnv.SPARK_SUBMIT_OPTS="--  
agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=50014
```

在 Kubernetes 中提供了名为 Service 的资源, 它是将运行在一组 Pods 上的应用程序公开为网络服务的抽象方法。Service 大致可以分为以下几类:

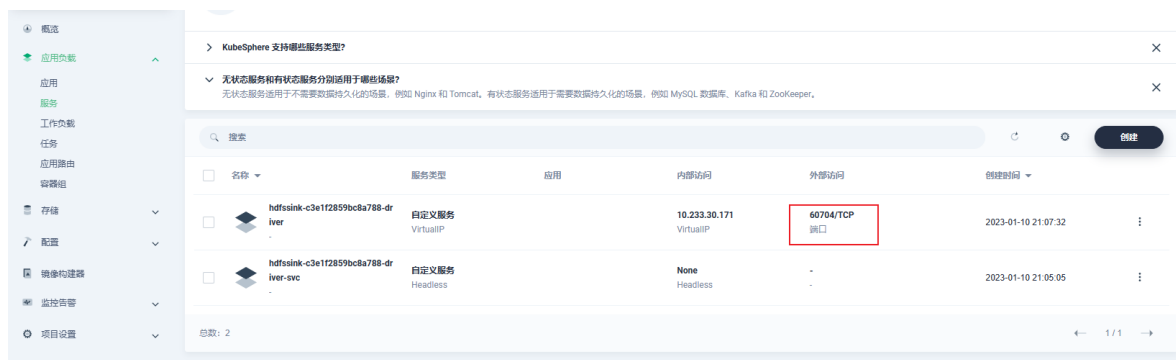
- ClusterIP: 默认值, 通过集群的内部 IP 暴露服务, 选择该值时服务只能够在集群内部访问。
- NodePort: 将 Service 通过指定的 Node 上的端口暴露给外部, 通过此方法, 就可以在集群外部访问服务。
- LoadBalancer: 使用云提供商的负载均衡器向外部暴露服务。外部负载均衡器可以将流量路由到自动创建的 NodePort 服务和 ClusterIP 服务上。
- ExternalName: 把集群外部的服务引入集群内部, 直接使用

在 Kubernetes 中, 当 Spark 的 driver pod 启动后, 默认会帮我们创建一个 ClusterIP 类型的 service, 且当前开源版本中我们无法修改 service 类型。

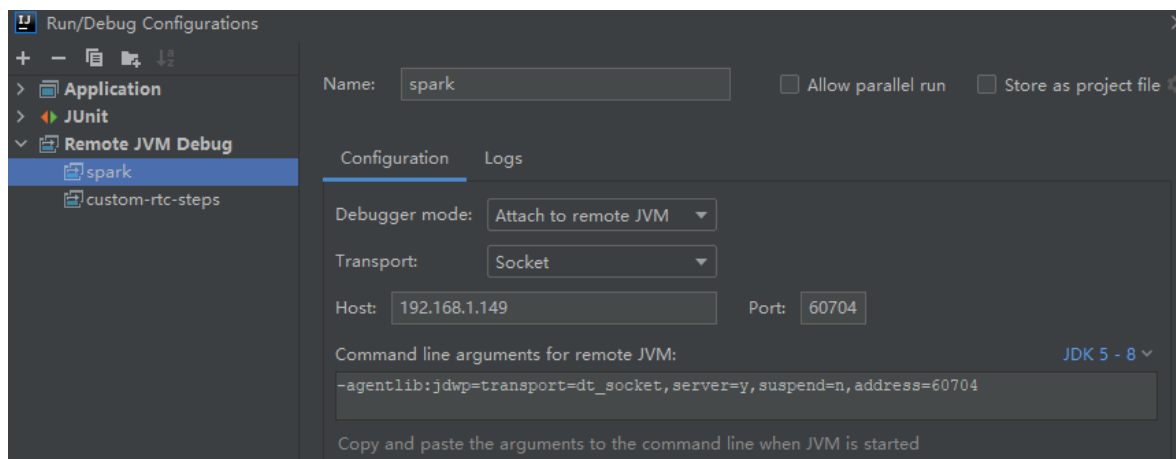
(1) 由于 Spark 默认创建的是 ClusterIP 类型 service, 无法直接在外访问, 我们可以直接为 driver pod 创建一个新的 NodePort/LoadBalancer 类型的 service, 以 NodePort 为例, 命令如下:

```
1 | kubectl expose pod <driver-pod-name> --type=NodePort --port 50014
```

(2) 查看web UI 选择服务



(3) 打开IDEA配置远程调试.



(4) 点击调试按钮进行调试

3. 日志输出异常, 排查流程