Monitoring

Date published: 2020-03-06
Date modified: 2020-03-06

Legal Notice

Contents

Flink Dashboard

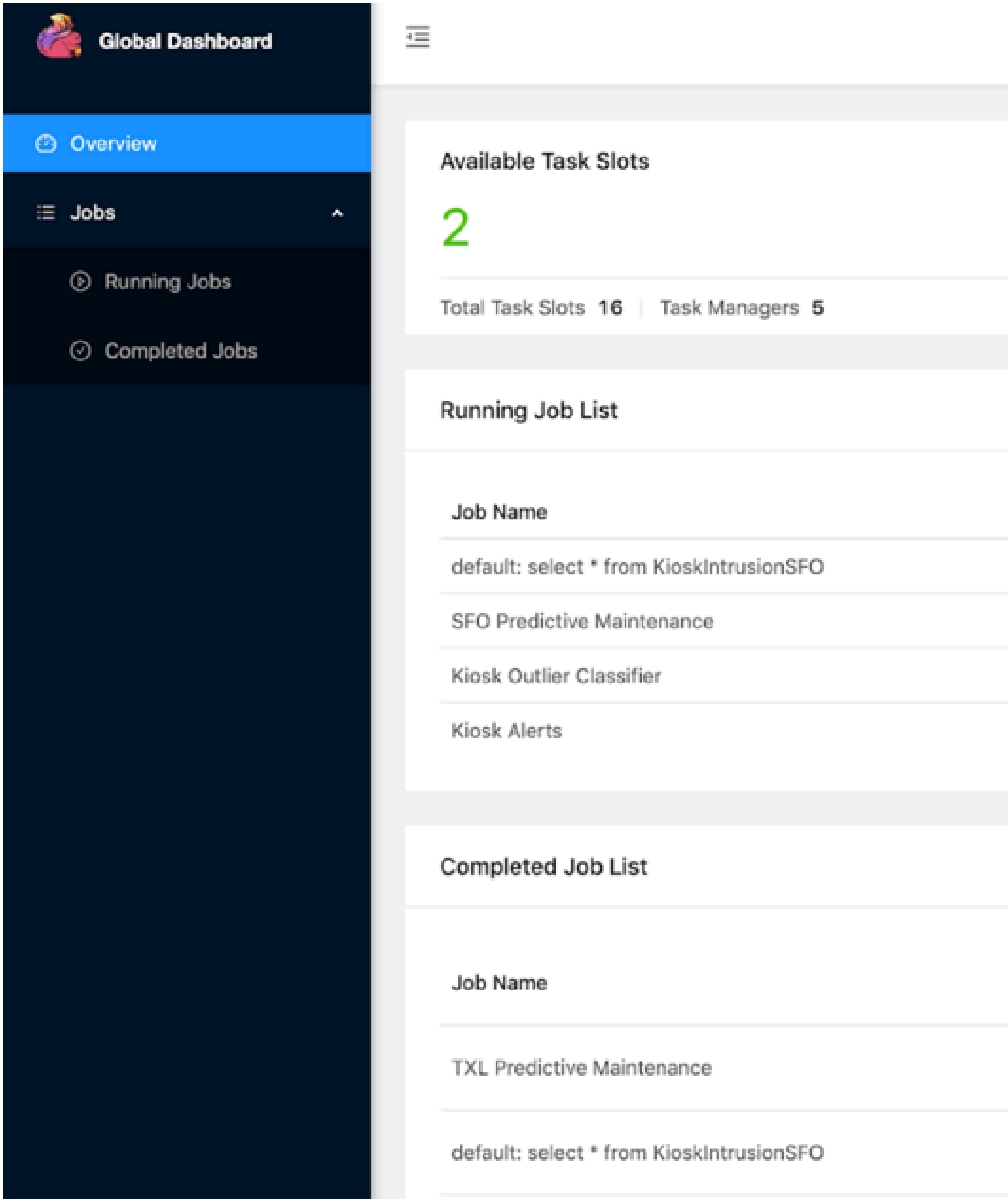
The Flink Dashboard is a built-in monitoring interface for Flink applications in Cloudera Streaming Analytics. You can monitor your running, completed and stopped Flink jobs on the dashboard. You reach the Flink Dashboard through Cloudera Manager.

After deploying Flink and the required components, you can configure and monitor each component individually, or the whole cluster with Cloudera Manager. For the general use of Cloudera Manager, see the Cloudera Manager documentation.

The Flink Dashboard acts as a single UI for monitoring all the jobs running on the YARN cluster. It shows all the running, failed, and finished jobs.

Note: The Flink Dashboard is an updated version of the Flink HistoryServer.

You can also use the dashboard to navigate between the different Flink clusters from a central place.



Available Task Slots

2

Total Task Slots 16 | Task Managers 5

Running Job List

Job Name
default: select * from KioskIntrusionSFO
SFO Predictive Maintenance
Kiosk Outlier Classifier
Kiosk Alerts

Completed Job List

Job Name
TXL Predictive Maintenance
default: select * from KioskIntrusionSFO

Kafka Metrics Reporter

In Cloudera Streaming Analytics, Kafka Metrics Reporter is available as another monitoring solution when Kafka is used as a connector within the pipeline to retrieve metrics about your streaming performance.

Flink offers a flexible Metrics Reporter API for collecting the metrics generated by your streaming pipelines. Cloudera provides an additional implementation of this, which writes metrics to Kafka with the following JSON schema:

```
{
  "timestamp" : number -> millisecond timestamp of the metric record
  "name" : string -> name of the metric
    (e.g. numBytesOut)
  "type" : string -> metric type enum: GAUGE, COUNTER, METER, HISTOGRAM
  "variables" : {string => string} -> Scope variables
    (e.g. {"<job_id>" : "123", "<host>" : "localhost"})
  "values" : {string => number} -> Metric specific values
    (e.g. {"count" : 100})
}
```

For more information about Metrics Reporter, see the Apache Flink documentation.

Configuration of Kafka Metrics Reporter

The Kafka metrics reporter can be configured similarly to other upstream metric reporters.

Required parameters

- topic: target Kafka topic where the metric records will be written at the configured intervals
- bootstrap.servers: Kafka server addresses to set up the producer

Optional parameters

- interval: reporting interval, default value is 10 seconds, format is 60 SECONDS
- log.errors: logging of metric reporting errors, value either true or false

You can configure the Kafka metrics reporter per job using the following command line properties:

```
flink run --jobmanager yarn-cluster --detached --parallelism 2 --yarnname He
apMonitor \
-yD metrics.reporter.kafka.class=org.apache.flink.metrics.kafka.KafkaMetrics
Reporter \
-yD metrics.reporter.kafka.topic=metrics-topic.log \
-yD metrics.reporter.kafka.bootstrap.servers=kafka-broker:9091 \
-yD metrics.reporter.kafka.interval="60 SECONDS" \
-yD metrics.reporter.kafka.log.errors=false \
flink-simple-tutorial-1.1-SNAPSHOT.jar
```

The following is a more advanced Flink command that also contains security related configurations:

```
flink run --jobmanager yarn-cluster --detached --parallelism 2 --yarnname He
apMonitor \
-yD security.kerberos.login.keytab=some.keytab \
-yD security.kerberos.login.principal=some_principal \
-yD metrics.reporter.kafka.class=org.apache.flink.metrics.kafka.KafkaMetrics
Reporter \
-yD metrics.reporter.kafka.topic=metrics-topic.log \
-yD metrics.reporter.kafka.bootstrap.servers=kafka_broker_host:9093 \
-yD metrics.reporter.kafka.interval="60 SECONDS" \
-yD metrics.reporter.kafka.log.errors=false \
-yD metrics.reporter.kafka.security.protocol=SASL_SSL \
-yD metrics.reporter.kafka.sasl.kerberos.service.name=kafka \
-yD metrics.reporter.kafka.ssl.truststore.location=truststore.jks \
flink-simple-tutorial-1.1-SNAPSHOT.jar
```

You can also set the metrics properties globally in Cloudera Manager using Flink Client Advanced Configuration Snippet (Safety Valve) for flink-conf-xml/flink-conf.xml.

Arbitrary Kafka producer properties

The reporter supports passing arbitrary Kafka producer properties that can be used to modify the behavior, enable security, and so on. Serializer classes should not be modified as it can lead to reporting errors.

See the following example configuration of the Kafka Metrics Reporter:

```
# Required configuration
metrics.reporter.kafka.class:
org.apache.flink.metrics.kafka.KafkaMetricsReporter
metrics.reporter.kafka.topic: metrics-topic.log
metrics.reporter.kafka.bootstrap.servers: broker1:9092,broker2:9092

# Optional configuration
metrics.reporter.kafka.interval: 60 SECONDS
metrics.reporter.kafka.log.errors: false

# Optional Kafka producer properties
metrics.reporter.kafka.security.protocol : SSL
metrics.reporter.kafka.ssl.truststore.location :
/var/private/ssl/kafka.client.truststore.jks
```

Note: Any optional property with metrics.reporter.kafka. prefix tag is processed as Kafka client configuration.

For example: metrics.reporter.kafka.property_name : property_value will be converted to property_name : property_value.

Kafka Logging

Cloudera Streaming Analytics include a Kafka log appender to provide a production grade solution. You can use Kafka logging to have a scalable storage layer for the logs, and you can also integrate with other logging applications with more simpler solutions.

By default, Flink logs are directed to files that can be viewed on the Flink GUI independently for each container. This solution is a best practice for a YARN application defaults, but long running production applications are lacking this function.

The log appender in the Flink parcel collects the logs into Kafka topics in a JSON format that is designed for downstream consumption in an enterprise log aggregation framework.

There are several benefits of storing the logs in Kafka:

- Provides a scalable storage layer for the logs
- Integrates easily with existing applications that has a simple logger configuration

To enable Kafka based logging, include the following log configuration in the Cloudera Manager Flink configuration page:

```
# Enable both file and kafka based logging
log4j.rootLogger=INFO, file, kafka log4j.appender.kafka=com.cloudera.kafk
a.log4jappender.KafkaLog4jAppender
```

```
log4j.appender.kafka.topic=flink.logs
log4j.appender.kafka.brokerList=<broker_host>:9092
# Log layout configuration
log4j.appender.kafka.layout=net.logstash.log4j.JSONEventLayoutV1
log4j.appender.kafka.layout.UserFields=yarnContainerId:${yarnContainerId}
```

With this configuration, logs are written to the flink.logs topic with a JSON format. The topic contains an extra field with the YARN container identifier for easier log separation. These additional logging configurations are added to the log4j.properties Flink default file. Any duplicate key overrides the previously configured values in the file.

When set up correctly the resulting logs should be similar to the following:

```
{ "source_host": "<flink_host>",
  "method": "completePendingCheckpoint",
  "level": "INFO",
  "message": "Completed checkpoint 1 for job 5e70cf704ed010372e2007333db10c
f0 (50738 bytes in 2721 ms).",
  "mdc": {},
  "yarnContainerId": "container_1571051884501_0001_01_000001",
  "@timestamp": "2019-10-14T11:21:07.400Z",
  "file": "CheckpointCoordinator.java",
  "line_number": "906",
  "thread_name": "jobmanager-future-thread-1",
  "@version": 1,
  "logger_name":    "org.apache.flink.runtime.checkpoint.CheckpointCoordi
nator",
  "class": "org.apache.flink.runtime.checkpoint.CheckpointCoordinator"
}
```

Security

In a secure environment with Kerberos and TLS enabled, add the following extra parameters:

```
log4j.appender.kafka.securityProtocol=SASL_SSL
log4j.appender.kafka.saslKerberosServiceName=kafka
log4j.appender.kafka.sslTruststoreLocation=/samePathOnAllNodes/truststore.
jks
log4j.appender.kafka.clientJaasConfPath=kafka.jaas.conf
```

Also provide the following kafka.jaas.conf file and ship it to the cluster with the -yt kafka.jaas.conf parameter for the flink run command.

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="krb5.keytab"
  principal="<user_name>";
};
```

Streams Messaging Manager integration

You can use the Streams Messaging Manager (SMM) UI to monitor end-to-end latency of your Flink application when using Kafka as a datastream connector.

End-to-end latency throughout the pipeline can be monitored using SMM. To use SMM with Flink, interceptors need to be enabled for Kafka in the Flink connectors.

For more information about enabling interceptors, see the SMM documentation.