How to Log in Apache Spark

Contributed by

Nicolas A Perez (/blog/author/nicolas-perez/)

(/blog/author/nicolas-perez/)

6 min read

An important part of any application is the underlying log system we incorporate into it. Logs are not only for debugging and traceability, but also for business intelligence. Building a robust logging system within our apps could be use as a great insights of the business problems we are solving.

Log4j in Apache Spark

Spark uses *log4j* as the standard library for its own logging. Everything that happens inside Spark gets logged to the shell console and to the configured underlying storage. Spark also provides a template for app writers so we could use the same_log4j_ libraries to add whatever *messages* we want to the existing and in place implementation of logging in Spark.

Configuring Log4j

Under the _SPARK_HOME/conf_ folder, there is_log4j.properties.template_ file which serves as an starting point for our own *logging* system.

Based on this file, we created the *log4j.properties* file and put it under the same directory.

log4j.properties looks like follows:

```
log4j.appender.myConsoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.myConsoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.myConsoleAppender.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

log4j.appender.RollingAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.RollingAppender.File=/var/log/spark.log
log4j.appender.RollingAppender.DatePattern='.'yyyy-MM-dd
log4j.appender.RollingAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.RollingAppender.layout.ConversionPattern=[%p] %d %c %M - %m%n

log4j.appender.RollingAppenderU=org.apache.log4j.DailyRollingFileAppender
log4j.appender.RollingAppenderU.File=/var/log/sparkU.log
log4j.appender.RollingAppenderU.DatePattern='.'yyyy-MM-dd
log4j.appender.RollingAppenderU.layout=org.apache.log4j.PatternLayout
log4j.appender.RollingAppenderU.layout.ConversionPattern=[%p] %d %c %M - %m%n
```

By default, everything goes to console and file

log4j.rootLogger=INFO, RollingAppender, myConsoleAppender

My custom logging goes to another file

log4j.logger.myLogger=INFO, RollingAppenderU

The noisier spark logs go to file only

log4j.logger.spark.storage=INFO, RollingAppender
log4j.additivity.spark.storage=false
log4j.logger.spark.scheduler=INFO, RollingAppender
log4j.additivity.spark.scheduler=false
log4j.logger.spark.CacheTracker=INFO, RollingAppender
log4j.additivity.spark.CacheTracker=false
log4j.logger.spark.CacheTrackerActor=INFO, RollingAppender
log4j.additivity.spark.CacheTrackerActor=false
log4j.logger.spark.MapOutputTrackerActor=INFO, RollingAppender
log4j.additivity.spark.MapOutputTrackerActor=false
log4j.logger.spark.MapOutputTracker=INFO, RollingAppender
log4j.additivty.spark.MapOutputTracker=false

Basically, we want to hide all logs Spark generates so we don't have to deal with them in the shell. We redirect them to be logged in the file system. On the other hand, we want our own logs to be logged in the shell and a separated file so they don't get mixed up with the ones from Spark. From here, we will point Splunk to the files where our own logs are which in this particular case is_/var/log/sparkU.log._

This (*log4j.properties*) file is picked up by Spark when the application starts so we don't have to do anything aside of placing it in the mentioned location.

Writing Our Own Logs

Now that we have configured the components that Spark requires in order to manage our logs, we just need to start writing logs within our apps.

In order to show how this is done, let's write a small app that helps us in the demonstration.

Our App:

```
object app {
def main(args: Array[String]) {
val log = LogManager.getRootLogger
log.setLevel(Level.WARN)

val conf = new SparkConf().setAppName("demo-app")
val sc = new SparkContext(conf)

log.warn("Hello demo")

val data = sc.parallelize(1 to 100000)

log.warn("I am done")
}
}
```

Running this Spark app will demonstrate that our log system works. We will be able to see how _Hello demo _and *I am done* messages being logged in the shell and in the file system while the Spark logs will only go to the file system.

So far, everything seems easy, yet there is a problem we haven't mentioned.

The class *org.apache.log4j.Logger* is not *serializable* which implies we cannot use it inside a *closure* while doing operations on some parts of the Spark API.

For example, if we do in our app:

```
val log = LogManager.getRootLogger
val data = sc.parallelize(1 to 100000)

data.map { value =>
log.info(value)
value.toString
}
```

This will fail when running on Spark. Spark complaints that the_log_ object is not *Serializable* so it cannot be sent over the network to the Spark workers.

This problem is actually easy to solve. Let's create a class that does something to our data set while doing a lot of logging.

```
class Mapper(n: Int) extends Serializable{
@transient lazy val log = org.apache.log4j.LogManager.getLogger("myLogger")

def doSomeMappingOnDataSetAndLogIt(rdd: RDD[Int]): RDD[String] =
rdd.map{ i =>
log.warn("mapping: " + i)
(i + n).toString
}
}
```

Mapper receives a _RDD[Int] _and returns a *RDD[String]* and it also logs what value its being mapped. In this case, noted how the_log_ object has been marked as *@transient* which allows the serialization system to ignore the *log* object. Now, *Mapper* is being serialized and sent to each worker but the log object is being resolved when it is needed in the worker, solving our

problem.

Another solution is to wrap the *log* object into a _object_construct and use it all over the place. We rather have *log* within the class we are going to use it, but the alternative is also valid.

At this point, our entire app looks like follows:

```
import org.apache.log4j.{Level, LogManager, PropertyConfigurator}
import org.apache.spark._
import org.apache.spark.rdd.RDD

class Mapper(n: Int) extends Serializable{
@transient lazy val log = org.apache.log4j.LogManager.getLogger("myLogger")
def doSomeMappingOnDataSetAndLogIt(rdd: RDD[Int]): RDD[String] =
rdd.map{ i =>
log.warn("mapping: " + i)
(i + n).toString
}
}
object Mapper {
def apply(n: Int): Mapper = new Mapper(n)
}
object app {
def main(args: Array[String]) {
val log = LogManager.getRootLogger
log.setLevel(Level.WARN)
val conf = new SparkConf().setAppName("demo-app")
val sc = new SparkContext(conf)

log.warn("Hello demo")
```

```
val data = sc.parallelize(1 to 100000)
val mapper = Mapper(1)
val other = mapper.doSomeMappingOnDataSetAndLogIt(data)
other.collect()

log.warn("I am done")
}
}
```

Conclusions

Our logs are now being shown in the shell and also stored in their own files. Spark logs are being hidden from the shell and being logged into their own file. We also solved the serialization problem that appears when trying to log in different workers.

We now can build more robust BI systems based on our own Spark logs as we do with other non distributed systems and applications we have today. Business Intelligence is for us a very big deal and having the right insights is always nice to have.

This post was originally published here (https://medium.com/@anicolaspp/how-to-log-in-apache-spark-f4204fad78a#.xo31z5vrd).

This blog post was published March 01, 2016.

Categories

All (/blog/)

Apache Drill (/blog/apache-drill/)

Apache Hadoop (/blog/apache-hadoop/)

Apache Hive (/blog/apache-hive/)

Apache Mesos (/blog/apache-mesos/)

Apache Myriad (/blog/apache-myriad/)

Apache Spark (/blog/apache-spark/)

Cloud Computing (/blog/cloud-computing/)

Enterprise Data Hub (/blog/enterprise-data-hub/)

Machine Learning (/blog/machine-learning/)

MapR Platform (/blog/mapr-platform/)

MapReduce (/blog/mapreduce/)

NoSQL (/blog/nosql/)

Open Source Software (/blog/open-source-software/)

Partners (/blog/partners/)

Streaming (/blog/streaming/)

Use Cases (/blog/use-cases/)

Whiteboard Walkthrough Videos (/blog/whiteboard-walkthrough-videos/)

Anuj Mehra • 2 years ago • edited

Hi,

Can we intergate MDC with spark logging?
I want to generate the "JobID" on my 'Driver Node' and pass this 'JobID' to all the worker nodes.

Thanks
Anuj

• Reply • Share ›

Raja K Thaw • 3 years ago

Hi Nicolas,

I face one issue as mentioned:

http://stackoverflow.com/qu...

I made many changes in spark-defaults.conf. Even I tried to have all jars in SPARK_CLASSPATH="/usr/share/java/*.jar" in spark-env.sh and download extra jars , I still get an error as mentioned in that stackoverflow .

Any idea how I can resolve? I am using SBT and run from command line.

Thanks,

Raja

• Reply • Share ›

golan2 • 2 years ago

Logs corruption:
Following the instructions above means that we log to the same file (/var/log/spark.log) from several executors.
i.e. from several jvm processes

In "regular" applications this is not advised.
https://stackoverflow.com/q...
any idea how spark solved it?

When using RollingFileAppender (size-based) it indeed leads to corruption.
The logs rotated too soon leading to many small files containing very little in each of them.
Using DailyRollingFileAppender seems to overcome this problem. Not sure how exactly...

• Reply • Share ›

(https://www.linkedin.com/company/mapr-technologies) (https://www.facebook.com/maprtech/) (https://twitter.com/mapr)

(https://www.youtube.com/user/maprtech) (/company/contact-mapr/)