Ejecutando Spark en YARN

- Seguridad
- Lanzamiento de Spark en YARN
  - Agregar otros JAR
- Preparativos
- Configuración
- Depuración de su aplicación
  - Propiedades de Spark
  - Patrones disponibles para la URL del registro del ejecutor personalizado de SHS
- Descripción general de la configuración y asignación de recursos
- Descripción general de la programación a nivel de etapa
- Notas importantes
- Kerberos
  - Configuración de Kerberos específica de YARN
  - Solución de problemas de Kerberos
- Configuración del servicio Shuffle externo
- Cómo iniciar su aplicación con Apache Oozie
- Uso del servidor de historial de Spark para reemplazar la interfaz web de Spark
- Ejecución de varias versiones del servicio Spark Shuffle

La compatibilidad para ejecutarse en YARN (Hadoop NextGen) se agregó a Spark en la versión 0.6.0 y se mejoró en versiones posteriores.

Seguridad

Las funciones de seguridad, como la autenticación, no están habilitadas de forma predeterminada. Al implementar un clúster abierto a Internet o a una red que no es de confianza, es importante proteger el acceso al clúster para evitar que se ejecuten aplicaciones no autorizadas en él. Consulta Seguridad de Spark y las secciones de seguridad específicas de este documento antes de ejecutar Spark.

Lanzamiento de Spark en YARN

Asegúrese de que `HADOOP_CONF_DIR` o `YARN_CONF_DIR` apunte al directorio que contiene los archivos de configuración (del lado del cliente) para el clúster Hadoop. Estas configuraciones se utilizan para escribir en HDFS y conectarse al administrador de recursos de YARN. La configuración contenida en este directorio se distribuirá al clúster de YARN para que todos los contenedores utilizados por la aplicación utilicen la misma configuración. Si la configuración hace referencia a propiedades del sistema Java o variables de entorno no administradas por YARN, también se deben configurar en la configuración de la aplicación Spark (controlador, ejecutores y administrador de recursos cuando se ejecuta en modo cliente).

Hay dos modos de implementación que se pueden usar para iniciar aplicaciones Spark en YARN.

- En `modo cluster`, el controlador de Spark se ejecuta dentro de un proceso maestro de aplicación que es administrado por YARN en el clúster, y el cliente puede irse después de iniciar la aplicación.
- En `modo cliente`, el controlador se ejecuta en el proceso del cliente y el maestro de aplicación solo se usa para solicitar recursos de YARN.

A diferencia de otros administradores de clúster compatibles con Spark en los que la dirección del maestro se especifica en el `--master parameter`, en el modo YARN la dirección del ResourceManager se obtiene de la configuración de Hadoop. Por lo tanto, el `–master` parameter es `yarn`.

Para iniciar una aplicación Spark en `clustermodo`:

```
$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode
cluster [options] <app jar> [app options]
```

Por ejemplo:

```
$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 1 \
    --queue thequeue \
    examples/jars/spark-examples*.jar \
    10
```

Lo anterior inicia un programa cliente YARN que inicia el Application Master predeterminado. Luego, SparkPi se ejecutará como un subproceso secundario del Application Master. El cliente sondeará periódicamente el Application Master para obtener actualizaciones de estado y las mostrará en la consola. El cliente saldrá una vez que su aplicación haya terminado de ejecutarse. Consulte la sección Depuración de su aplicación a continuación para saber cómo ver los registros del controlador y del ejecutor.

Para iniciar una aplicación Spark en modo `cliente`, haga lo mismo, pero reemplace `cluster` por `client`. A continuación, se muestra cómo ejecutar `spark-shell` en modo cliente :

```
$ ./bin/spark-shell --master yarn --deploy-mode client
```

Agregar otros JAR

En `modo cluster`, el controlador se ejecuta en una máquina diferente a la del cliente, por lo que `SparkContext.addJar` no funcionará de inmediato con archivos que sean locales para el cliente. Para que los archivos del cliente estén disponibles para `SparkContext.addJar`, inclúyalos con la opción `--jars` en el comando de inicio.

```
$ ./bin/spark-submit --class my.main.Class \
```

```
    --master yarn \
    --deploy-mode cluster \
    --jars my-other-jar.jar,my-other-other-jar.jar \
    my-main-jar.jar \
    app_arg1 app_arg2
```

Preparativos

Para ejecutar Spark en YARN se necesita una distribución binaria de Spark que esté creada con soporte para YARN. Las distribuciones binarias se pueden descargar desde la página de descargas del sitio web del proyecto. Hay dos variantes de distribuciones binarias de Spark que puedes descargar.

Una está preconstruida con una determinada versión de Apache Hadoop; esta distribución Spark contiene un entorno de ejecución de Hadoop integrado, por lo que la llamamos distribución Spark `with-hadoop`.

La otra está preconstruida con Hadoop proporcionado por el usuario; dado que esta distribución Spark no contiene un entorno de ejecución de Hadoop integrado, es más pequeña, pero los usuarios deben proporcionar una instalación de Hadoop por separado. Llamamos a esta variante distribución Spark `no-hadoop`.

Para la distribución Spark `with-hadoop`, dado que ya contiene un entorno de ejecución de Hadoop integrado, de forma predeterminada, cuando se envía un trabajo al clúster Hadoop Yarn, para evitar conflictos de jar, no completará la ruta de clase de Yarn en Spark. Para anular este comportamiento, puedes establecer `spark.yarn.populateHadoopClasspath=true`.

Para la distribución Spark `no-hadoop`, Spark completará la ruta de clase de Yarn de forma predeterminada para obtener el entorno de ejecución de Hadoop. Para `with-hadoop` la distribución de Spark, si su aplicación depende de una biblioteca determinada que solo está disponible en el clúster, puede intentar completar la ruta de clase de Yarn configurando la propiedad mencionada anteriormente. Si al hacerlo se produce un problema de conflicto de archivos jar, deberá desactivarla e incluir esta biblioteca en el archivo jar de su aplicación.

Para crear Spark usted mismo, consulte Creación de Spark .

Para que los archivos jar de tiempo de ejecución de Spark sean accesibles desde el lado de YARN, puede especificar `spark.yarn.archive` o `spark.yarn.jars`. Para obtener más detalles, consulte Propiedades de Spark . Si no se especifica `spark.yarn.archive` ni `spark.yarn.jars`, Spark creará un archivo zip con todos los archivos jar `$SPARK_HOME/jars` y lo cargará en la memoria caché distribuida.

Configuración

La mayoría de las configuraciones son las mismas para Spark en YARN que para otros modos de implementación. Consulta la página de configuración para obtener más información sobre ellas. Estas son configuraciones específicas para Spark en YARN.

Depuración de su aplicación

En la terminología de YARN, los ejecutores y los maestros de aplicaciones se ejecutan dentro de “contenedores”. YARN tiene dos modos para manejar los registros de contenedores después de que se haya completado una aplicación. Si la agregación de registros está activada (con la configuración `yarn.log-aggregation-enable`), los registros de contenedores se copian a HDFS y se eliminan en la máquina local. Estos registros se pueden ver desde cualquier lugar del clúster con el comando `yarn logs`.

```
yarn logs -applicationId <app ID>
```

imprimirá el contenido de todos los archivos de registro de todos los contenedores de la aplicación dada. También puede ver los archivos de registro de los contenedores directamente en HDFS usando el shell o la API de HDFS. El directorio donde se encuentran se puede encontrar mirando las configuraciones de YARN (`yarn.nodemanager.remote-app-log-dir` y `yarn.nodemanager.remote-app-log-dir-suffix`). Los registros también están disponibles en la interfaz de usuario web de Spark en la pestaña Ejecutores. Debe tener tanto el servidor de historial de Spark como el servidor de historial de MapReduce ejecutándose y configurados `yarn.log.server.url` correctamente `yarn-site.xml`. La URL de registro en la interfaz de usuario del servidor de historial de Spark lo redireccionará al servidor de historial de MapReduce para mostrar los registros agregados.

Cuando la agregación de registros no está activada, los registros se conservan localmente en cada máquina bajo `YARN_APP_LOGS_DIR`, que generalmente está configurado para `/tmp/logs` o `$HADOOP_HOME/logs/userlogs` según la versión e instalación de Hadoop. Para ver los registros de un contenedor, es necesario ir al host que los contiene y buscar en este directorio. Los subdirectorios organizan los archivos de registro por ID de aplicación e ID de contenedor. Los registros también están disponibles en la interfaz de usuario web de Spark en la pestaña Ejecutores y no requieren ejecutar el servidor de historial de MapReduce.

Para revisar el entorno de lanzamiento por contenedor, aumente `yarn.nodemanager.delete.debug-delay-sec` a un valor alto (por ejemplo, `36000`) y, a continuación, acceda a la memoria caché de la aplicación a través `yarn.nodemanager.local-dirs` de los nodos en los que se lanzan los contenedores. Este directorio contiene el script de lanzamiento, los archivos JAR y todas las variables de entorno utilizadas para lanzar cada contenedor. Este proceso es útil para depurar problemas de rutas de clase en particular. (Tenga en cuenta que para habilitar esto se requieren privilegios de administrador en la configuración del clúster y un reinicio de todos los administradores de nodos. Por lo tanto, esto no es aplicable a los clústeres alojados).

Para utilizar una configuración personalizada de log4j2 para el maestro o los ejecutores de la aplicación, estas son las opciones:

- cargue un archivo personalizado `log4j2.properties`usando `spark-submit`, agregándolo `–files` a la lista de archivos que se cargarán con la aplicación.
- agregar `-Dlog4j.configurationFile=<location of configuration file>`a `spark.driver.extraJavaOptions` (para el controlador) o `spark.executor.extraJavaOptions` (para los ejecutores). Tenga en cuenta que

si se utiliza un archivo, el `file:` protocolo debe proporcionarse explícitamente y el archivo debe existir localmente en todos los nodos.
- Actualice el `$SPARK_CONF_DIR/log4j2.properties`archivo y se cargará automáticamente junto con las demás configuraciones. Tenga en cuenta que las otras dos opciones tienen mayor prioridad que esta opción si se especifican varias opciones.

Tenga en cuenta que, para la primera opción, tanto los ejecutores como el maestro de la aplicación compartirán la misma configuración de log4j, lo que puede causar problemas cuando se ejecutan en el mismo nodo (por ejemplo, al intentar escribir en el mismo archivo de registro).

Si necesita una referencia a la ubicación adecuada para colocar los archivos de registro en YARN para que YARN pueda mostrarlos y agregarlos correctamente, utilice `spark.yarn.app.container.log.dir` en su `log4j2.properties`. Por ejemplo, `appender.file_appender.fileName=${sys:spark.yarn.app.container.log.dir}/spark.log`. Para las aplicaciones de transmisión, configurar `RollingFileAppender` y establecer la ubicación del archivo en el directorio de registro de YARN evitará el desbordamiento del disco causado por archivos de registro de gran tamaño, y se puede acceder a los registros utilizando la utilidad de registro de YARN.

Para utilizar un archivo metrics.properties personalizado para el maestro y los ejecutores de la aplicación, actualice el `$SPARK_CONF_DIR/metrics.properties` archivo. Se cargará automáticamente con otras configuraciones, por lo que no necesita especificarlo manualmente con `--files`.

Propiedades de Spark

Nombre de la propiedad	Por defecto	Significado	Desde la versión
`spark.yarn.am.memory`	`512m`	Cantidad de memoria que se utilizará para el maestro de aplicaciones YARN en modo cliente, en el mismo formato que las cadenas de memoria de JVM (por ejemplo `512m`, , `2g`). En modo de clúster, utilice `spark.driver.memory`en su lugar. Utilice sufijos en minúscula, como `k`, , `m`, `g`y `t`, `p`para kibi-, mebi-, gibi-, tebi- y pebibytes, respectivamente.	1.3.0
`spark.yarn.am.resource.{resource-type}.amount`	(none)	Cantidad de recursos que se utilizarán para el maestro de aplicaciones YARN en modo cliente. En modo clúster, utilice `spark.yarn.driver.resource.<resource-type>.amount`en su lugar. Tenga en cuenta que esta función solo se puede utilizar con YARN 3.0+. Para obtener referencia, consulte la documentación del modelo de recursos de YARN: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ResourceModel.html Ejemplo: Para solicitar recursos de GPU de YARN, utilice:`spark.yarn.am.resource.yarn.io/gpu.amount`	3.0.0
`spark.yarn.applicationType`	`SPARK`	Define tipos de aplicaciones más específicos, p. ej `SPARK`. `SPARK-SQL`, , `SPARK-STREAMING`,	3.1.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		`SPARK-MLLIB`y `SPARK-GRAPH`. Tenga cuidado de no superar los 20 caracteres.	
`spark.yarn.driver.resource.{resource-type}.amount`	(none)	Cantidad de recursos que se utilizarán para el maestro de aplicaciones YARN en modo de clúster. Tenga en cuenta que esta función solo se puede utilizar con YARN 3.0+. Para obtener referencia, consulte la documentación del modelo de recursos de YARN: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ResourceModel.html Ejemplo: Para solicitar recursos de GPU de YARN, utilice:`spark.yarn.driver.resource.yarn.io/gpu.amount`	3.0.0
`spark.yarn.executor.resource.{resource-type}.amount`	(none)	Cantidad de recursos que se utilizarán por proceso ejecutor. Tenga en cuenta que esta función solo se puede utilizar con YARN 3.0+. Para obtener referencia, consulte la documentación del modelo de recursos de YARN: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ResourceModel.html Ejemplo: Para solicitar recursos de GPU de YARN, utilice:`spark.yarn.executor.resource.yarn.io/gpu.amount`	3.0.0
`spark.yarn.resourceGpuDeviceName`	`yarn.io/gpu`	Especifique la asignación del tipo de recurso Spark `gpu`al recurso YARN que representa una GPU. De forma predeterminada, YARN utiliza , `yarn.io/gpu`pero si YARN se ha configurado con un tipo de recurso personalizado, esto permite reasignarlo. Se aplica cuando se utilizan las `spark.{driver/executor}.resource.gpu.*`configuraciones.	3.2.1
`spark.yarn.resourceFpgaDeviceName`	`yarn.io/fpga`	Especifique la asignación del tipo de recurso Spark `fpga`al recurso YARN que representa un FPGA. De forma predeterminada, YARN utiliza , `yarn.io/fpga`pero si YARN se ha configurado con un tipo de recurso personalizado, esto permite reasignarlo. Se aplica cuando se utilizan las `spark.{driver/executor}.resource.fpga.*`configuraciones.	3.2.1
`spark.yarn.am.cores`	1	Número de núcleos que se utilizarán para el maestro de aplicaciones YARN en modo cliente. En modo clúster, utilice `spark.driver.cores`en su lugar.	1.3.0
`spark.yarn.am.waitTime`	100s	Solo se utiliza en `cluster`modo Tiempo que el maestro de aplicaciones YARN debe esperar a que se inicialice SparkContext.	1.3.0
`spark.yarn.submit.file.replication`	La replicación HDFS predeterminada (normalmente 3)	Nivel de replicación de HDFS para los archivos cargados en HDFS para la aplicación. Estos incluyen elementos como el jar de Spark, el jar de la aplicación y cualquier archivo o archivo de caché distribuido.	0.8.1

Nombre de la propiedad	Por defecto	Significado	Desde la versión
spark.yarn.stagingDir	Directorio de inicio del usuario actual en el sistema de archivos	Directorio de almacenamiento temporal utilizado al enviar solicitudes.	2.0.0
spark.yarn.preserve.staging.files	false	Configure truepara conservar los archivos almacenados (jar de Spark, jar de la aplicación, archivos de caché distribuidos) al final del trabajo en lugar de eliminarlos.	1.1.0
spark.yarn.scheduler.heartbeat.interval-ms	3000	El intervalo en ms en el que el maestro de la aplicación Spark realiza un latido en el ResourceManager de YARN. El valor está limitado a la mitad del valor de la configuración de YARN para el intervalo de vencimiento, es decir yarn.am.liveness-monitor.expiry-interval-ms, .	0.8.1
spark.yarn.scheduler.initial-allocation.interval	200ms	El intervalo inicial en el que el maestro de la aplicación Spark envía un mensaje de alerta al administrador de recursos de YARN cuando hay solicitudes de asignación de contenedores pendientes. No debe ser mayor que spark.yarn.scheduler.heartbeat.interval-ms. El intervalo de asignación se duplicará en los mensajes de alerta sucesivos si aún existen contenedores pendientes, hasta que spark.yarn.scheduler.heartbeat.interval-msse alcance .	1.4.0
spark.yarn.historyServer.address	(ninguno)	La dirección del servidor de historial de Spark, p. ej host.com:18080. La dirección no debe contener un esquema (http://). El valor predeterminado es que no se establece, ya que el servidor de historial es un servicio opcional. Esta dirección se proporciona al ResourceManager de YARN cuando la aplicación Spark termina de vincular la aplicación desde la interfaz de usuario del ResourceManager a la interfaz de usuario del servidor de historial de Spark. Para esta propiedad, las propiedades de YARN se pueden usar como variables y Spark las sustituye en el tiempo de ejecución. Por ejemplo, si el servidor de historial de Spark se ejecuta en el mismo nodo que el ResourceManager de YARN, se puede establecer en ${hadoopconf-yarn.resourcemanager.hostname}:18080.	1.0.0
spark.yarn.dist.archives	(ninguno)	Lista separada por comas de archivos que se extraerán en el directorio de trabajo de cada ejecutor.	1.0.0
spark.yarn.dist.files	(ninguno)	Lista separada por comas de archivos que se colocarán en el directorio de trabajo de cada ejecutor.	1.0.0
spark.yarn.dist.jars	(ninguno)	Lista separada por comas de archivos jar que se colocarán en el directorio de trabajo de cada ejecutor.	2.0.0
spark.yarn.dist.forceDownloadSchemes	(none)	Lista separada por comas de esquemas para los cuales se descargarán recursos al disco local antes de agregarlos a la memoria caché distribuida de YARN. Para usar en casos en los que el servicio	2.3.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		YARN no admite esquemas que sean compatibles con Spark, como http, https y ftp, o archivos jar que deban estar en la ruta de clase del cliente YARN local. El comodín "*" indica que se descargarán recursos para todos los esquemas.	
`spark.executor.instances`	2	La cantidad de ejecutores para la asignación estática. Con `spark.dynamicAllocation.enabled`, el conjunto inicial de ejecutores será al menos de este tamaño.	1.0.0
`spark.yarn.am.memoryOverhead`	Memoria AM * 0,10, con un mínimo de 384	Igual que `spark.driver.memoryOverhead`, pero para el YARN Application Master en modo cliente.	1.3.0
`spark.yarn.queue`	`default`	El nombre de la cola YARN a la que se envía la aplicación.	1.0.0
`spark.yarn.jars`	(ninguno)	Lista de bibliotecas que contienen código Spark para distribuir a los contenedores YARN. De manera predeterminada, Spark en YARN usará archivos jar de Spark instalados localmente, pero los archivos jar de Spark también pueden estar en una ubicación legible para todo el mundo en HDFS. Esto permite que YARN los almacene en caché en los nodos para que no sea necesario distribuirlos cada vez que se ejecuta una aplicación. Para apuntar a archivos jar en HDFS, por ejemplo, configure esta configuración en `hdfs:///some/path`. Se permiten los globs.	2.0.0
`spark.yarn.archive`	(ninguno)	Un archivo que contiene los archivos jar de Spark necesarios para su distribución a la memoria caché de YARN. Si se configura, esta configuración reemplaza `spark.yarn.jars` y el archivo se utiliza en todos los contenedores de la aplicación. El archivo debe contener archivos jar en su directorio raíz. Al igual que con la opción anterior, el archivo también se puede alojar en HDFS para acelerar la distribución de archivos.	2.0.0
`spark.yarn.appMasterEnv.[EnvironmentVariableName]`	(ninguno)	Agregue la variable de entorno especificada por `EnvironmentVariableName`al proceso Application Master iniciado en YARN. El usuario puede especificar varias de estas y configurar varias variables de entorno. En `modo cluster` , esto controla el entorno del controlador Spark y en `modo cliente` , solo controla el entorno del iniciador del ejecutor.	1.1.0
`spark.yarn.containerLauncherMaxThreads`	25	La cantidad máxima de subprocesos que se utilizarán en el maestro de aplicaciones YARN para iniciar contenedores ejecutores.	1.2.0
`spark.yarn.am.extraJavaOptions`	(ninguno)	Una cadena de opciones JVM adicionales para pasar al maestro de aplicaciones YARN en modo cliente. En modo clúster, utilice `spark.driver.extraJavaOptions`en su lugar. Tenga en cuenta que es ilegal establecer configuraciones de tamaño de montón máximo (-Xmx) con esta opción. Las configuraciones de	1.3.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		tamaño de montón máximo se pueden establecer con`spark.yarn.am.memory`	
`spark.yarn.am.extraLibraryPath`	(ninguno)	Establezca una ruta de biblioteca especial para usar al iniciar YARN Application Master en modo cliente.	1.4.0
`spark.yarn.populateHadoopClasspath`	Para `with-hadoop`la distribución de Spark, esto se establece como falso; para `no-hadoop`la distribución, esto se establece como verdadero.	Si se debe completar la ruta de clase de Hadoop desde `yarn.application.classpath`y `mapreduce.application.classpath` Tenga en cuenta que si esto se establece en `false`, requiere una `with-Hadoop` distribución de Spark que incluya el tiempo de ejecución de Hadoop o el usuario debe proporcionar una instalación de Hadoop por separado.	2.4.6
`spark.yarn.maxAppAttempts`	`yarn.resourcemanager.am.max-attempts`en HILO	El número máximo de intentos que se realizarán para enviar la solicitud. No debe ser mayor que el número global de intentos máximos en la configuración de YARN.	1.3.0
`spark.yarn.am.attemptFailuresValidityInterval`	(ninguno)	Define el intervalo de validez para el seguimiento de fallas de AM. Si el AM ha estado en funcionamiento durante al menos el intervalo definido, se restablecerá el recuento de fallas de AM. Esta función no está habilitada si no está configurada.	1.6.0
`spark.yarn.am.clientModeTreatDisconnectAsFailed`	FALSO	Trate las desconexiones incorrectas de yarn-client como fallas. En el modo yarn-client, normalmente la aplicación siempre terminará con un estado final de ÉXITO porque en algunos casos, no es posible saber si la aplicación fue finalizada intencionalmente por el usuario o si hubo un error real. Esta configuración cambia ese comportamiento de modo que si el Application Master se desconecta del controlador de manera incorrecta (es decir, sin el protocolo de cierre de sesión adecuado), la aplicación terminará con un estado final de FAILED. Esto permitirá que el llamador decida si realmente fue una falla. Tenga en cuenta que si esta configuración está establecida y el usuario simplemente termina la aplicación cliente de manera incorrecta, puede mostrar un estado de FAILED cuando en realidad no fue FAILED.	3.3.0
`spark.yarn.am.clientModeExitOnError`	FALSO	En el modo de cliente de hilo, cuando esto es verdadero, si el controlador recibió un informe de aplicación con el estado final de KILLED o FAILED, el controlador detendrá el SparkContext correspondiente y saldrá del programa con el código 1. Tenga en cuenta que, si esto es verdadero y se llama desde otra aplicación, también finalizará la aplicación principal.	3.3.0
`spark.yarn.am.tokenConfRegex`	(ninguno)	El valor de esta configuración es una expresión regular que se utiliza para buscar una lista de entradas de configuración del archivo de configuración del trabajo (por ejemplo, hdfs-site.xml) y enviarlas a RM, que las utiliza al renovar los tokens de delegación. Un caso de uso típico de esta función es admitir tokens de	3.3.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		delegación en un entorno donde un clúster YARN necesita comunicarse con varios clústeres HDFS de bajada, donde el RM YARN puede no tener configuraciones (por ejemplo, dfs.nameservices, dfs.ha.namenodes.*, dfs.namenode.rpc-address.*) para conectarse a estos clústeres. En este escenario, los usuarios de Spark pueden especificar el valor de configuración para analizar estas configuraciones HDFS de los archivos de configuración locales del trabajo. Esta configuración es muy similar a . Consulte YARN-5910 para obtener más detalles. `^dfs.nameservices`|dfs.namenode.rpc−address.∗|`dfs.namenode.rpc−address.∗|^dfs.ha.namenodes.*$mapreduce.job.send-token-conf`	
`spark.yarn.submit.waitAppCompletion`	`true`	En el modo de clúster de YARN, controla si el cliente espera para salir hasta que se complete la aplicación. Si se configura en `true`, el proceso del cliente permanecerá activo informando el estado de la aplicación. De lo contrario, el proceso del cliente saldrá después del envío.	1.4.0
`spark.yarn.am.nodeLabelExpression`	(ninguno)	Una expresión de etiqueta de nodo de YARN que restringe el conjunto de nodos en los que se programará AM. Solo las versiones de YARN mayores o iguales a 2.6 admiten expresiones de etiqueta de nodo, por lo que, cuando se ejecuta en versiones anteriores, se ignorará esta propiedad.	1.6.0
`spark.yarn.executor.nodeLabelExpression`	(ninguno)	Una expresión de etiqueta de nodo de YARN que restringe el conjunto de nodos en los que se programarán los ejecutores. Solo las versiones de YARN mayores o iguales a 2.6 admiten expresiones de etiqueta de nodo, por lo que, cuando se ejecuta en versiones anteriores, se ignorará esta propiedad.	1.4.0
`spark.yarn.tags`	(ninguno)	Lista de cadenas separadas por comas que se pasarán como etiquetas de aplicación YARN que aparecen en YARN ApplicationReports y que se pueden usar para filtrar al consultar aplicaciones YARN.	1.5.0
`spark.yarn.priority`	(ninguno)	Prioridad de aplicaciones para YARN para definir la política de ordenación de aplicaciones pendientes. Las que tengan un valor entero más alto tienen más posibilidades de activarse. Actualmente, YARN solo admite la prioridad de aplicaciones cuando se utiliza la política de ordenación FIFO.	3.0.0
`spark.yarn.config.gatewayPath`	(ninguno)	Una ruta válida en el host de la puerta de enlace (el host donde se inicia una aplicación Spark), pero que puede diferir de las rutas del mismo recurso en otros nodos del clúster. Junto con `spark.yarn.config.replacementPath`, se utiliza para admitir clústeres con configuraciones heterogéneas, de modo que Spark pueda iniciar correctamente los procesos remotos. La ruta de reemplazo normalmente contendrá una referencia a alguna variable de entorno exportada	1.5.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		por YARN (y, por lo tanto, visible para los contenedores de Spark). Por ejemplo, si el nodo de puerta de enlace tiene bibliotecas Hadoop instaladas en `/disk1/hadoop`, y la ubicación de la instalación de Hadoop es exportada por YARN como `HADOOP_HOME` variable de entorno, establecer este valor en `/disk1/hadoop`y la ruta de reemplazo en `$HADOOP_HOME`garantizará que las rutas utilizadas para iniciar procesos remotos hagan referencia correctamente a la configuración local de YARN.	
`spark.yarn.config.replacementPath`	(ninguno)	Ver `spark.yarn.config.gatewayPath`.	1.5.0
`spark.yarn.rolledLog.includePattern`	(ninguno)	Expresión regular de Java para filtrar los archivos de registro que coinciden con el patrón de inclusión definido y esos archivos de registro se agregarán de manera continua. Esto se usará con la agregación continua de registros de YARN; para habilitar esta función en el lado de YARN `yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds`, se debe configurar en yarn-site.xml. El apéndice Spark log4j debe cambiarse para usar FileAppender u otro apéndice que pueda manejar los archivos que se eliminan mientras se ejecuta. Según el nombre de archivo configurado en la configuración de log4j (como spark.log), el usuario debe configurar la expresión regular (spark*) para incluir todos los archivos de registro que se deben agregar.	2.0.0
`spark.yarn.rolledLog.excludePattern`	(ninguno)	Expresión regular de Java para filtrar los archivos de registro que coinciden con el patrón de exclusión definido y esos archivos de registro no se agregarán de forma continua. Si el nombre del archivo de registro coincide con el patrón de inclusión y exclusión, este archivo se excluirá eventualmente.	2.0.0
`spark.yarn.executor.launch.excludeOnFailure.enabled`	FALSO	Marca para habilitar la exclusión de nodos que tengan problemas de asignación de recursos de YARN. El límite de error para la exclusión se puede configurar mediante `spark.excludeOnFailure.application.maxFailedExecutorsPerNode`.	2.4.0
`spark.yarn.exclude.nodes`	(ninguno)	Lista separada por comas de nombres de nodos YARN que están excluidos de la asignación de recursos.	3.0.0
`spark.yarn.metrics.namespace`	(ninguno)	El espacio de nombres raíz para los informes de métricas de AM. Si no se configura, se utiliza el ID de la aplicación YARN.	2.4.0
`spark.yarn.report.interval`	`1s`	Intervalo entre informes del estado actual del trabajo de Spark en modo de clúster.	0.9.0
`spark.yarn.report.loggingFrequency`	`30`	Número máximo de informes de aplicaciones procesados hasta que se registre el siguiente estado de la aplicación. Si hay un cambio de estado, el	3.5.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		estado de la aplicación se registrará independientemente del número de informes de aplicaciones procesados.	
spark.yarn.clientLaunchMonitorInterval	1s	Intervalo entre solicitudes de estado del modo cliente AM al iniciar la aplicación.	2.3.0
spark.yarn.includeDriverLogsLink	false	En el modo de clúster, si el informe de la aplicación cliente incluye vínculos a los registros del contenedor del controlador. Esto requiere sondear la API REST de ResourceManager, por lo que coloca una carga adicional en el RM.	3.1.0
spark.yarn.unmanagedAM.enabled	false	En modo cliente, si se debe iniciar el servicio Application Master como parte del cliente mediante am no administrado.	3.0.0
spark.yarn.shuffle.server.recovery.disabled	FALSO	Establezca esta opción en verdadero para las aplicaciones que tienen requisitos de seguridad más altos y prefieren que su secreto no se guarde en la base de datos. Los datos de reproducción aleatoria de dichas aplicaciones no se recuperarán después de que se reinicie el servicio de reproducción aleatoria externa.	3.5.0

Patrones disponibles para la URL del registro del ejecutor personalizado de SHS

Patrón	Significado
{{ESQUEMA_HTTP}}	`http://`o `https://`según la política HTTP de YARN. (Configurado mediante `yarn.http.policy`)
{{Host_NM}}	El "host" del nodo donde se ejecutó el contenedor.
{{PUERTO_NM}}	El "puerto" del administrador de nodos donde se ejecutó el contenedor.
{{PUERTO HTTP_NM}}	El "puerto" del servidor http del administrador de nodos donde se ejecutó el contenedor.
{{DIRECCIÓN HTTP_NM}}	URI HTTP del nodo en el que está asignado el contenedor.
{{ID_DE_CLÚSTER}}	El ID del clúster del Administrador de recursos. (Configurado mediante `yarn.resourcemanager.cluster-id`)
{{ID_DE_CONTENEDOR}}	El ID del contenedor.
{{USUARIO}}	`SPARK_USER`en el entorno del sistema.
{{NOMBRE DEL ARCHIVO}}	`stdout`, `stderr`.

Por ejemplo, supongamos que desea apuntar el enlace de la URL del registro al servidor de historial de trabajos directamente en lugar de dejar que el servidor http de NodeManager lo redirija, puede configurarlo `spark.history.custom.executor.log.url`de la siguiente manera:

```
{{HTTP_SCHEME}}<JHS_HOST>:<JHS_PORT>/jobhistory/logs/{{NM_HOST}}:{{NM_POR
T}}/{{CONTAINER_ID}}/{{CONTAINER_ID}}/{{USER}}/{{FILE_NAME}}?start=-4096
```

NOTA: es necesario reemplazar `<JHS_HOST>`y `<JHS_PORT>`con el valor real.

Descripción general de la configuración y asignación de recursos

Asegúrese de haber leído la sección Descripción general de la configuración y programación de recursos personalizados en la página de configuración . Esta sección solo habla sobre los aspectos específicos de YARN de la programación de recursos.

YARN debe configurarse para admitir cualquier recurso que el usuario desee utilizar con Spark. La programación de recursos en YARN se agregó en YARN 3.1.0. Consulte la documentación de YARN para obtener más información sobre cómo configurar recursos y configurar correctamente el aislamiento. Lo ideal es que los recursos se configuren aislados para que un ejecutor solo pueda ver los recursos que se le asignaron. Si no tiene habilitado el aislamiento, el usuario es responsable de crear un script de descubrimiento que garantice que el recurso no se comparta entre ejecutores.

YARN admite tipos de recursos definidos por el usuario, pero tiene tipos integrados para GPU (`yarn.io/gpu`) y FPGA (`yarn.io/fpga`). Por ese motivo, si utiliza cualquiera de esos recursos, Spark puede traducir su solicitud de recursos de Spark en recursos de YARN y usted solo tiene que especificar las `spark.{driver/executor}.resource.`configuraciones. Tenga en cuenta que, si utiliza un tipo de recurso personalizado para GPU o FPGA con YARN, puede cambiar la asignación de Spark utilizando `spark.yarn.resourceGpuDeviceName`y `spark.yarn.resourceFpgaDeviceName`. Si utiliza un recurso distinto de FPGA o GPU, el usuario es responsable de especificar las configuraciones tanto para YARN (`spark.yarn.{driver/executor}.resource.`) como para Spark (`spark.{driver/executor}.resource.`).

Por ejemplo, el usuario desea solicitar 2 GPU para cada ejecutor. El usuario puede simplemente especificar `spark.executor.resource.gpu.amount=2`y Spark se encargará de solicitar `yarn.io/gpu` el tipo de recurso de YARN.

Si el usuario tiene un recurso YARN definido por el usuario, lo llamaremos `acceleratorX`y luego el usuario debe especificar `spark.yarn.executor.resource.acceleratorX.amount=2`y `spark.executor.resource.acceleratorX.amount=2`.

YARN no le indica a Spark las direcciones de los recursos asignados a cada contenedor. Por ese motivo, el usuario debe especificar un script de descubrimiento que el ejecutor ejecuta al inicio para descubrir qué recursos están disponibles para ese ejecutor. Puede encontrar un script de ejemplo en `examples/src/main/scripts/getGpusResources.sh`. El script debe tener permisos de ejecución establecidos y el usuario debe configurar los permisos para no permitir que usuarios malintencionados lo modifiquen. El script debe escribir en STDOUT una cadena JSON en el formato de la clase ResourceInformation. Esta tiene el nombre del recurso y una matriz de direcciones de recursos disponibles solo para ese ejecutor.

Descripción general de la programación a nivel de etapa

La programación a nivel de etapa es compatible con YARN:

- Cuando la asignación dinámica está deshabilitada: permite a los usuarios especificar diferentes requisitos de recursos de tareas en el nivel de etapa y utilizará los mismos ejecutores solicitados al inicio.
- Cuando la asignación dinámica está habilitada: permite a los usuarios especificar los requisitos de recursos de tareas y ejecutores en el nivel de etapa y solicitará ejecutores adicionales.

Una cosa a tener en cuenta que es específica de YARN es que cada ResourceProfile requiere una prioridad de contenedor diferente en YARN. La asignación es simplemente que el id de ResourceProfile se convierte en la prioridad, en YARN los números más bajos tienen mayor prioridad. Esto significa que los perfiles creados anteriormente tendrán una mayor prioridad en YARN. Normalmente esto no importará ya que Spark finaliza una etapa antes de comenzar otra, el único caso en el que esto podría tener un efecto es en un escenario de tipo servidor de trabajo, por lo que es algo a tener en cuenta. Tenga en cuenta que hay una diferencia en la forma en que se manejan los recursos personalizados entre el perfil predeterminado base y los ResourceProfiles personalizados. Para permitir que el usuario solicite contenedores YARN con recursos adicionales sin que Spark los programe, el usuario puede especificar recursos a través de la `spark.yarn.executor.resource.`configuración. Sin embargo, esas configuraciones solo se usan en el perfil predeterminado base y no se propagan a ningún otro ResourceProfiles personalizado. Esto se debe a que no habría forma de eliminarlos si quisiera que una etapa no los tuviera. Esto da como resultado que su perfil predeterminado obtenga recursos personalizados definidos en `spark.yarn.executor.resource.`recursos definidos por Spark de GPU o FPGA. Spark convierte los recursos de GPU y FPGA en los tipos integrados de YARN `yarn.io/gpu`) y `yarn.io/fpga`, pero no conoce la asignación de ningún otro recurso. Cualquier otro recurso personalizado de Spark no se propaga a YARN para el perfil predeterminado. Por lo tanto, si desea que Spark programe en función de un recurso personalizado y que se lo solicite a YARN, debe especificarlo en las configuraciones de YARN (`spark.yarn.{driver/executor}.resource.`) y Spark (`spark.{driver/executor}.resource.`). Deje la configuración de Spark desactivada si solo desea contenedores de YARN con los recursos adicionales, pero que Spark no programe su uso. Ahora bien, para los ResourceProfiles personalizados, actualmente no tiene una forma de especificar solo recursos de YARN sin que Spark programe a partir de ellos. Esto significa que para los ResourceProfiles personalizados propagamos todos los recursos definidos en el ResourceProfile a YARN. También seguimos convirtiendo GPU y FPGA a los tipos integrados de YARN. Esto requiere que el nombre de cualquier recurso personalizado que especifique coincida con lo que está definido en YARN.

Notas importantes

- El hecho de que las solicitudes principales se respeten en las decisiones de programación depende del programador que se utilice y de cómo esté configurado.

- En `modo cluster`, los directorios locales utilizados por los ejecutores de Spark y el controlador de Spark serán los directorios locales configurados para YARN (Hadoop YARN config `yarn.nodemanager.local-dirs`). Si el usuario especifica `spark.local.dir`, se ignorará. En `modo cliente`, los ejecutores de Spark utilizarán los directorios locales configurados para YARN mientras que el controlador de Spark utilizará los definidos en `spark.local.dir`. Esto se debe a que el controlador de Spark no se ejecuta en el clúster de YARN en `modo cliente`, solo lo hacen los ejecutores de Spark.
- Las opciones `--files`y `--archives`permiten especificar nombres de archivos con el símbolo # de forma similar a Hadoop. Por ejemplo, puede especificar: `--files localtest.txt#appSees.txt`y esto cargará el archivo que haya nombrado localmente `localtest.txt`en HDFS, pero se vinculará a este mediante el nombre `appSees.txt`y su aplicación debe usar el nombre como `appSees.txt`para hacer referencia a él cuando se ejecute en YARN.
- La `–jars` opción permite que la `SparkContext.addJar` función funcione si la usa con archivos locales y se ejecuta en `clustermodo`. No es necesario usarla si la usa con archivos HDFS, HTTP, HTTPS o FTP.

Kerberos

La compatibilidad estándar con Kerberos en Spark se cubre en la página de Seguridad .

En el modo YARN, al acceder a los sistemas de archivos Hadoop, además del sistema de archivos predeterminado en la configuración de Hadoop, Spark también obtendrá automáticamente tokens de delegación para el servicio que aloja el directorio de prueba de la aplicación Spark.

Configuración de Kerberos específica de YARN

Nombre de la propiedad	Por defecto	Significado	Desde la versión
`spark.kerberos.keytab`	(ninguno)	Ruta completa al archivo que contiene la tabla de claves para el principal especificado anteriormente. Esta tabla de claves se copiará al nodo que ejecuta el maestro de aplicaciones YARN a través de la caché distribuida de YARN y se utilizará para renovar los tickets de inicio de sesión y los tokens de delegación periódicamente. Equivale al `–keytab` argumento de la línea de comandos. (Funciona también con el maestro "local").	3.0.0
`spark.kerberos.principal`	(ninguno)	Principal que se utilizará para iniciar sesión en KDC mientras se ejecuta en clústeres seguros. Equivalente al `--principal`argumento de la línea de comandos. (También funciona con el maestro "local").	3.0.0
`spark.yarn.kerberos.relogin.period`	1 metro	Con qué frecuencia se debe comprobar si se debe renovar el TGT de Kerberos. Este valor debe establecerse en un valor más corto que el período de renovación del TGT (o la duración del TGT si la renovación del TGT no está	2.3.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		habilitada). El valor predeterminado debería ser suficiente para la mayoría de las implementaciones.	
`spark.yarn.kerberos.renewal.excludeHadoopFileSystems`	(ninguno)	Una lista separada por comas de los sistemas de archivos Hadoop cuyos hosts se excluirán de la renovación del token de delegación en el programador de recursos. Por ejemplo, `spark.yarn.kerberos.renewal.excludeHadoopFileSystems=hdfs://nn1.com:8032, hdfs://nn2.com:8032`. Se sabe que esto funciona en YARN por ahora, por lo que YARN Resource Manager no renovará los tokens para la aplicación. Tenga en cuenta que, como el programador de recursos no renueva el token, cualquier aplicación que se ejecute más allá de la fecha de vencimiento del token original e intente usar ese token probablemente fallará.	3.2.0

Solución de problemas de Kerberos

La depuración de problemas de Hadoop/Kerberos puede resultar “difícil”. Una técnica útil es habilitar el registro adicional de operaciones de Kerberos en Hadoop configurando la `HADOOP_JAAS_DEBUG` variable de entorno.

```
export HADOOP_JAAS_DEBUG=true
```

Las clases JDK se pueden configurar para habilitar el registro adicional de su autenticación Kerberos y SPNEGO/REST a través de las propiedades del sistema `sun.security.krb5.debug` y`sun.security.spnego.debug=true`

```
-Dsun.security.krb5.debug=true -Dsun.security.spnego.debug=true
```

Todas estas opciones se pueden habilitar en el Maestro de aplicaciones:

```
spark.yarn.appMasterEnv.HADOOP_JAAS_DEBUG true

spark.yarn.am.extraJavaOptions -Dsun.security.krb5.debug=true -
Dsun.security.spnego.debug=true
```

Finalmente, si el nivel de registro `org.apache.spark.deploy.yarn.Client` se establece en `DEBUG`, el registro incluirá una lista de todos los tokens obtenidos y sus detalles de vencimiento.

Configuración del servicio Shuffle externo

Para iniciar el servicio Spark Shuffle en cada uno `NodeManager` de los clústeres YARN, siga estas instrucciones:

1. Cree Spark con el perfil YARN . Omita este paso si está utilizando una distribución preempaquetada.
2. Localice el `spark-<version>-yarn-shuffle.jar`. Debería estar debajo `$SPARK_HOME/common/network-yarn/target/scala-<version>`si está compilando Spark usted mismo y debajo `yarn` si está usando una distribución.
3. Agregue este jar al classpath de todos `NodeManagers` en su clúster.
4. En `yarn-site.xml`cada nodo, agregue `spark_shuffle`a `yarn.nodemanager.aux-services`y luego configúrelo `yarn.nodemanager.aux-services.spark_shuffle.class`en `org.apache.spark.network.yarn.YarnShuffleService`.
5. Aumente `NodeManager's`el tamaño del montón estableciendo `YARN_HEAPSIZE`(1000 por defecto) `etc/hadoop/yarn-env.sh` para evitar problemas de recolección de basura durante la mezcla.
6. Reinicie todos `NodeManagers` en su clúster.

Las siguientes opciones de configuración adicionales están disponibles cuando el servicio aleatorio se ejecuta en YARN:

Nombre de la propiedad	Por defecto	Significado	Desde la versión
`spark.yarn.shuffle.stopOnFailure`	`false`	Si se debe detener NodeManager cuando se produce un error en la inicialización del servicio Spark Shuffle. Esto evita errores de aplicación provocados por la ejecución de contenedores en NodeManagers donde no se está ejecutando el servicio Spark Shuffle.	2.1.0
`spark.yarn.shuffle.service.metrics.namespace`	`sparkShuffleService`	El espacio de nombres que se utilizará al emitir métricas de servicio aleatorio en el sistema Hadoop metrics2 del NodeManager.	3.2.0
`spark.yarn.shuffle.service.logs.namespace`	`(not set)`	Un espacio de nombres que se agregará al nombre de la clase al formar el nombre del registrador que se usará para emitir registros desde el servicio de reproducción aleatoria de YARN, como `org.apache.spark.network.yarn.YarnShuffleService.logsNamespaceValue`. Dado que algunos marcos de registro pueden esperar que el nombre del registrador se parezca a un nombre de clase, generalmente se recomienda proporcionar un valor que sea un nombre de clase o paquete Java válido y que no incluya espacios.	3.3.0
`spark.shuffle.service.db.backend`	NIVELDB	Cuando se habilita el reinicio con conservación de trabajo en YARN, esto se utiliza para especificar el almacenamiento en base de disco utilizado en el almacenamiento de estado del servicio aleatorio, admite `LEVELDB` y `ROCKSDB` con `LEVELDB` como valor	3.4.0

Nombre de la propiedad	Por defecto	Significado	Desde la versión
		predeterminado. El almacenamiento de datos original en `LevelDB/RocksDB` no se convertirá automáticamente a otro tipo de almacenamiento ahora. El almacenamiento de datos original se conservará y se creará el nuevo tipo de almacenamiento de datos al cambiar los tipos de almacenamiento.	

Tenga en cuenta que las instrucciones anteriores suponen que se ha utilizado el nombre de servicio de reproducción aleatoria predeterminado `spark_shuffle`, Es posible utilizar cualquier nombre aquí, pero los valores utilizados en las configuraciones de YARN NodeManager deben coincidir con el valor de `spark.shuffle.service.name` en la aplicación Spark.

El servicio Shuffle, de manera predeterminada, tomará todas sus configuraciones de la configuración de Hadoop utilizada por NodeManager (por ejemplo, `yarn-site.xml`). Sin embargo, también es posible configurar el servicio Shuffle de manera independiente utilizando un archivo llamado `spark-shuffle-site.xml` que debe ubicarse en la ruta de clase del servicio Shuffle (que, de manera predeterminada, se comparte con la ruta de clase de NodeManager). El servicio Shuffle tratará esto como un recurso de configuración de Hadoop estándar y lo superpondrá sobre la configuración de NodeManager.

Cómo iniciar su aplicación con Apache Oozie

Apache Oozie puede iniciar aplicaciones Spark como parte de un flujo de trabajo. En un clúster seguro, la aplicación iniciada necesitará los tokens pertinentes para acceder a los servicios del clúster. Si Spark se inicia con un keytab, esto es automático. Sin embargo, si Spark se va a iniciar sin un keytab, la responsabilidad de configurar la seguridad debe ser delegada a Oozie.

Los detalles sobre cómo configurar Oozie para clústeres seguros y obtener credenciales para un trabajo se pueden encontrar en el sitio web de Oozie en la sección “Autenticación” de la documentación de la versión específica.

Para las aplicaciones Spark, el flujo de trabajo de Oozie debe configurarse para que Oozie solicite todos los tokens que necesita la aplicación, incluidos:

- El administrador de recursos de YARN.
- El sistema de archivos Hadoop local.
- Cualquier sistema de archivos Hadoop remoto utilizado como origen o destino de E/S.
- Colmena —si se utiliza.
- HBase —si se utiliza.
- El servidor de línea de tiempo de YARN, si la aplicación interactúa con este.

Para evitar que Spark intente (y luego no pueda) obtener tokens de Hive, HBase y HDFS remotos, la configuración de Spark debe configurarse para deshabilitar la recopilación de tokens para los servicios.

La configuración de Spark debe incluir las líneas:

```
spark.security.credentials.hive.enabled   false
spark.security.credentials.hbase.enabled  false
```

La opción de configuración `spark.kerberos.access.hadoopFileSystems` debe estar desconfigurada.

Uso del servidor de historial de Spark para reemplazar la interfaz web de Spark

Es posible utilizar la página de la aplicación Spark History Server como URL de seguimiento para las aplicaciones en ejecución cuando la interfaz de usuario de la aplicación está deshabilitada. Esto puede ser conveniente en clústeres seguros o para reducir el uso de memoria del controlador Spark. Para configurar el seguimiento a través de Spark History Server, haga lo siguiente:

- En el lado de la aplicación, configúrelo `spark.yarn.historyServer.allowTracking=true`en la configuración de Spark. Esto le indicará a Spark que use la URL del servidor de historial como URL de seguimiento si la interfaz de usuario de la aplicación está deshabilitada.
- En el servidor de historial de Spark, agregue `org.apache.spark.deploy.yarn.YarnProxyRedirectFilter` a la lista de filtros en la `spark.ui.filters`configuración.

Tenga en cuenta que es posible que la información del servidor histórico no esté actualizada con el estado de la aplicación.

Ejecución de varias versiones del servicio Spark Shuffle

Tenga en cuenta que esta sección solo se aplica cuando se ejecuta en versiones de YARN >= 2.9.0.

En algunos casos, puede ser conveniente ejecutar varias instancias del servicio Spark Shuffle que utilicen diferentes versiones de Spark. Esto puede ser útil, por ejemplo, al ejecutar un clúster YARN con una carga de trabajo mixta de aplicaciones que ejecutan varias versiones de Spark, ya que una versión determinada del servicio Shuffle no siempre es compatible con otras versiones de Spark. Las versiones de YARN desde la 2.9.0 admiten la capacidad de ejecutar servicios Shuffle dentro de un cargador de clases aislado (consulte YARN-4577), lo que significa que varias versiones de Spark pueden coexistir dentro de un único NodeManager. Las opciones `yarn.nodemanager.aux-services.<service-name>.classpathy`, a partir de YARN 2.10.2/3.1.1/3.2.0, `yarn.nodemanager.aux-services.<service-name>.remote-classpath` se pueden utilizar para configurar esto. Tenga en cuenta que YARN 3.3.0/3.3.1 tiene un problema que requiere una configuración `yarn.nodemanager.aux-services.<service-name>.system-classes` como solución alternativa. Consulte YARN-11053 para obtener más detalles. Además de configurar rutas de clases independientes, es necesario garantizar que las dos versiones se anuncien en puertos diferentes. Esto se puede lograr

utilizando el `spark-shuffle-site.xml` archivo descrito anteriormente. Por ejemplo, puede tener una configuración como:

```
  yarn.nodemanager.aux-services = spark_shuffle_x,spark_shuffle_y
  yarn.nodemanager.aux-services.spark_shuffle_x.classpath = /path/to/spark-x-path/fat.jar:/path/to/spark-x-config
  yarn.nodemanager.aux-services.spark_shuffle_y.classpath = /path/to/spark-y-path/fat.jar:/path/to/spark-y-config
```

O

```
  yarn.nodemanager.aux-services = spark_shuffle_x,spark_shuffle_y
  yarn.nodemanager.aux-services.spark_shuffle_x.classpath = /path/to/spark-x-path/*:/path/to/spark-x-config
  yarn.nodemanager.aux-services.spark_shuffle_y.classpath = /path/to/spark-y-path/*:/path/to/spark-y-config
```

Cada uno de los dos `spark-*-config` directorios contiene un archivo, `spark-shuffle-site.xml`. Se trata de archivos XML en formato de configuración de Hadoop que contienen algunas configuraciones para ajustar el número de puerto y el prefijo del nombre de métricas utilizado:

```
<configuration>
  <property>
    <name>spark.shuffle.service.port</name>
    <value>7001</value>
  </property>
  <property>
    <name>spark.yarn.shuffle.service.metrics.namespace</name>
    <value>sparkShuffleServiceX</value>
  </property>
</configuration>
```

Los valores deben ser diferentes para los dos servicios diferentes.

Luego en la configuración de las aplicaciones Spark se debe configurar:

```
  spark.shuffle.service.name = spark_shuffle_x
  spark.shuffle.service.port = 7001
```

y uno debe configurarse con:

```
  spark.shuffle.service.name = spark_shuffle_y
  spark.shuffle.service.port = <other value>
```