BIG DATA

YARN MAPREDUCE

EDUARD LARA

1. INTRODUCCIÓN

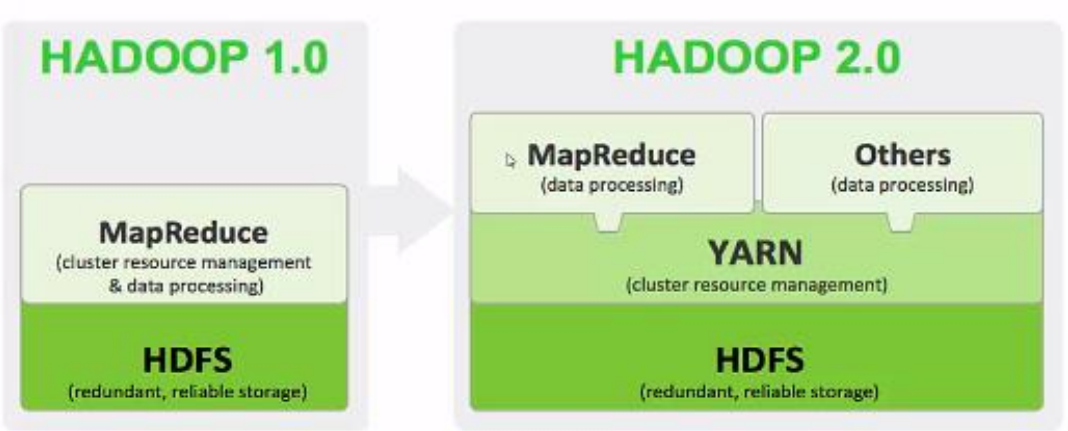
- Hasta ahora hemos visto la parte de gestión de datos de Hadoop: HDFS, creación y subida de ficheros, etc.
- Ahora veremos la parte de los procesos de Hadoop
- Esta compuesto de dos versiones distintas:
  - MAPREDUCE V1, la versión 1 de hadoop. No es la más recomendable a dia de hoy.
  - MAPREDUCE V2, mas conocida como YARN. Es la versión que se normalmente se utiliza ya
- **NOTA**: MapReduce V1 sigue utilizándose con YARN.
- Veremos que MapReduce V2 tiene la V1 de MapReduce como un componente más.

1. INTRODUCCION

- MapReduce V1
  - Estaba pensada para procesos Batch.
  - Se encarga tanto de la parte de procesamiento de los datos como de la parte de la gestión del cluster
- MapReduce V2 - Yarn
  - Admite otra tipo de productos y procesos que no sean sólo batch
  - Tiene procesos distintos para el proceso de datos y para la gestion del clúster.
  - Yarn ha separado la parte de la gestión de la parte del proceso consiguiendo un mejor rendimiento.

1. INTRODUCCIÓN

- Diferencias entre V1 y V2.



- En Hadoop V1 MapReduce se encargaba de toda la gestión del cluster y del procesamiento de los datos
- En Hadoop V2, Yarn es el gestor del cluster. MapReduce ha pasado de ser el core central a ser un componente mas de Hadoop. En la parte superior podemos tener procesos MapReduce (de la V1) o procesos de otro tipo.

1. INTRODUCCIÓN

MapReduce V1

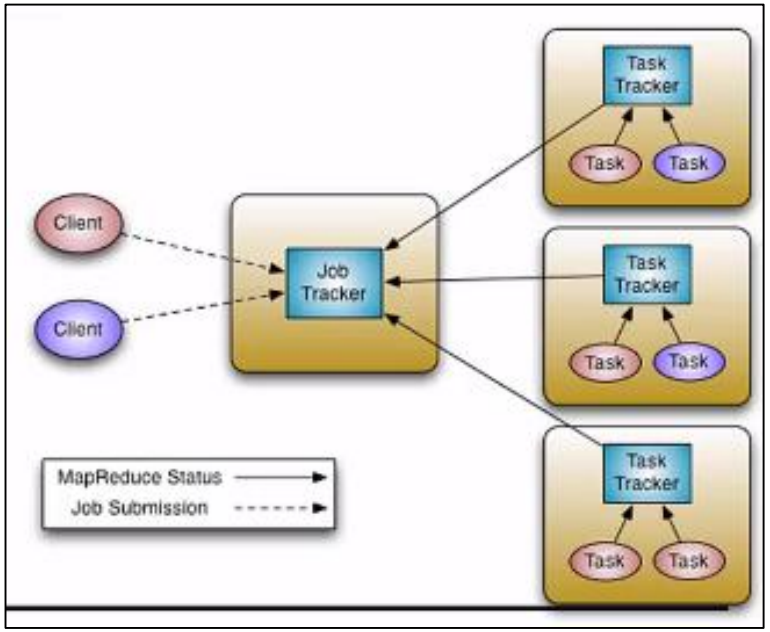
- Tiene problemas de escalabilidad a partir de los 5000 nodos. Hay muchas empresas grandes que tienen muchos más de 5000 nodos de Cluster hadoop
- Tiene graves problemas de rendimiento sobretodo en aplicaciones que no han sido hechas propiamente para ser batch
- No es capaz de gestionar correctamente los recursos.

1. INTRODUCCIÓN

MapReduce V1

Tenia 2 procesos denominados:

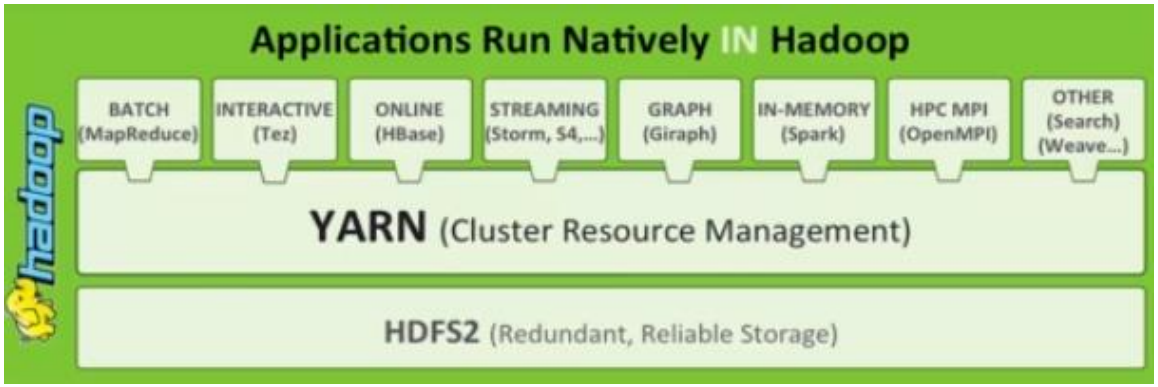
- JobTracker
  - En el maestro
  - Gestionaba los recursos del cluster y la planificación de cada uno de los trabajos.
- TaskTracker
  - Agente en los esclavos
  - Gestionaba las tareas y recursos en los nodos esclavos



1. INTRODUCCIÓN

Yarn

- Ha cambiado completamente la arquitectura de procesos



- Ahora MapReduce se reduce a una pequeña parte, pero podemos poner otros componentes que no sean Batch.
- Tez, Hbase, Storm, Giraph, Spark que son determinados productos que se han ido subiendo al carro de Big Data en los últimos tiempos y que no son procesos batch

1. INTRODUCCIÓN

Yarn

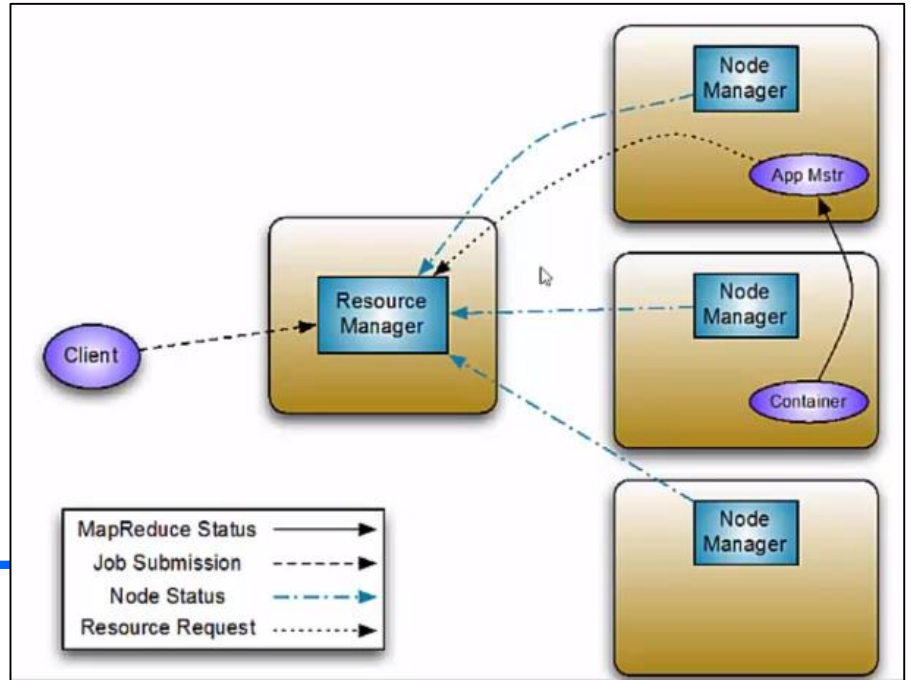
- La capa de Yarn que sería la capa del gestor del cluster
- HDFS versión 2 que ha tenido mejoras con respecto a versión 1
- Ya hemos trabajado anteriormente con HDFS 2
- Por ejemplo los snapshots corresponden esta versión
- La gestión que se hace a nivel de Yarn es mucho más efectiva que la que se hace a nivel de MapReduce

2. FUNCIONAMIENTO YARN

Los principales procesos de Yarn son:

1) **ResourceManager (RM):** En el maestro. Gestiona los recursos del cluster: que nodos hay, quién necesita más.
2) **Nodemanager (NM):** En cada uno de los esclavos. Se encarga de gestionar los recursos de cada nodo

- Ya no hay un solo proceso para hacer eso como en la v1. En cada nodo hay un gestor que se encarga de gestionar estos recursos



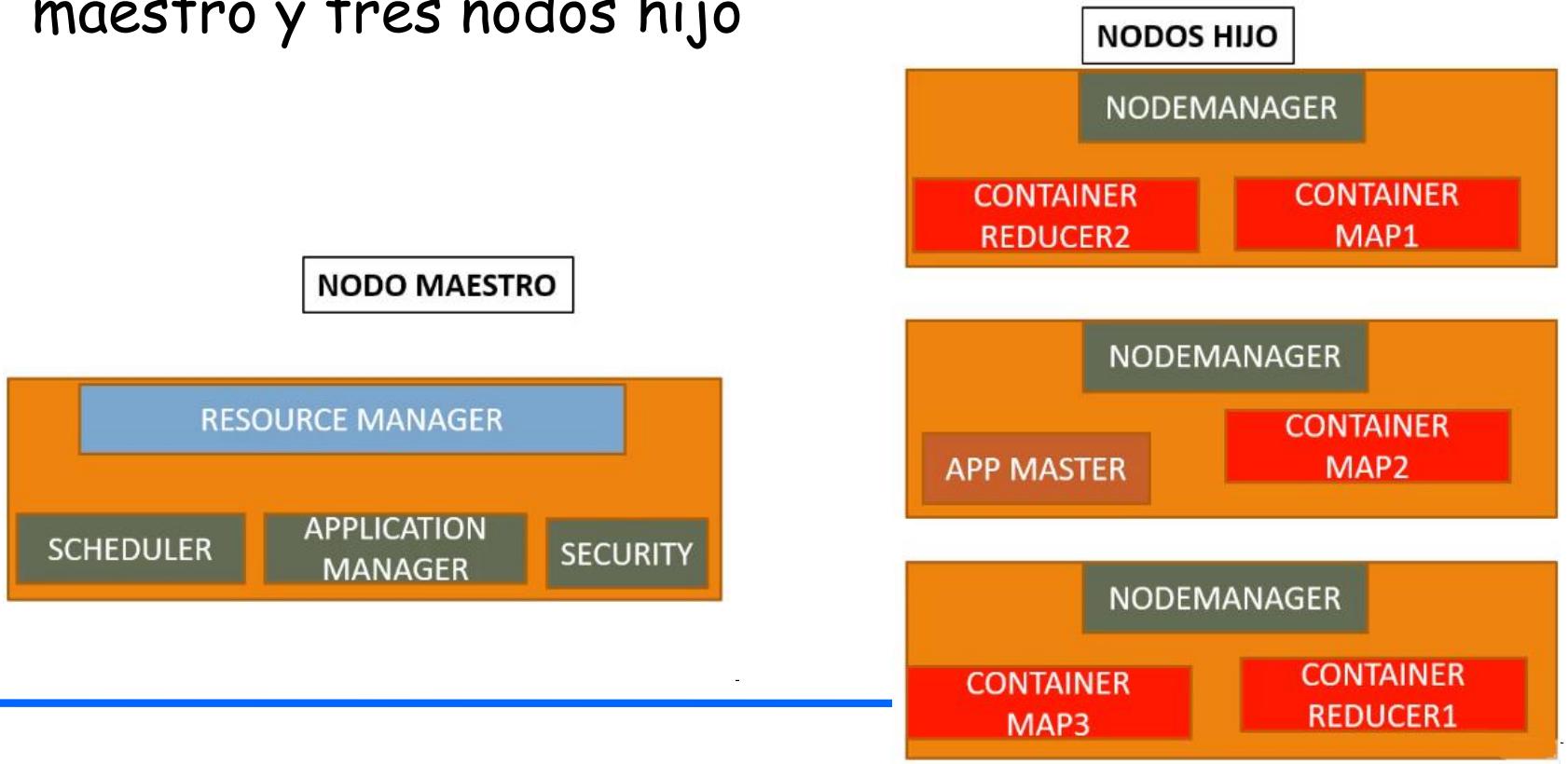
2. FUNCIONAMIENTO YARN

3) **ApplicationMaster (AM).** Vamos a tener uno por aplicación.

- Se encarga de gestionar los recursos que necesita la aplicación su ciclo de vida y la planificación de la aplicación
- Va a arrancar determinados contenedores que van a permitir que se gestione mejor todos los recursos del cluster.
- Ya no sólo son procesos batch, sino que estamos utilizando otro tipo de recursos.

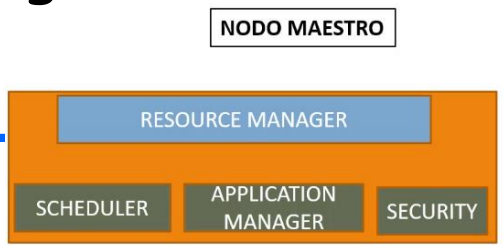
2. FUNCIONAMIENTO YARN

- Funcionamiento de los procesos Yarn y cómo se integran unos con otros.
- Suponemos una arquitectura de 4 servidores uno de maestro y tres nodos hijo



2. FUNCIONAMIENTO YARN

- En el **Nodo Maestro** se ejecuta el **Resource Manager**: Gestor de todo Yarn. Tiene 3 subcomponentes:
  - **Scheduler**: Determina la planificación de trabajos. Determina los recursos en un nodo que el App Master va a utilizar. Hay dos schedulers. El más habitual es el capacity Scheduler → el planificador de capacidad.
  - **Application Manager**: Cuando alguien solicita ejecutar una aplicación, arranca un App Master, en los nodos para que se encargue de esa aplicación
  - **Security**: Implementa medidas de seguridad



2. FUNCIONAMIENTO YARN

- En **cada nodo hijo** tenemos el **NodeManager**.
- Cuando se lanza una aplicación, primero se solicita al Application Manager que abra un App Master.
- Application Manager tendrá en cuenta el Scheduler para decidir en qué nodo va a arrancar el App Master
- App Master será el coordinador de la aplicación lanzada, si tenemos 100 aplicaciones lanzadas dentro de un clúster, tendremos 100 procesos App Master.
- Este es uno de los motivos por el que en los Clusters de tipo Yarn pueden escalar tanto

2. FUNCIONAMIENTO YARN

- El **App Master** junto con la información suministrada por el Scheduler, determinará en qué nodos va a ir arrancando los containers. Un container es un recurso de memoria y CPU donde se ejecuta algo
- En el ejemplo el APP Master ha arrancado 3 containers en los 3 nodos hijo, donde se ejecuta un mapper (app de tipo MapReduce). Y 2 contenedores MapReduce
- Cada proceso tiene que acceder a su bloque de datos en local. El bloque de datos de cada mapper está en el nodo donde se han lanzado
- Yarn es un gestor completo de recursos que no hace nada de trabajo de aplicación sino que sencillamente se encarga de gestionar los recursos del cluster.

3. CONFIGURAR YARN EN UN CLUSTER

Paso 1. Montaremos nuestro primer cluster con Yarn. Hasta ahora hemos arrancado solo la parte de los datos. Después de arrancar el sistema de ficheros HDFS con **start-dfs.sh,** mediante el comando **jps** vemos que tenemos tres procesos:

- datanode
- secondary namenode
- Namenode

```
hadoop@nodo1:~$ start-dfs.sh
Starting namenodes on [nodo1]
nodo1: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-nod
o1.out
localhost: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode
-nodo1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-s
econdarynamenode-nodo1.out
hadoop@nodo1:~$ jps
9730 SecondaryNameNode
18163 Jps
9194 NameNode
9388 DataNode
hadoop@nodo1:~$
```

3. CONFIGURAR YARN EN UN CLUSTER

Paso 2 . Ahora montaremos la parte de los procesos. Para ello no usaremos la versión 1 de MapReduce puesto que es obsoleta. Montaremos la versión 2 denominada Yarn.

Con **stop-dfs.sh** pararemos nuestro cluster. Se ve como estamos parando:

- el namenode
- el Datanode
- y por último el secundarynamenode.

```
hadoop@nodo1:~$ stop-dfs.sh
Stopping namenodes on [nodo1]
nodo1: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
hadoop@nodo1:~$
```

3. CONFIGURAR YARN EN UN CLUSTER

Ficheros de configuración

- Los ficheros de configuración estan en /opt/hadoop/etc/hadoop
- Para arrancar la parte de HDFS hemos tocado el **core-site.xml** y el **hdfs-site.xml**
- Para montar la parte de procesos Yarn tendremos que tocar otros dos:
  - **mapred-site.xml**
  - **yarn-site.xml**

```
hadoop@nodo1:/opt/hadoop/etc/hadoop$ ls
capacity-scheduler.xml      httpfs-env.sh            mapred-env.sh
configuration.xsl           httpfs-log4j.properties  mapred-queues.xml.template
container-executor.cfg      httpfs-signature.secret  mapred-site.xml.template
core-site.xml               httpfs-site.xml          slaves
hadoop-env.cmd              kms-acls.xml             ssl-client.xml.example
hadoop-env.sh               kms-env.sh               ssl-server.xml.example
hadoop-metrics2.properties  kms-log4j.properties     yarn-env.cmd
hadoop-metrics.properties   kms-site.xml             yarn-env.sh
hadoop-policy.xml           log4j.properties         yarn-site.xml
hdfs-site.xml               mapred-env.cmd
hadoop@nodo1:/opt/hadoop/etc/hadoop$
```

3. CONFIGURAR YARN EN UN CLUSTER

Paso 3. Modificaremos el primer fichero de configuración:

- Primero debemos hacer una copia de mapred-site.xml.template:

```
hadoop@nodo1:/opt/hadoop/etc/hadoop$ cp mapred-site.xml.template mapred-site.xml
hadoop@nodo1:/opt/hadoop/etc/hadoop$
```

- Editaremos el fichero mapred-site.xml con el siguiente contenido:

```
  GNU nano 6.2                    mapred-site.xml

<!-- Put site-specific property overrides in this file. -->

<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

- Indicamos que el motor que va a gestionar hadoop va a ser de tipo Yarn, mediante la propiedad mapreduce.framework.name, es decir el Framework que se va a utilizar va a ser Yarn

3. CONFIGURAR YARN EN UN CLUSTER

Paso 4. El segundo fichero que tenemos que tocar es el yarn-site.xml

- Indicaremos 3 propiedades

```
  GNU nano 6.2                                                yarn-site.xml
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>

<!-- Site specific YARN configuration properties -->
        <property>
                <name>yarn.resourcemanager.hostname</name>
                <value>localhost</value>
        </property>
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
                <value>org.apache.hadoop.mapred.ShuffleHandler</value>
        </property>
</configuration>
```

3. CONFIGURAR YARN EN UN CLUSTER

- **La 1° propiedad** especifica el nombre de la máquina donde se ejecuta el proceso resourcemanager, el gestor del clúster Yarn. Se indica el maestro que va a gestionar los procesos Yarn
- Las otras dos propiedades indican la clase que gestiona el Yarn
- Una indica que el gestor de servicios auxiliares que va a gestionar el MapReduce vale mapreduce_shuffe
- En la otra propiedad indicamos que la clase mapreduce_shuffle que se va a usar para hacer esa gestión, es una clase hadoop: **org.apache.hadoop.mapred.ShuffleHandler**
- Este es el valor por defecto que tiene la arquitectura de Yarn

4. INICIAR EL CLUSTER YARN

Paso 1. Una vez configurados los ficheros, para iniciar el cluster primero arrancamos siempre HDFS (los datos) con **start-dfs.sh:** se inicia el namenode, el datanode y el secundary namenode

```
hadoop@nodo1:/opt/hadoop/etc/hadoop$ start-dfs.sh
Starting namenodes on [nodo1]
nodo1: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-nodo1.out
localhost: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode-nodo1.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-secondarynamenode-nodo1.out
hadoop@nodo1:/opt/hadoop/etc/hadoop$
```

Paso 2. Arrancamos la segunda parte del cluster que es el Yarn, mediante el comando **start-yarn.sh.** Si no nos hemos equivocado en los fichero de configuración nos va a arrancar el resource manager y el nodemanager. El aplicationmanager no aparece hasta que no arrancamos una aplicación

```
hadoop@nodo1:/opt/hadoop/etc/hadoop$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /opt/hadoop/logs/yarn-hadoop-resourcemanager-nodo1.out
localhost: starting nodemanager, logging to /opt/hadoop/logs/yarn-hadoop-nodemanager-nodo1.out
hadoop@nodo1:/opt/hadoop/etc/hadoop$
```

4. INICIAR EL CLUSTER YARN

Paso 3. Mediante el comando jps podemos ver que tenemos todos los procesos en el mismo nodo:

- los de datos: DataNode, NameNode y SecundaryNameNode
- los de procesos: ResourceManager, NodeManager

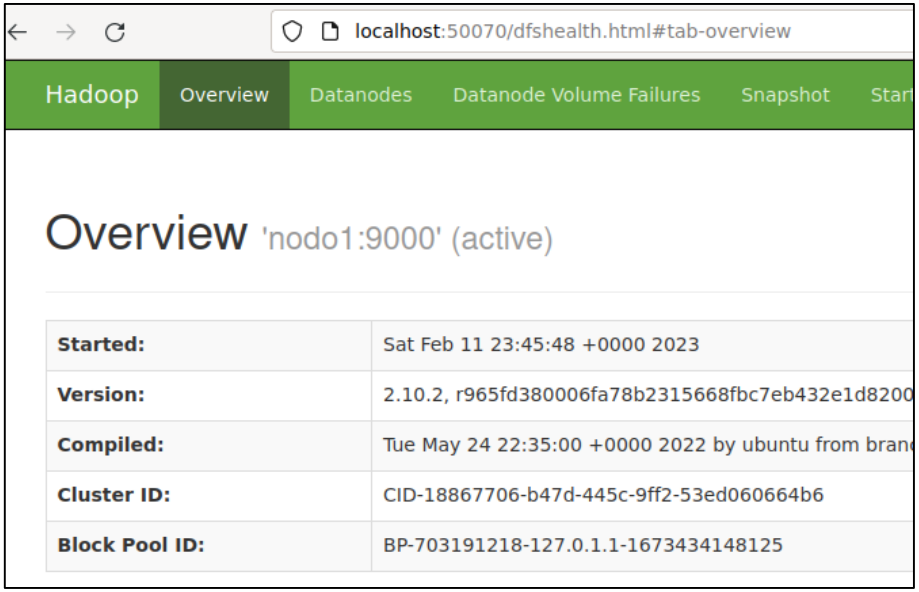
```
hadoop@nodo1:/opt/hadoop/etc/hadoop$ jps
687130 NodeManager
683090 NameNode
688550 Jps
686935 ResourceManager
683668 SecondaryNameNode
683301 DataNode
hadoop@nodo1:/opt/hadoop/etc/hadoop$
```

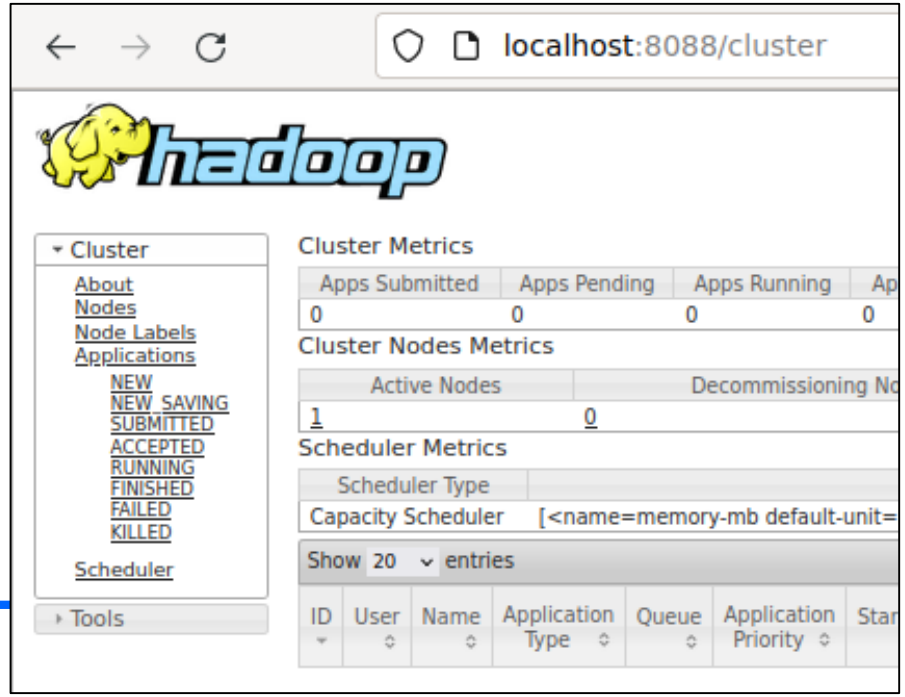
NOTA: Hemos visto cómo configurar y arrancar de una manera muy rápida nuestro primer cluster Yarn completo, es decir con HFS y con Yarn, dentro de un solo nodo.

5. WEB ADMINISTRACION YARN

Paso 1. De la misma manera que tenemos una página web donde podemos ver la configuración HDFS, también disponemos de una página parecida para la configuración de YARN

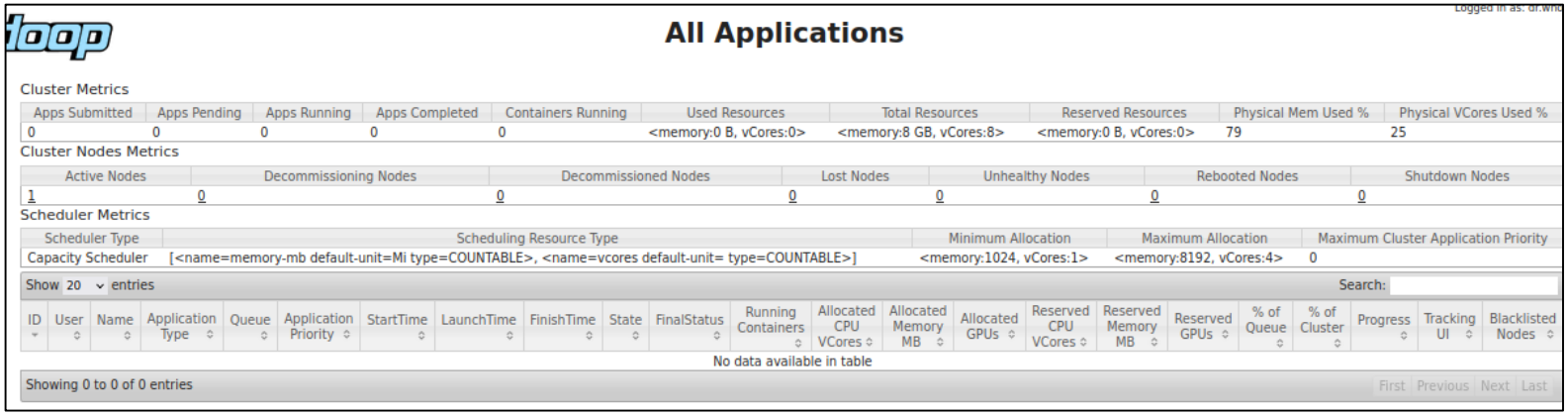
- nodo1:50070 o localhost:50070 → para HDFS
- nodo1:8088 o localhost:8088 → para YARN





5. WEB ADMINISTRACION YARN

Paso 2. En la ventana principal **Cluster** aparecen un montón de opciones a la izquierda. Es un poco distinta a la de HDFS:

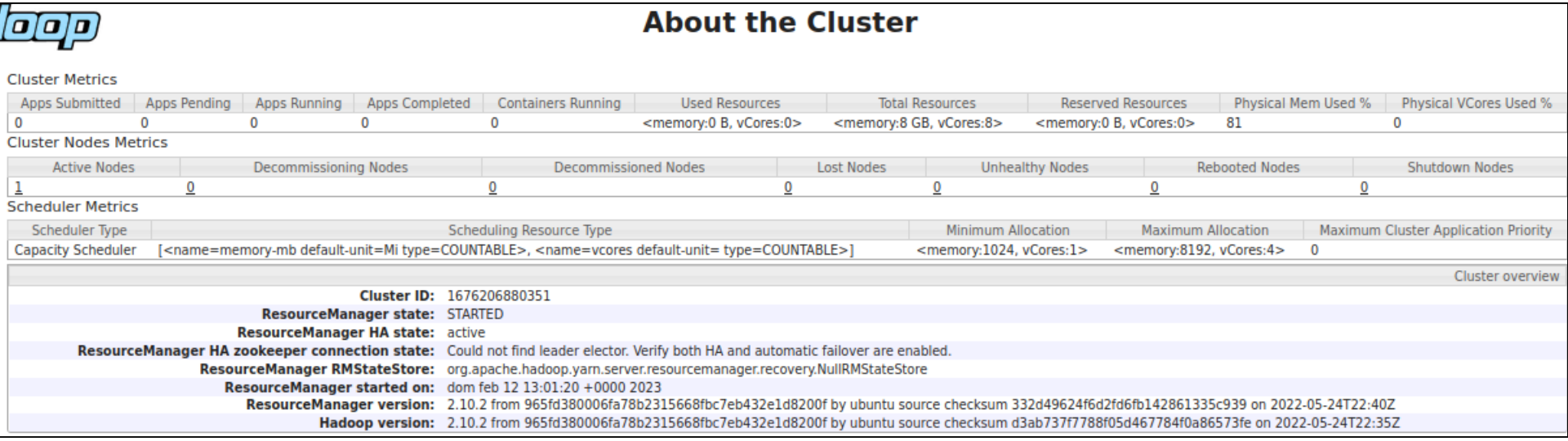


Primero vemos todas las métricas del cluster junto con la planificación que hay reservada para aplicaciones (ahora no tenemos ninguna). Podemos ver: memoria física utilizada, cores usados, etc

5. WEB ADMINISTRACION YARN

Paso 3. En la opción de **Cluster/About** nos permite ver información de todo nuestro cluster:

- cuando ha arrancado
- si está activo o no el resourcemanager
- la memoria que se está utilizando (por defecto asume que cada nodo debería tener un mínimo 1GB y un máximo 8GB), etc



About the Cluster

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	Reserved Resources	Physical Mem Used %	Physical VCores Used %
0	0	0	0	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>	<memory:0 B, vCores:0>	81	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

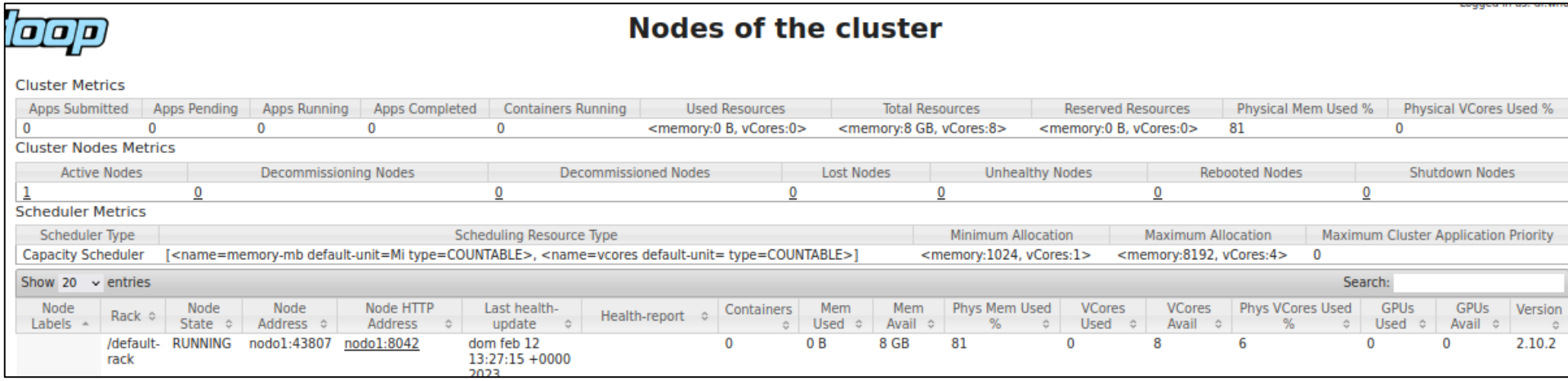
Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[<name=memory-mb default-unit=Mi type=COUNTABLE>, <name=vcores default-unit= type=COUNTABLE>]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Cluster overview

Cluster ID:	1676206880351
ResourceManager state:	STARTED
ResourceManager HA state:	active
ResourceManager HA zookeeper connection state:	Could not find leader elector. Verify both HA and automatic failover are enabled.
ResourceManager RMStateStore:	org.apache.hadoop.yarn.server.resourcemanager.recovery.NullRMStateStore
ResourceManager started on:	dom feb 12 13:01:20 +0000 2023
ResourceManager version:	2.10.2 from 965fd380006fa78b2315668fbc7eb432e1d8200f by ubuntu source checksum 332d49624f6d2fd6fb142861335c939 on 2022-05-24T22:40Z
Hadoop version:	2.10.2 from 965fd380006fa78b2315668fbc7eb432e1d8200f by ubuntu source checksum d3ab737f7788f05d467784f0a86573fe on 2022-05-24T22:35Z

5. WEB ADMINISTRACION YARN

Paso 4. La opción **Cluster/Nodes** nos permita ver los nodos que tenemos activos dentro del cluster. En este momento solo tenemos uno, con el que nos podemos comunicar por el puerto 8042.



loop

Nodes of the cluster

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	Reserved Resources	Physical Mem Used %	Physical VCores Used %
0	0	0	0	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>	<memory:0 B, vCores:0>	81	0

Cluster Nodes Metrics

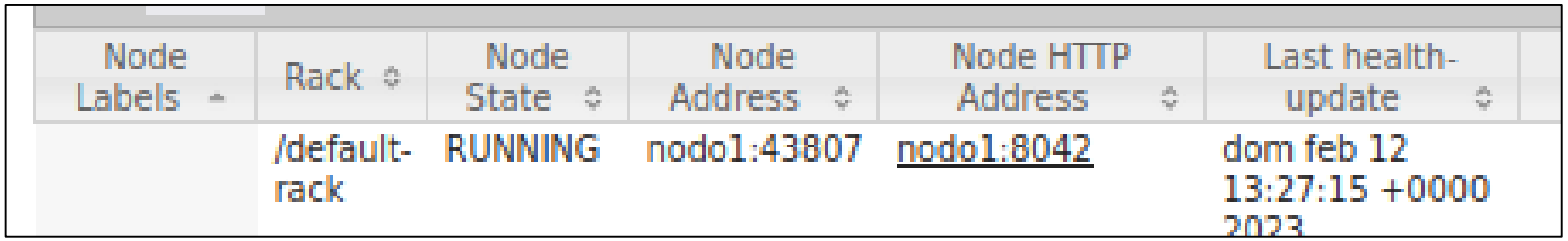
Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[<name=memory-mb default-unit=Mi type=COUNTABLE>, <name=vcores default-unit= type=COUNTABLE>]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries Search:

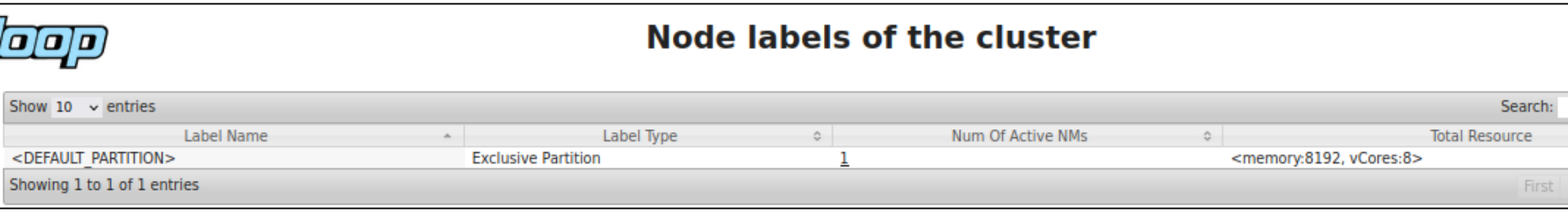
Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	Phys Mem Used %	VCores Used	VCores Avail	Phys VCores Used %	GPUs Used	GPUs Avail	Version
	/default-rack	RUNNING	nodo1:43807	nodo1:8042	dom feb 12 13:27:15 +0000 2023		0	0 B	8 GB	81	0	8	6	0	0	2.10.2



Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update
	/default-rack	RUNNING	nodo1:43807	nodo1:8042	dom feb 12 13:27:15 +0000 2023

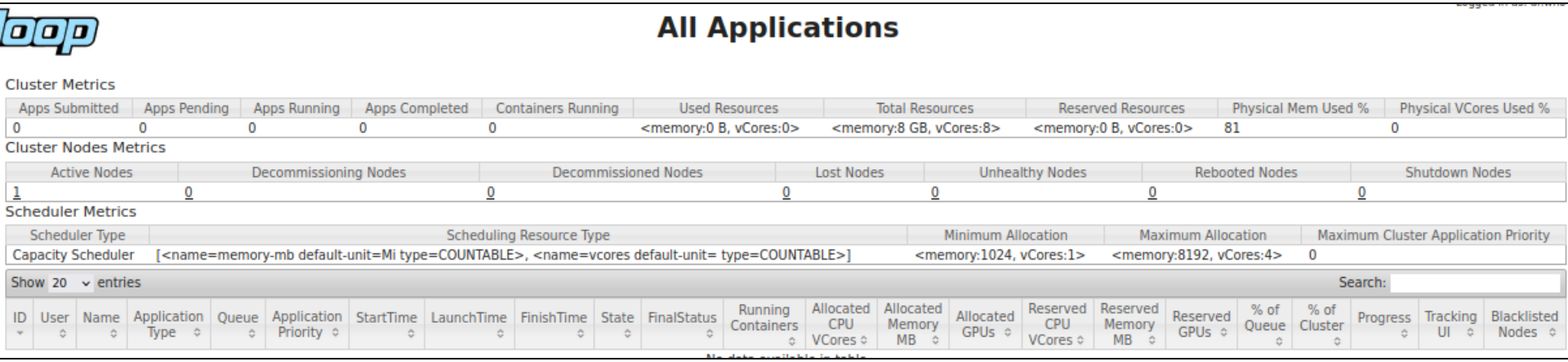
5. WEB ADMINISTRACION YARN

Paso 5. En la opción **Cluster/Node Labels**, se muestran las etiquetas que podemos poner a cada nodo



5. WEB ADMINISTRACION YARN

Paso 6. En la pestaña **Clusters/Applications**, vemos las aplicaciones que tenemos en este momento funcionando. Se puede filtrar por estado: nuevas, lanzadas, aceptadas, ejecutándose, terminadas, que han fallado, etc.



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	Reserved Resources	Physical Mem Used %	Physical VCores Used %
0	0	0	0	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>	<memory:0 B, vCores:0>	81	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
1	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[<name=memory-mb default-unit=Mi type=COUNTABLE>, <name=vcores default-unit= type=COUNTABLE>]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Allocated GPUs	Reserved CPU VCores	Reserved Memory MB	Reserved GPUs	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes

No data available in table

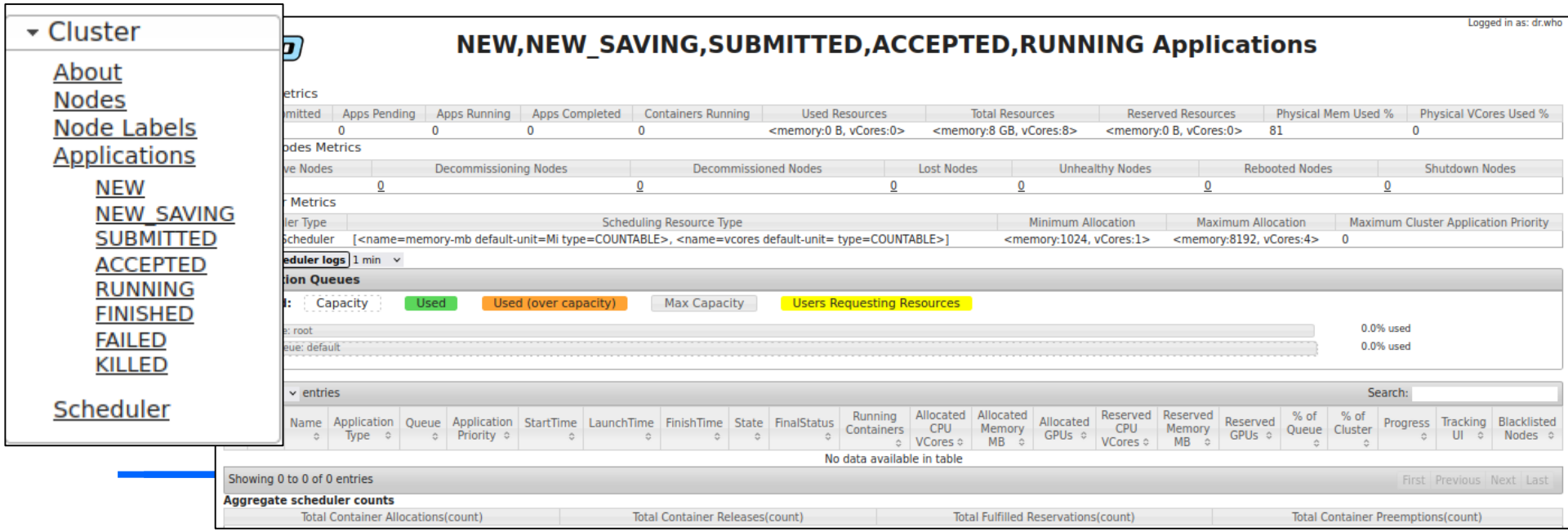
Applications
- NEW
- NEW_SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED

Evidentemente en este momento no tenemos nada, porque todavía no hemos lanzado ninguna aplicación

5. WEB ADMINISTRACION YARN

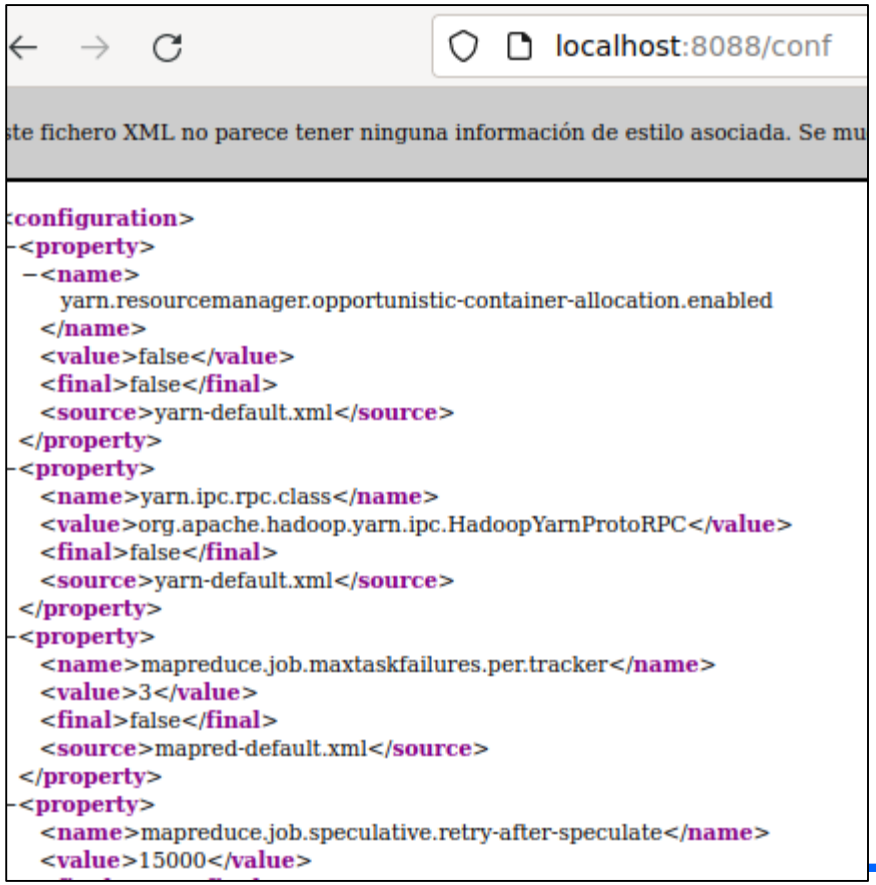
Paso 7. En el pestaña **Cluster/Scheduler** podemos ver todas las métricas del planificador: que capacidad tenemos. cuánto se ha utilizado, cuánto se ha utilizado sobre la capacidad actual o permitida.

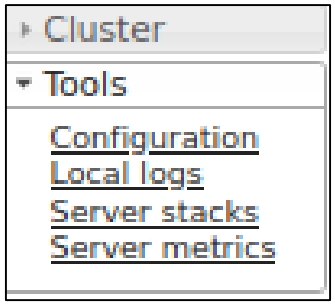
Esta vacío puesto que no hemos lanzado ninguna aplicación. Una vez que se haya lanzado algún aplicativo permite ver lo que ha sucedido y los recursos que se han estado utilizando a nivel del cluster.



5. WEB ADMINISTRACION YARN

Paso 8. En la pestaña **Tools/Configuration** podemos ver la configuración de este nodo del cluster.

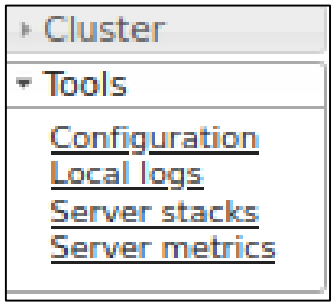


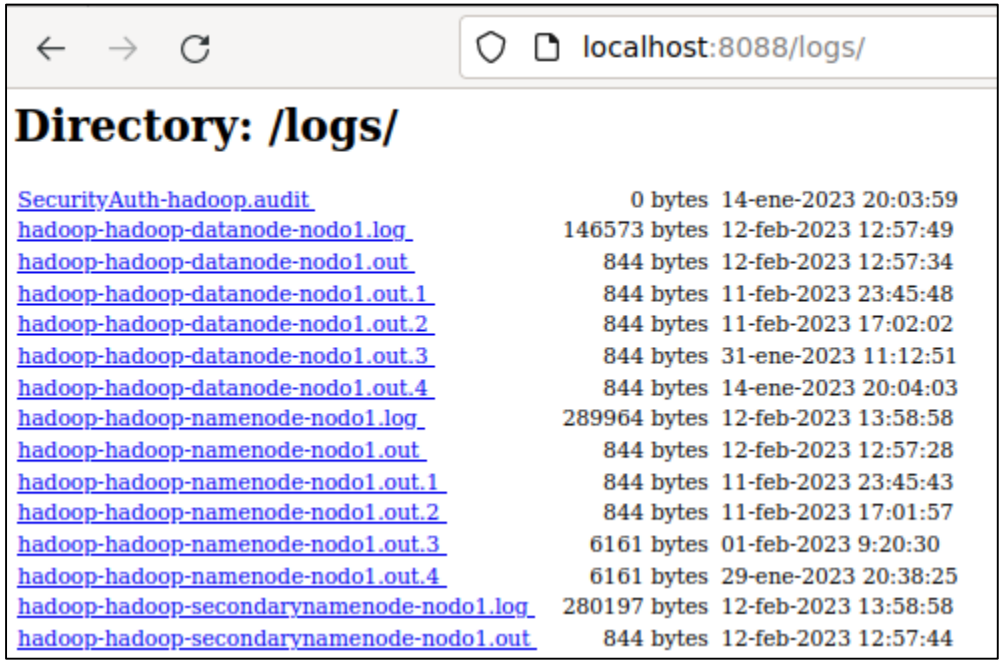


Hadoop tiene muchas propiedades de las cuales sólo hemos tocado 2, las que nos han interesado. Las que no ponemos se ponen por defecto

5. WEB ADMINISTRACION YARN

Paso 9. En **Tools/Local Logs** podemos ver los log locales. En realidad tenemos los log de todos los procesos del cluster: datanode, namenode, Secondarynamenode, ResourceManager, etc

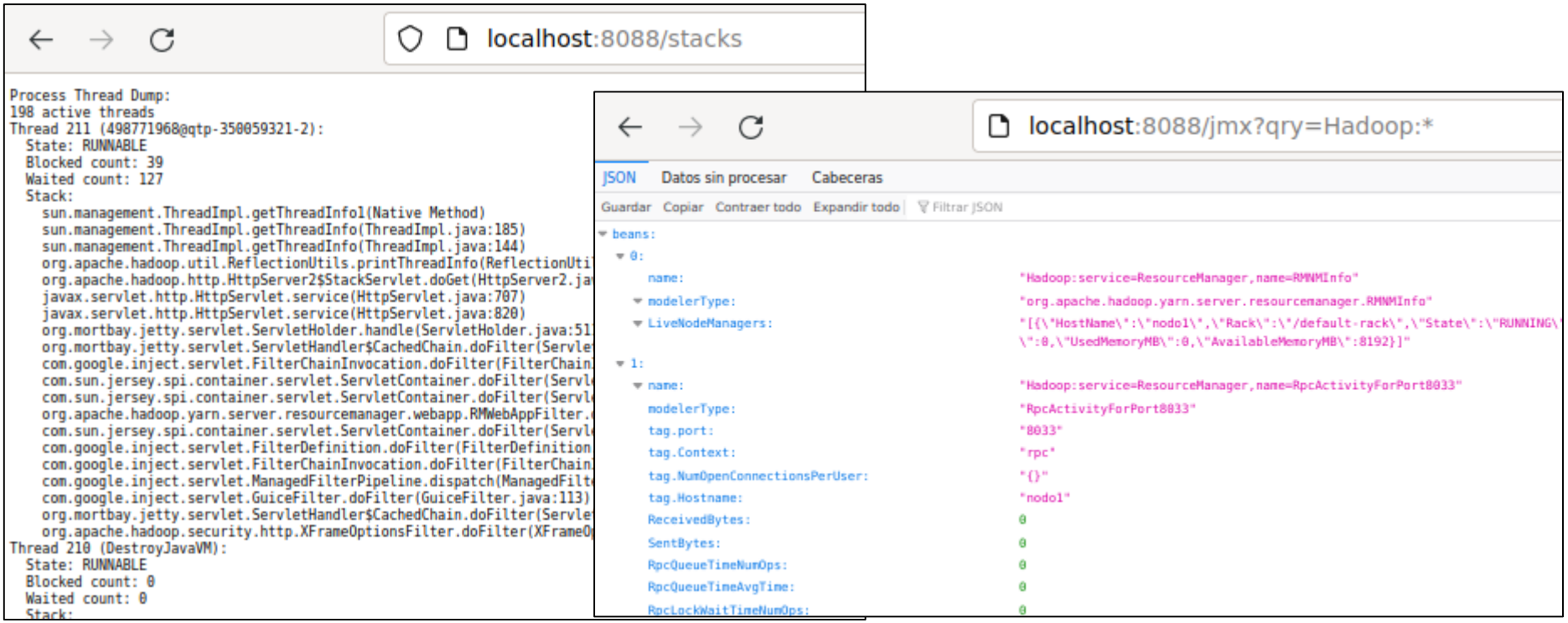




5. WEB ADMINISTRACION YARN

Paso 10. También podemos ver el Server stacks (en **cluster/Server stacks**) y las métricas del server (en **cluster/Server metrics**): qué cosas se han lanzado, que tengo libre, estadísticas de uso, etc.

Al profundizar más en los clusters hadoop, estas propiedades o estos logs pueden ser de utilidad.

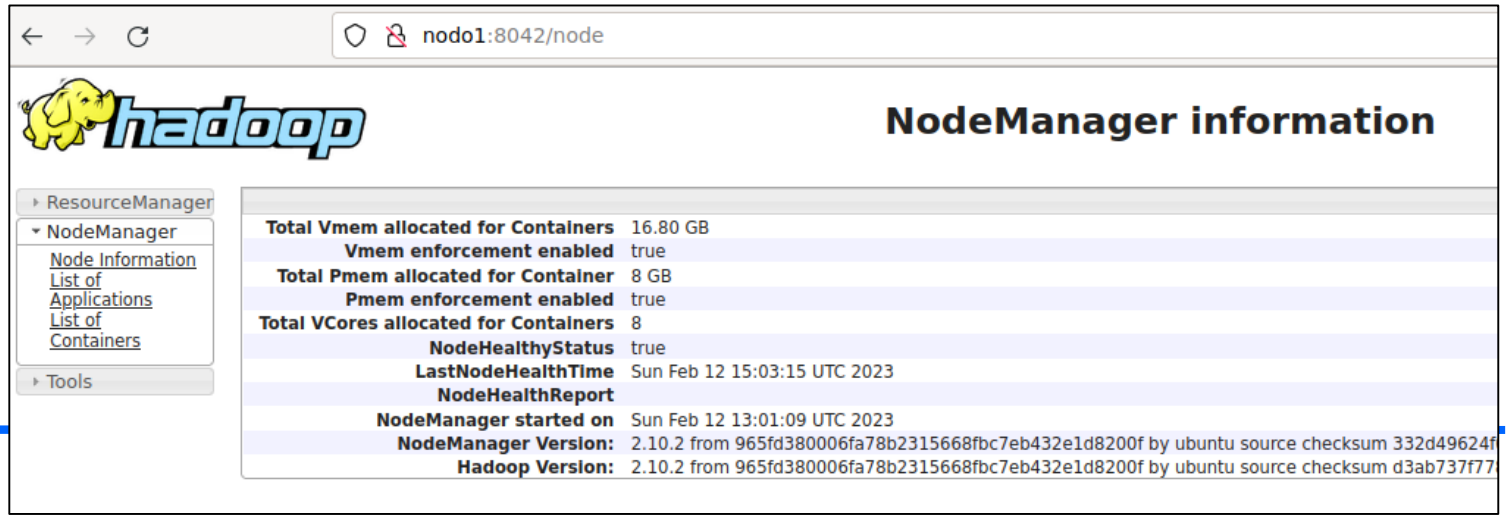


5. WEB ADMINISTRACION YARN

Paso 11. Por último si nos vamos a **cluster/Nodes** podemos entrar a ver la información general de cada nodo concreto haciendo click en su enlace. Ya no estamos en el puerto 8088 sino en el 8042.

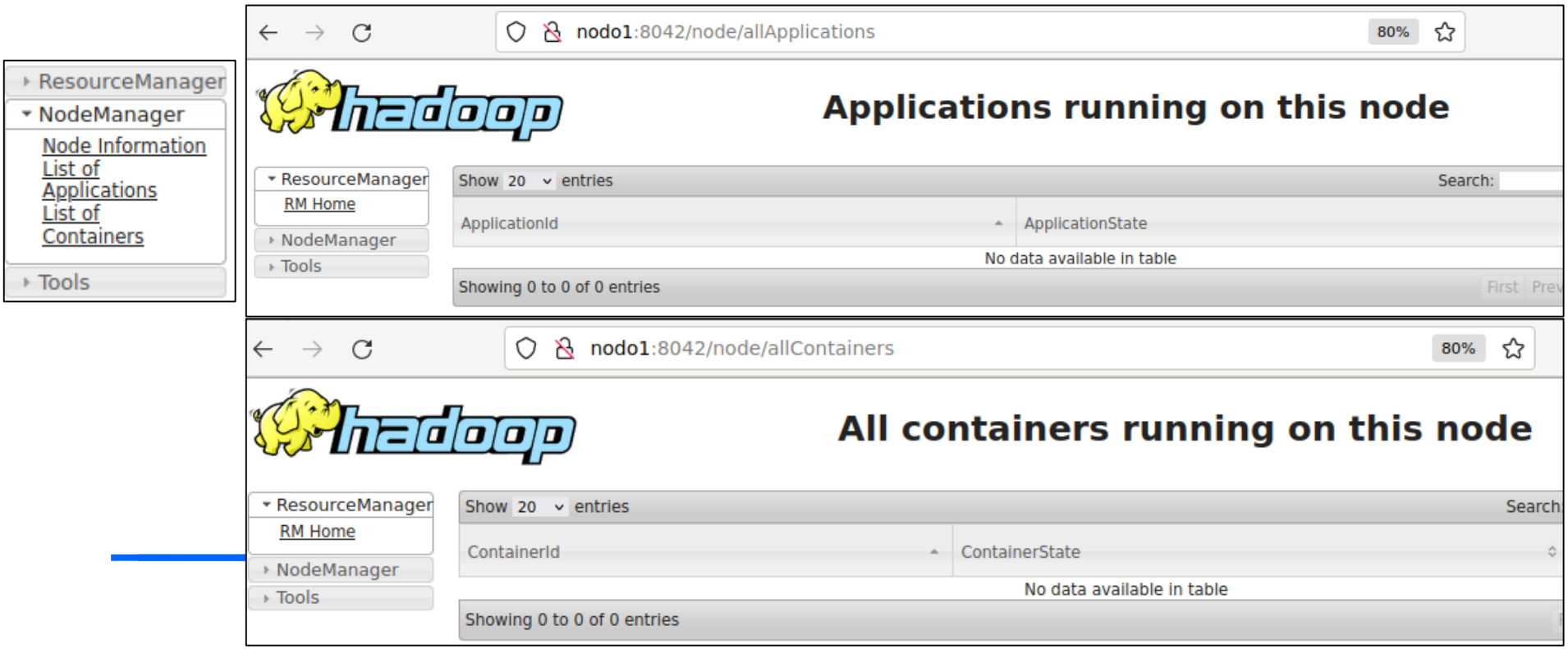
Node Labels	Rack	Node State	Node Address	Node HTTP Address
	/default-rack	RUNNING	nodo1:43807	nodo1:8042

La información del NodeManager que aparece es virtual (memory, process) y no se corresponde con la máquina real que tenemos por debajo.



5. WEB ADMINISTRACION YARN

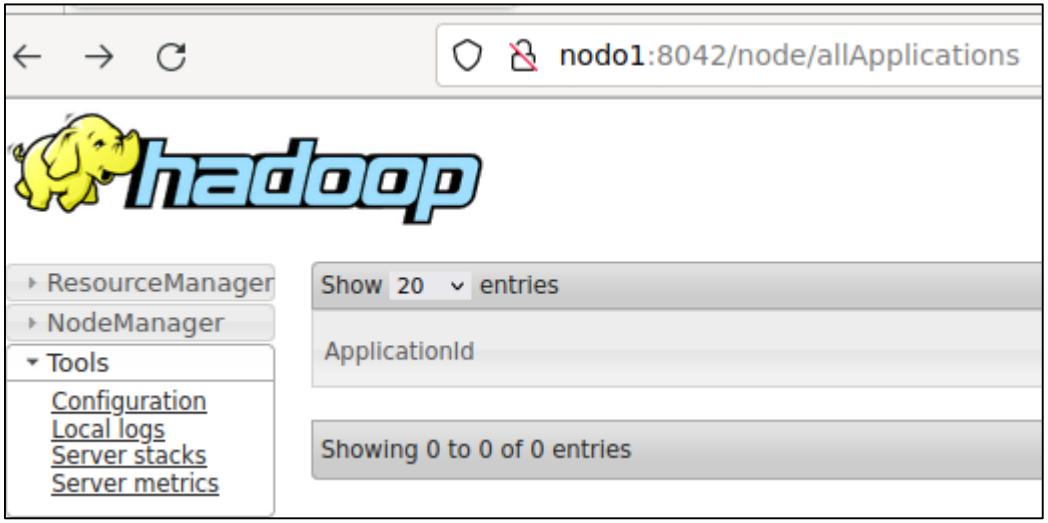
Paso 12. En la opción **List of Applications** podemos ver las aplicaciones que están lanzándose aquí o que le han tocado. En la opción **List of Containers** podemos ver los containers. Por cada aplicación se abren los contenedores que son los recursos reales que se utilizan para hacer los procesos. Como vemos no hay nada



5. WEB ADMINISTRACION YARN

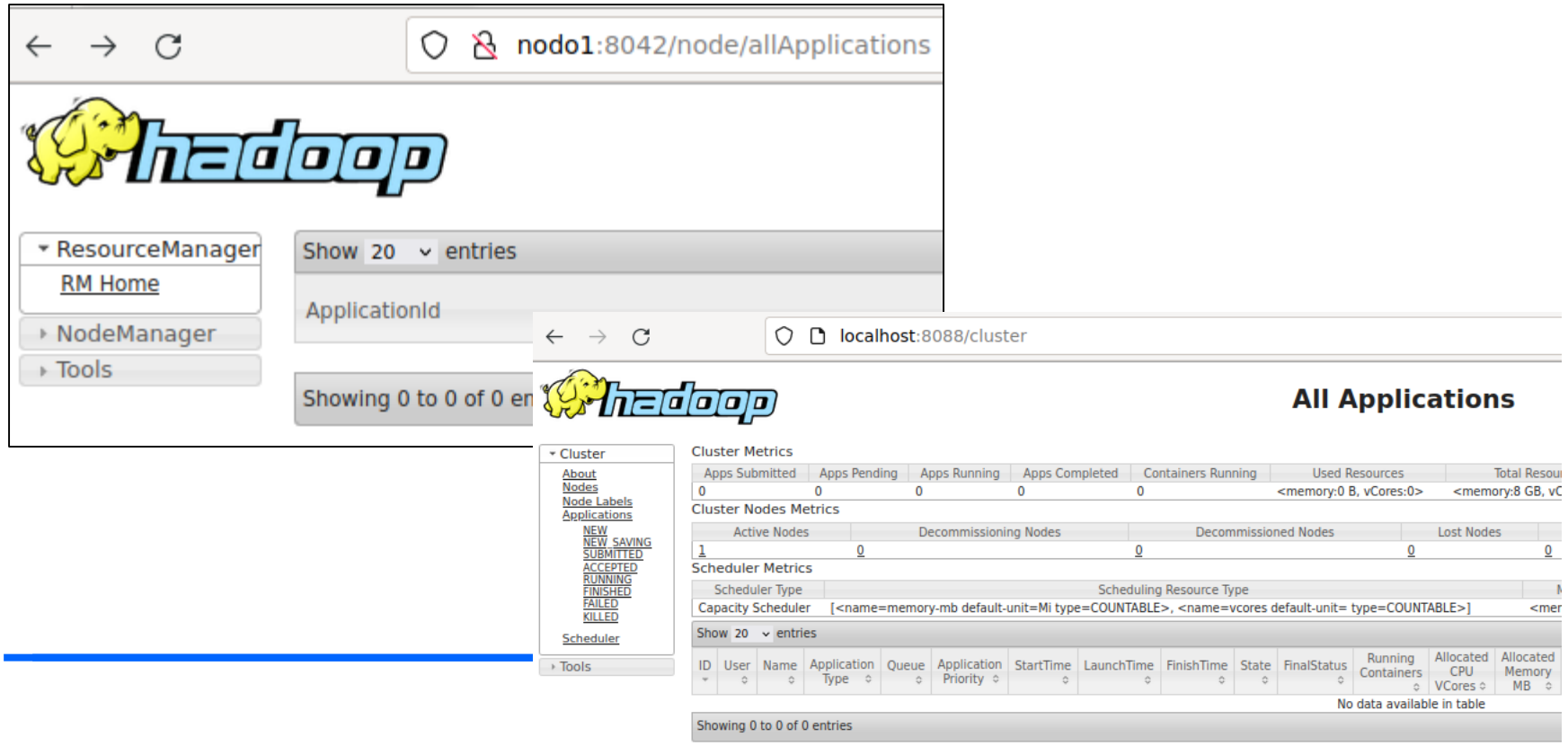
Paso 13. En **Tools** también podemos ver lo mismo

- el fichero de configuración,
- los logs locales
- las estadísticas del servidor.



5. WEB ADMINISTRACION YARN

Paso 14. Si queremos volver de nuevo a la ventana principal donde estaba trabajando con el clúster, hacemos click en el ResourceManager, y después en RM Home, volvemos de nuevo al cluster nodo1:8088 o localhost:8088



5. WEB ADMINISTRACION YARN

Paso 15. En la versión 3 de Hadoop, a veces puede generar un error al realizar alguno de los ejemplos. Para solucionarlo, es necesario añadir algunas librerías en yarn-site.xml:

```
<property>
<name>yarn.application.classpath</name>
      <value>
                  /opt/hadoop3/hadoop/etc/hadoop,
                  /opt/hadoop3/share/hadoop/common/*,
                  /opt/hadoop3/share/hadoop/common/lib/*,
                  /opt/hadoop3/share/hadoop/hdfs/*,
                  /opt/hadoop3/share/hadoop/hdfs/lib/*,
                  /opt/hadoop3/share/hadoop/mapreduce/*,
                  /opt/hadoop3/share/hadoop/mapreduce/lib/*,
                  /opt/hadoop3/share/hadoop/yarn/*,
                  /opt/hadoop3/share/hadoop/yarn/lib/*
</value>
</property>
```

5. WEB ADMINISTRACION YARN

Paso 16. Para apagar hadoop se hace en orden inverso, primero se apaga yarn y después hdfs.

```
hadoop@nodo1:/opt/hadoop/etc/hadoop$ stop-yarn.sh
stopping yarn daemons
stopping resourcemanager
localhost: no nodemanager to stop
no proxyserver to stop
hadoop@nodo1:/opt/hadoop/etc/hadoop$ stop-dfs.sh
Stopping namenodes on [nodo1]
nodo1: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
hadoop@nodo1:/opt/hadoop/etc/hadoop$
```

6. FUNCIONAMIENTO DE MAPREDUCE

- Haremos una pequeña descripción del funcionamiento MapReduce.
- Aunque Yarn le ha cogido terreno en algunas cosas de la gestión del clúster, sigue siendo uno de los procesos más habituales de BigData
- La mayor parte de las herramientas de consulta están diseñadas para realizar consultas simples que deban ejecutarse rápidamente.
- El dato suele estar indexado y por tanto solo pequeñas porciones de datos se examinan durante la búsqueda.
- Esta solución en cambio no es útil para datos que no pueden ser indexados de tipo semi estructurado (textos) o sin estructurar (multimedia)
- Para responder una query en esta solución es necesario examinar todos los datos
- Hadoop utiliza MapReduce para realizar un análisis exhaustivo que permita hacer estos procesos de búsqueda en cargas masivas de manera rápida.

6. FUNCIONAMIENTO DE MAPREDUCE

- MapReduce es un algoritmo de datos de tipo batch que va implementando procesos en paralelo.
- La filosofía es divide y vencerás: en vez de ejecutar un proceso de manera secuencial, se trocea y se manda a cada nodo y eso hace que el rendimiento sea mucho mayor, al ejecutarse en paralelo
- MapReduce de forma simple distribuye las tareas a través de los nodos de un cluster ejecutando una función "map". Esta función estudia el problema, divide cada proceso en múltiples trozos y los manda a diferentes maquinas a ejecutarse en paralelo contra cada uno de los bloques de 128MB de datos.
- Si tenemos 50 trozos va a ejecutarse 50 veces el mismo proceso en paralelo pero con datos distintos.
- Los resultados de este proceso paralelo se recogen y se distribuyen a través de distintos servidores que ejecutan una función "reduce", que toma los resultados de los trozos y los recombina para obtener una respuesta simple

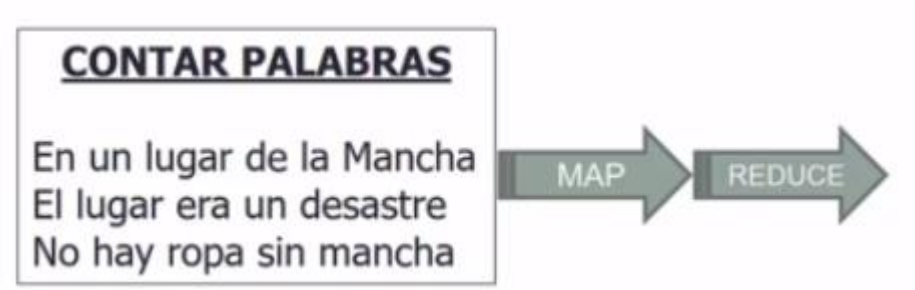
6. FUNCIONAMIENTO DE MAPREDUCE

Vamos a tener en realidad tres pasos/componentes:

- **Procesos Mapper:**
  - Los programas Map Reduce se dividen en Mappers, tareas que se ejecutan en los nodos contra los bloques de datos
  - Cada tarea MAP ataca a un solo bloque de datos HDFS
  - Siempre se ejecutan en el nodo donde reside el bloque
- **Shuffle y Sort**
  - Ordena y consolida los datos intermedios (temporales) que han generado los mapper
  - Se lanzan después de que todos los mappers hayan terminado y antes de que se lancen los procesos Reducer
- **Reducer**
  - Operan sobre los datos intermedios que se han generado en el paso anterior. Son los que generan la salida final .

6. FUNCIONAMIENTO DE MAPREDUCE

- Imaginamos que queremos contar palabras. Puede parecer sencillo pero si tenemos millones de datos ya no es tan sencillo
- Esto sería un proceso MapReduce. Cogemos un fichero que tiene unas cuantas frases y cuento las palabras para ver cuántas veces aparece cada palabra.



Resultado final

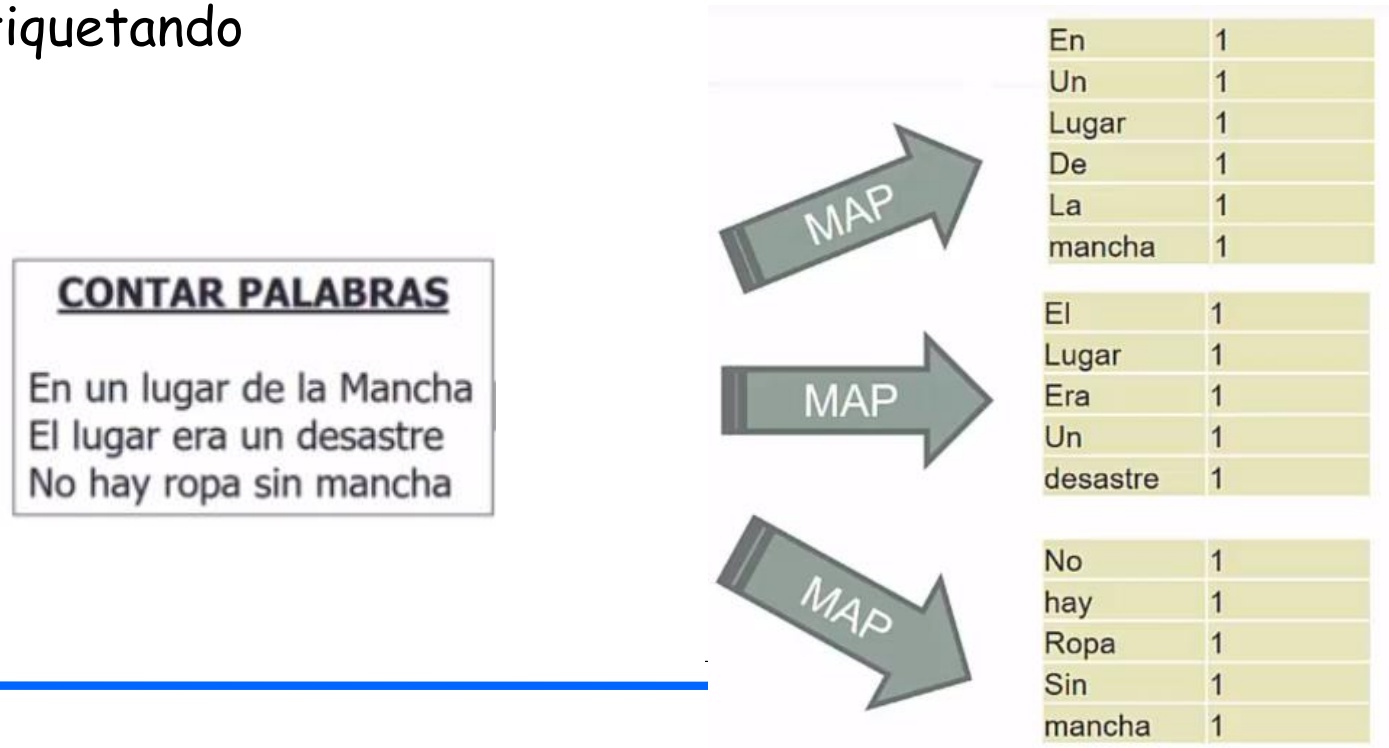
RESULTADOS

De	1
El	1
En	1
Era	1
Hay	1
La	1
Lugar	2
Mancha	2
No	1
Ropa	1
Sin	1
Un	2

6. FUNCIONAMIENTO DE MAPREDUCE

a) Procesos Mapper

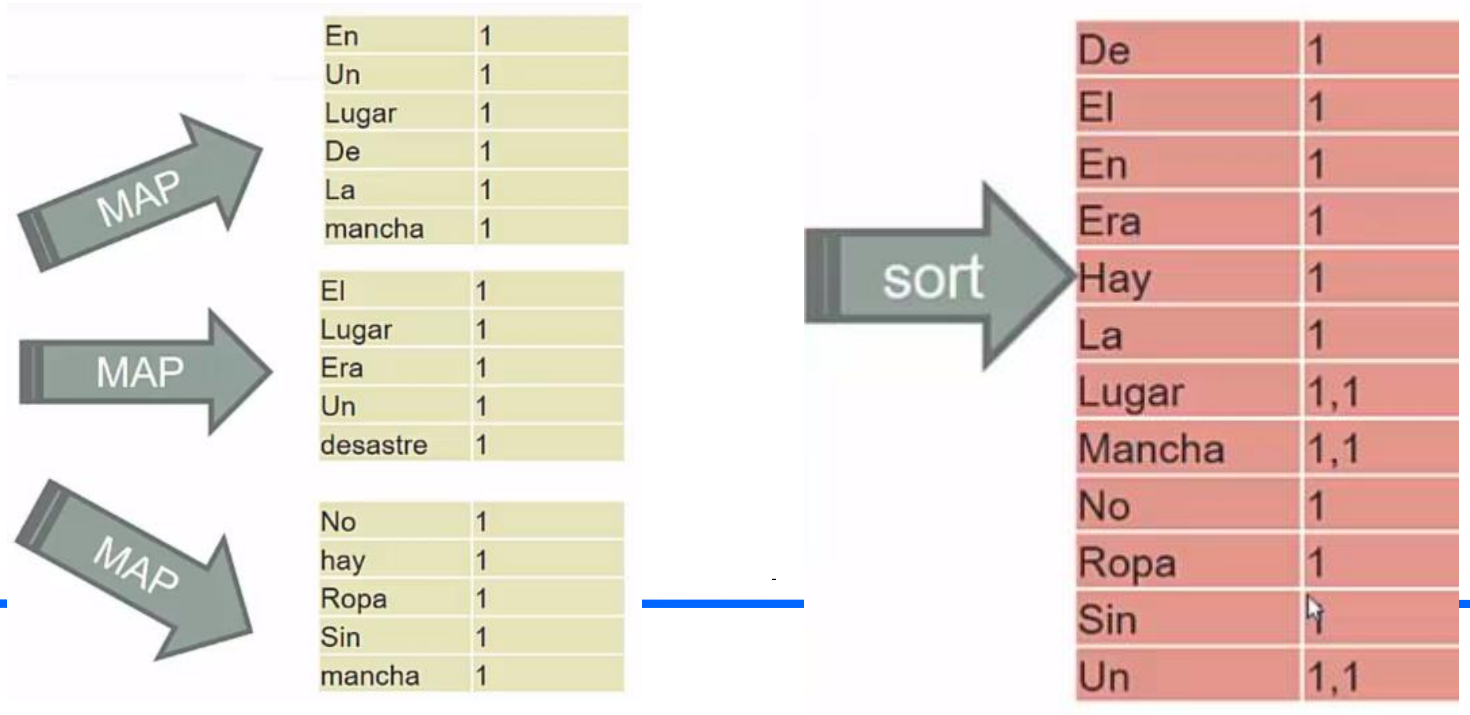
- Vamos a suponer que cada una de estas frases es un bloque. Imaginamos que en vez de ser 3 líneas son 3 bloques de 128 megas. Hay 3 Mappers donde cada mapper coge un trozo de ese fichero y empieza a contar las ocurrencias que van apareciendo y las va etiquetando



6. FUNCIONAMIENTO DE MAPREDUCE

b) Shuffle y sort

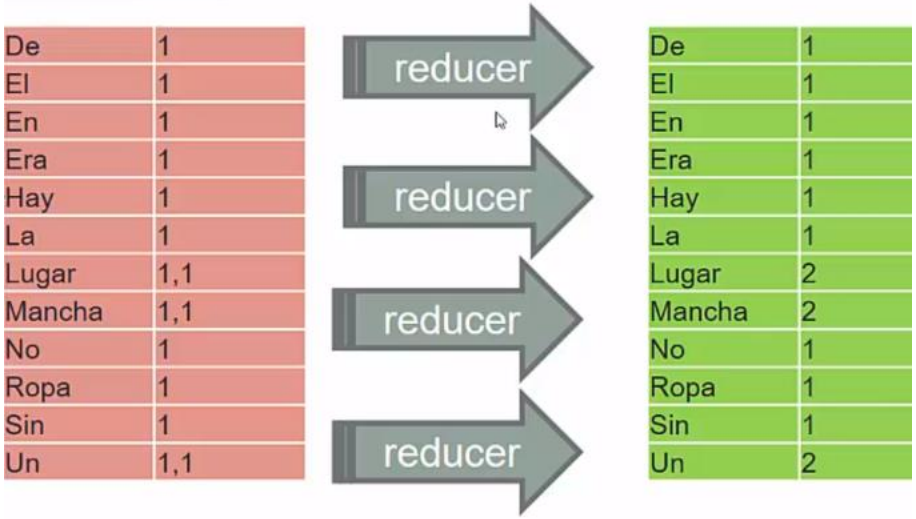
- En el siguiente paso Shuffle trabaja con clave-valor: Confecciona una lista donde la clave es cada palabra y los valores son el número de veces que aparece.
- En el proceso intermedio generamos una serie de claves con posibles valores que no tienen porqué ser sólo 1



6. FUNCIONAMIENTO DE MAPREDUCE

c) Procesos Reducers

En el proceso final los Reducers van a coger esto básicamente lo que van a hacer como veis es contar



Un proceso grande, lo hemos dividido en múltiples procesos paralelos. Hemos generado con sort y Shuffle una serie de datos intermedios y por último otros procesos de tipo Reducers que normalmente son menos que los Mappers nos han hecho el resultado final.

7. EJEMPLO CON MAPREDUCE

Paso 1. Contaremos las palabras que más aparecen en el libro **El Quijote** mediante un proceso Hadoop MapReduce. Descargaremos un fichero muy pequeño, el cual no generará muchos Mappers ni Reducers.

Desde la pagina Proyecto Gutenberg **https://www.gutenberg.org** podemos descargar libremente El Quijote, como simple texto.

7. EJEMPLO CON MAPREDUCE

Paso 2. Movemos el fichero quijote.txt al directorio /tmp. Creamos el directorio /libros dentro de HDFS y subimos el fichero quijote.txt a este directorio

```
hadoop@nodo1:~/Downloads$ ls
hadoop-2.10.2.tar.gz  hadoop-3.3.4.tar.gz  quijote.txt
hadoop@nodo1:~/Downloads$ mv quijote.txt /tmp
hadoop@nodo1:~/Downloads$ hdfs dfs -mkdir /libros
hadoop@nodo1:~/Downloads$ hdfs dfs -put /tmp/quijote.txt /libros
hadoop@nodo1:~/Downloads$ hdfs dfs -ls /libros
Found 1 items
-rw-r--r--   1 hadoop supergroup    2226045 2023-02-18 21:24 /libros/quijote.txt
hadoop@nodo1:~/Downloads$
```

OBJETIVO: Haremos que nos genere en otro fichero el número de palabras que tiene y cuantas veces aparecen cada palabra.

7. EJEMPLO CON MAPREDUCE

Paso 3. Recordamos que en **/opt/hadoop/share/hadoop/mapreduce** tenemos una serie de librerías entre la que destaca hadoop-mapreduce-examples

```
hadoop@nodo1:~$ pwd
/home/hadoop
hadoop@nodo1:~$ cd /opt/hadoop/share/hadoop/mapreduce/
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ ls
hadoop-mapreduce-client-app-2.10.2.jar
hadoop-mapreduce-client-common-2.10.2.jar
hadoop-mapreduce-client-core-2.10.2.jar
hadoop-mapreduce-client-hs-2.10.2.jar
hadoop-mapreduce-client-hs-plugins-2.10.2.jar
hadoop-mapreduce-client-jobclient-2.10.2.jar
hadoop-mapreduce-client-jobclient-2.10.2-tests.jar
hadoop-mapreduce-client-shuffle-2.10.2.jar
hadoop-mapreduce-examples-2.10.2.jar
jdiff
lib
lib-examples
sources
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$
```

7. EJEMPLO CON MAPREDUCE

Paso 4. Con el comando jar, podemos ver una lista de determinados comandos o opciones que hay dentro de este paquete. Utilizaremos el comando wordcount que permite contar palabras dentro de los ficheros.

```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ jar tf hadoop-mapreduce-examples-2.10.2.jar
META-INF/
org/apache/
org/apache/hadoop/
org/apache/hadoop/examples/
org/apache/hadoop/examples/terasort/
org/apache/hadoop/examples/dancing/
org/apache/hadoop/examples/pi/
org/apache/hadoop/examples/pi/math/
org/apache/hadoop/examples/BaileyBorweinPlouffe$BbpInputFormat$1.class
```

```
org/apache/hadoop/examples/DBCountPageView.class
org/apache/hadoop/examples/DBCountPageView$PageviewMapper.class
org/apache/hadoop/examples/WordCount.class
org/apache/hadoop/examples/AggregateWordCount$WordCountPlugInClass.class
org/apache/hadoop/examples/SecondarySort.class
META-INF/maven/
META-INF/maven/org.apache.hadoop/
META-INF/maven/org.apache.hadoop/hadoop-mapreduce-examples/
META-INF/maven/org.apache.hadoop/hadoop-mapreduce-examples/pom.xml
META-INF/maven/org.apache.hadoop/hadoop-mapreduce-examples/pom.properties
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$
```

7. EJEMPLO CON MAPREDUCE

Paso 5. Ejecutaremos el programa hadoop-mapreduce **wordcount** que cogerá el contenido del directorio /libros y dejará en /salida_libros el resultado de las palabras contadas.

- **hadoop jar hadoop-mapreduce-examples-2.10.2.jar wordcount /libros /salida_libros** (salida_libros no es necesario crearlo).

```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-2.10.2.jar wordcount
 /libros /salida_libros
23/02/18 22:07:21 INFO client.RMProxy: Connecting to ResourceManager at localhost
23/02/18 22:07:50 INFO mapreduce.Job:  map 0% reduce 0%
23/02/18 22:08:08 INFO mapreduce.Job:  map 100% reduce 0%
23/02/18 22:08:20 INFO mapreduce.Job:  map 100% reduce 100%
23/02/18 22:08:21 INFO mapreduce.Job: Job job_1676758030441_0001 completed successfully
23/02/18 22:08:21 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=602460
                FILE: Number of bytes written=1625401
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2226146
                HDFS: Number of bytes written=446927
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
```

7. EJEMPLO CON MAPREDUCE

```
                Map output bytes=3740951
                Map output materialized bytes=602460
                Input split bytes=101
                Combine input records=389719
                Combine output records=39791
                Reduce input groups=39791
                Reduce shuffle bytes=602460
                Reduce input records=39791
                Reduce output records=39791
                Spilled Records=79582
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=452
                CPU time spent (ms)=12080
                Physical memory (bytes) snapshot=509894656
                Virtual memory (bytes) snapshot=3772424192
                Total committed heap usage (bytes)=304087040
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2226045
        File Output Format Counters
                Bytes Written=446927
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$
```

7. EJEMPLO CON MAPREDUCE

Paso 5. Observamos el resultado obtenido

- **hdfs dfs –ls /salida_libros** :

```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hdfs dfs -ls /salida_libros
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2023-02-18 22:08 /salida_libros/_SUCCESS
-rw-r--r--   1 hadoop supergroup     446927 2023-02-18 22:08 /salida_libros/part-r-00000
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$
```

- Se han generado dos ficheros:
  - _SUCCESS
  - part-r-00000 → Debe de contener las palabras contadas.

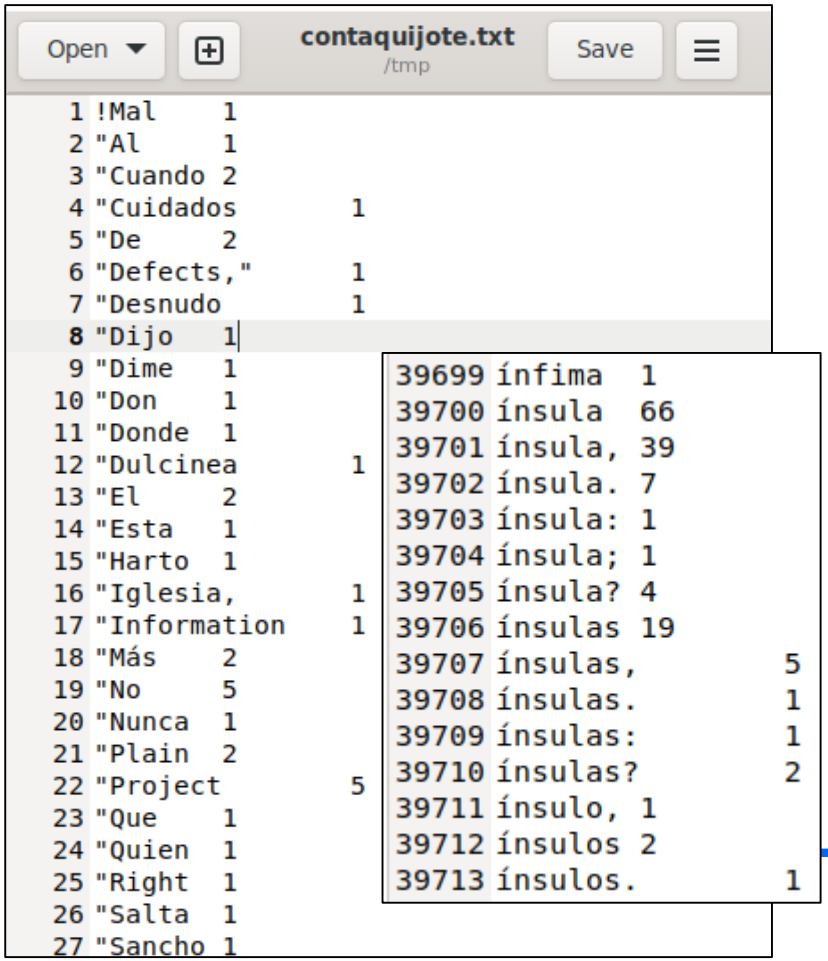
Paso 6. Descargamos el resultado a nuestra maquina local

- **hdfs dfs –get /salida_libros/part-r-0000 /tmp/contaquijote.txt**

```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hdfs dfs -get /salida_libros/part-r-00000
 /tmp/contaquijote.txt
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$
```

7. EJEMPLO CON MAPREDUCE

Paso 7. Abrimos el fichero contaquijote.txt con gedit:
gedit/tmp/contaquijote.txt



- Vemos que el programa wordcount no es muy inteligente: cuenta palabras, pero entiende que algo que empiece por doble comilla o finalice con punto o coma son palabras diferentes.
- El fichero tiene 40000 líneas, que en teoría son el nº de palabras diferentes, pero en realidad hay muchas palabras repetidas. Por ejemplo:
  - ínsula → aparece 66
  - ínsula, → aparece 39
- Wordcount aunque puede ser útil para algunos casos, para contar palabras dentro de un texto no es el más satisfactorio.

7. EJEMPLO CON MAPREDUCE

Paso 8. Si ejecutamos otra vez el programa wordcount:

- **hadoop jar hadoop-mapreduce-examples-2.10.2.jar wordcount /libros /salida_libros** → Encontramos un error

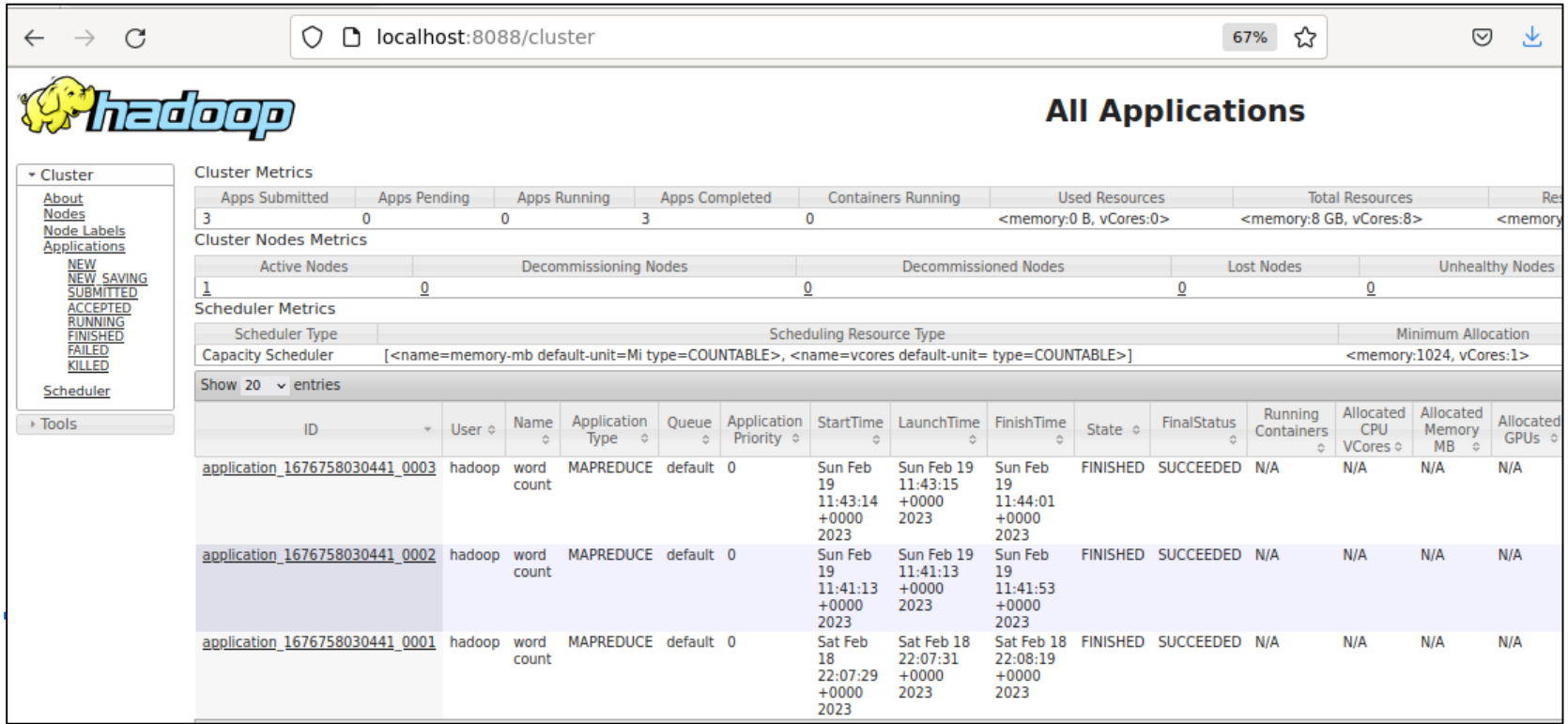
```
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-2.10.2.jar
 wordcount /libros /salida_libros
23/02/19 11:13:55 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory hdfs://nodo1:9000/salida_l
ibros already exists
        at org.apache.hadoop.mapreduce.lib.output.FileOutputFormat.checkOutputSpecs(FileOutputFo
rmat.java:146)
        at org.apache.hadoop.mapreduce.JobSubmitter.checkSpecs(JobSubmitter.java:279)
        at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:145)
```

Debemos borrar el directorio de salida para ejecutarla otra vez:

```
255 hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hdfs dfs -rm -R /salida_libros
Deleted /salida_libros
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hdfs dfs -ls /
Found 5 items
drwxr-xr-x   - hadoop supergroup          0 2023-02-05 20:49 /datos
drwxr-xr-x   - hadoop supergroup          0 2023-02-18 21:24 /libros
drwxr-xr-x   - hadoop supergroup          0 2023-02-05 14:59 /temporal
drwxr-xr-x   - hadoop supergroup          0 2023-02-05 14:53 /temporal1
drwx------   - hadoop supergroup          0 2023-02-18 22:03 /tmp
hadoop@nodo1:/opt/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-2.10.2.jar
 wordcount /libros /salida_libros
23/02/19 11:43:11 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
```

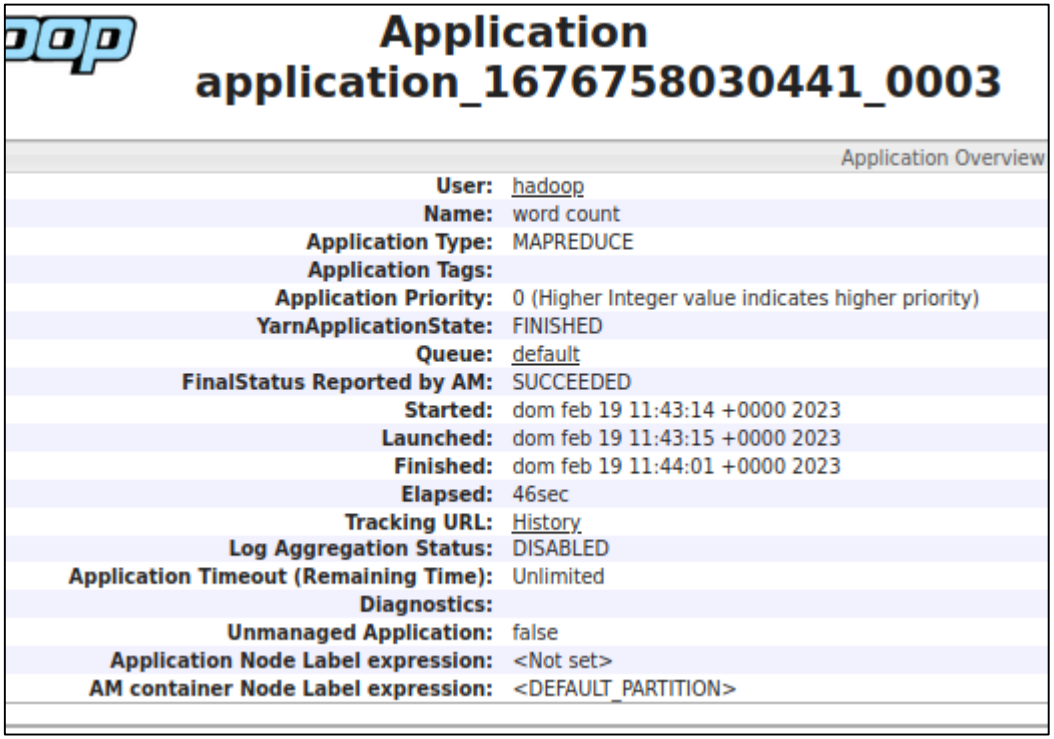
7. EJEMPLO CON MAPREDUCE

Paso 9. Vamos a la pagina web de gestión nodo1:8088 o localhost:8088. Podemos ver todas las aplicaciones que se han lanzado (con su Application Ids), las veces que hemos lanzado el proceso (en nuestro caso 3 veces), que aplicaciones han terminado, cuales han fallado, cuáles se están ejecutando, etc



7. EJEMPLO CON MAPREDUCE

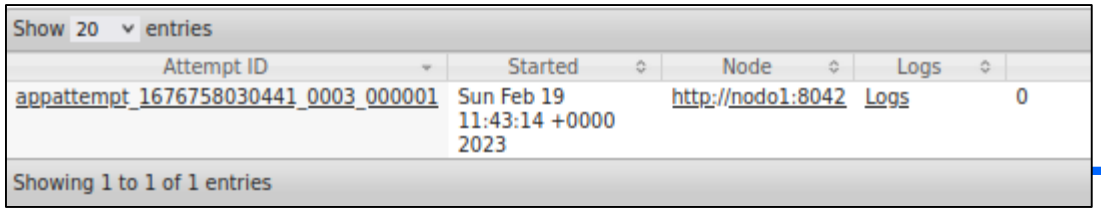
Paso 10. Si seleccionamos en Application_ID vemos lo que ha hecho



Application application_1676758030441_0003

Application Overview

User:	hadoop
Name:	word count
Application Type:	MAPREDUCE
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	dom feb 19 11:43:14 +0000 2023
Launched:	dom feb 19 11:43:15 +0000 2023
Finished:	dom feb 19 11:44:01 +0000 2023
Elapsed:	46sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>



Show 20 entries

Attempt ID	Started	Node	Logs	
appattempt_1676758030441_0003_000001	Sun Feb 19 11:43:14 +0000 2023	http://nodo1:8042	Logs	0

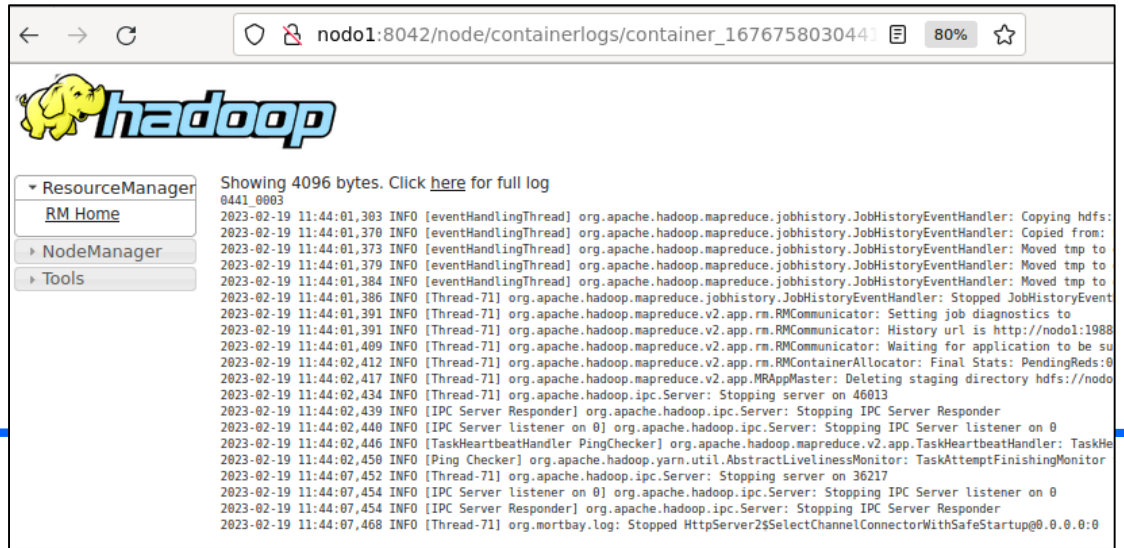
Showing 1 to 1 of 1 entries

- Indica el programa que se ha lanzado: Word count
- Tipo Aplicación: MapReduce.
- Cuánto ha tardado: 46 segundos.
- Cuál ha sido el intento de generar la aplicación.
- El nodo dónde se ha lanzado.
- Los logs

7. EJEMPLO CON MAPREDUCE

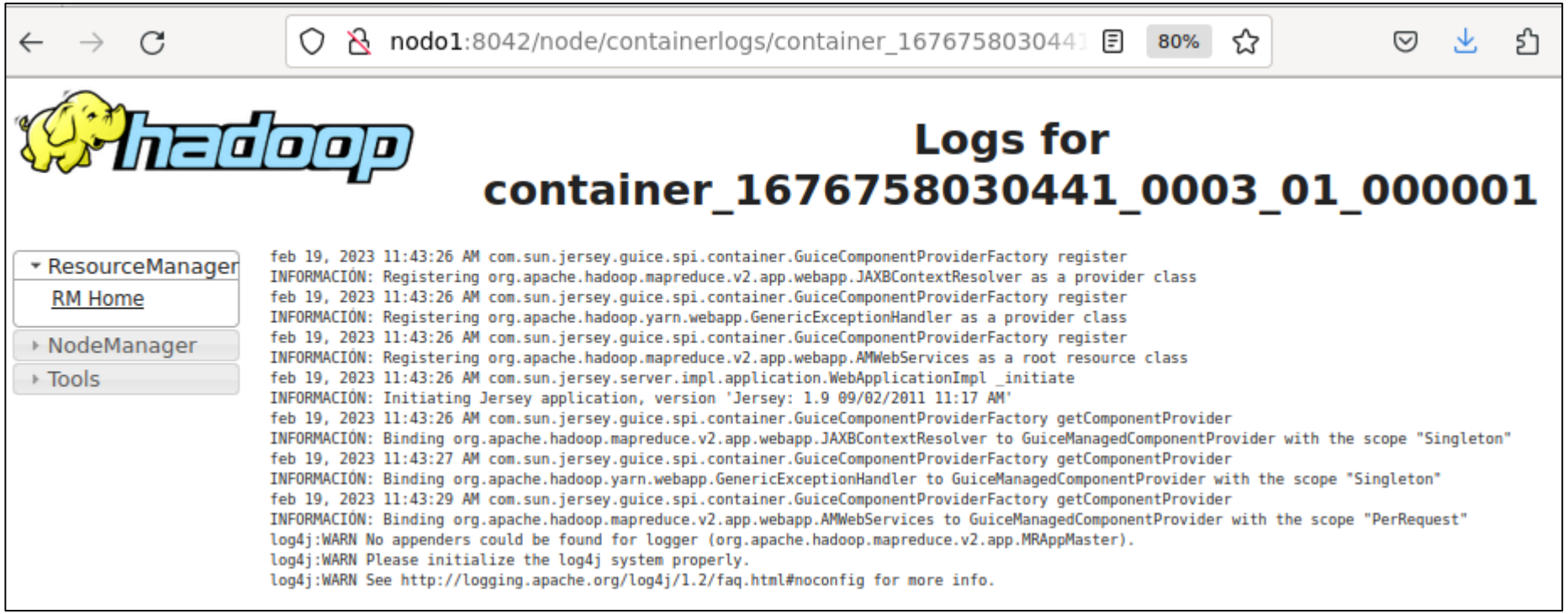
Paso 11. Si vamos a ver los logs para ver un poco cuál ha sido el proceso

- Si vamos al syslog nos dice lo que ha ido haciendo. En este proceso ha estado haciendo procesos de MapReduce



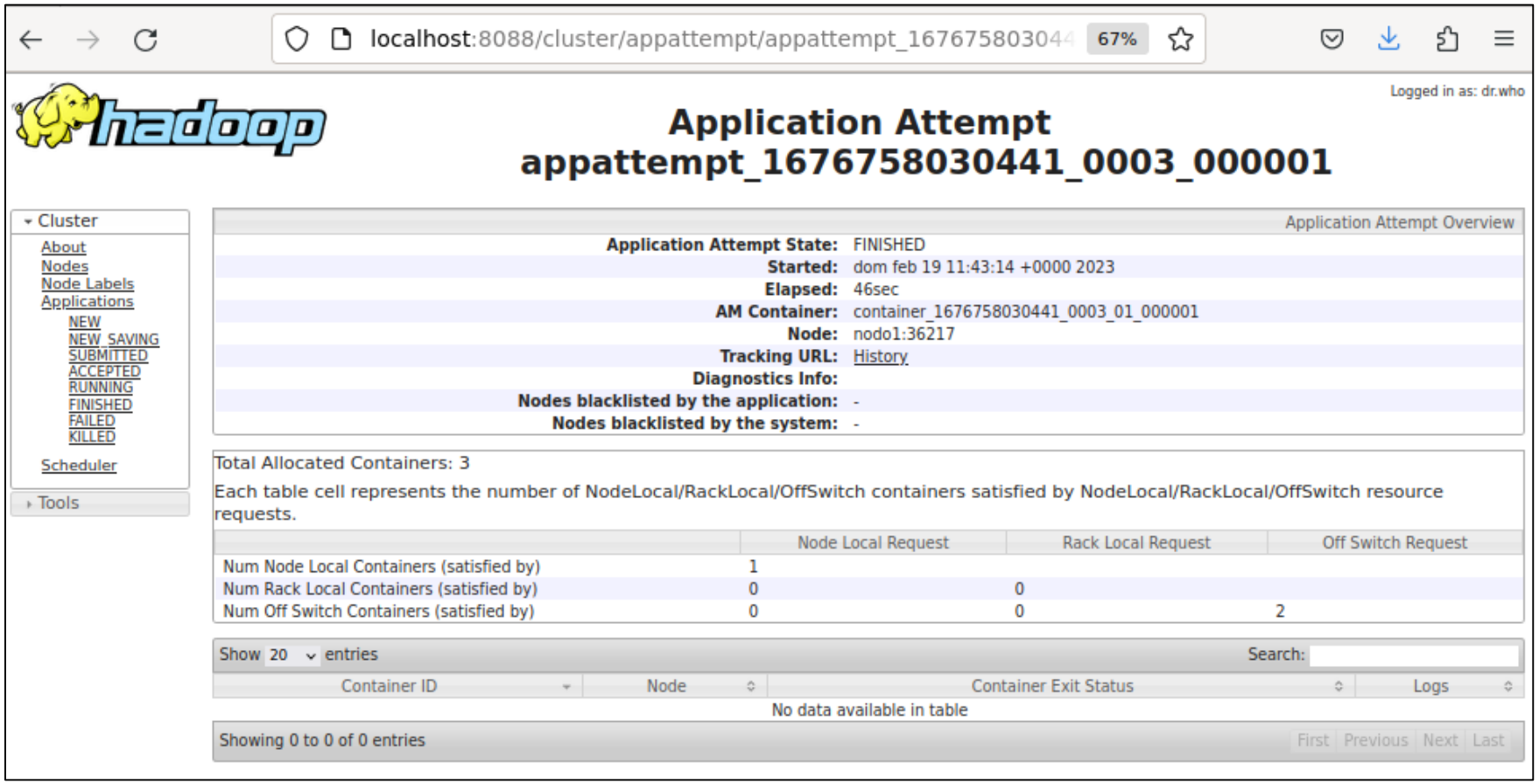
7. EJEMPLO CON MAPREDUCE

Paso 12. Dentro de los logs de stderr. Si hubiéramos tenido algún error pues también nos lo hubiera dicho



7. EJEMPLO CON MAPREDUCE

Paso 13. En el intento, vemos el número de peticiones que se han hecho, el número de containers que se han lanzado

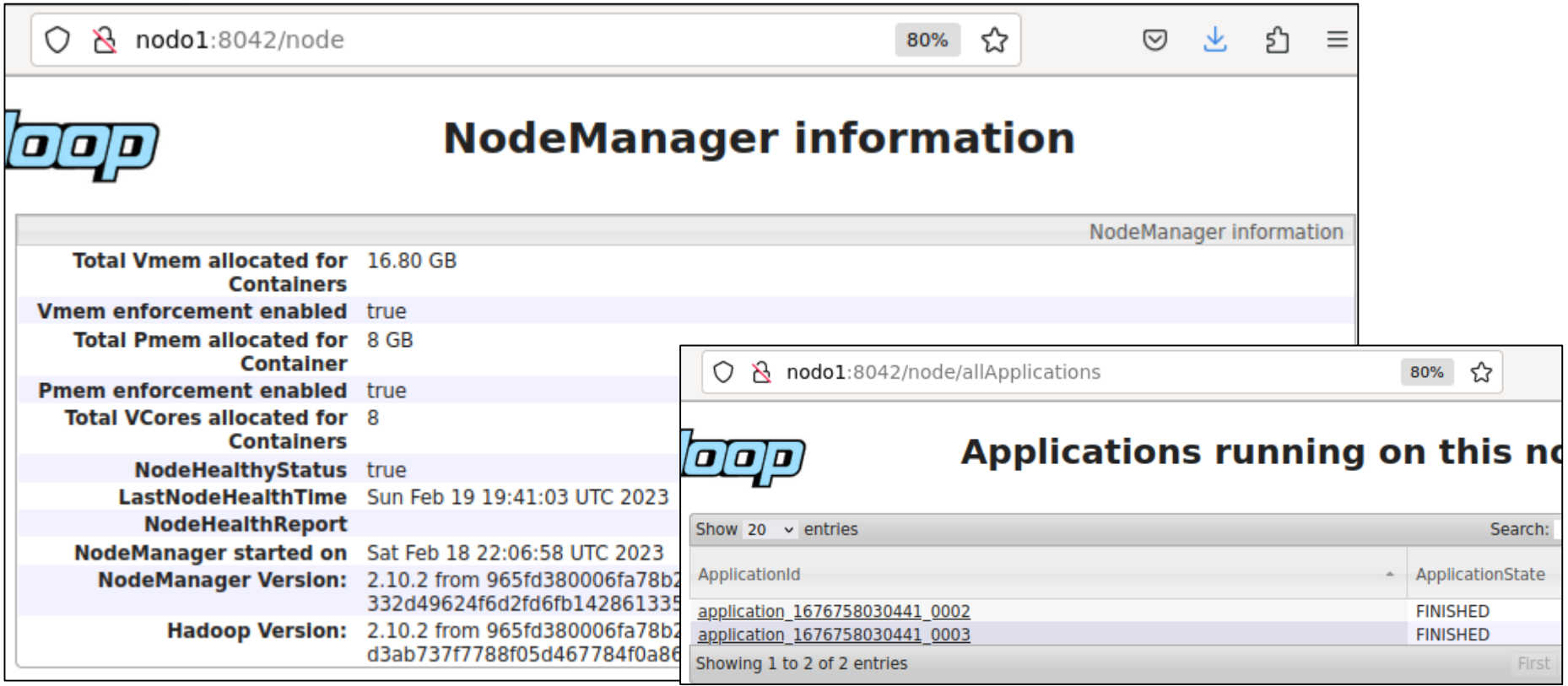


7. EJEMPLO CON MAPREDUCE

Paso 14. Si hacemos click en History nos da un error porque no tenemos el histórico arrancado. Nos permite ver comandos que se han lanzado hace tiempo..

7. EJEMPLO CON MAPREDUCE

Paso 15. Si hacemos click en http://nodo1:8042, podemos ver que se ha lanzado en este nodo. En **List of Applications** vemos las aplicaciones que se han lanzado en este nodo y cómo han terminado. Como ya han terminado no tenemos en este momento ningún otro tipo de información.



7. EJEMPLO CON MAPREDUCE

Paso 16. En **List of Containers** podemos ver la lista de contenedores. Vemos que ya no hay ninguno pero que en su momento los debió haber cuando se lanzó la aplicación.

8. MAPREDUCE DESDE JAVA

Paso 1. Compilaremos el código Java de un programa ContarPalabras.java, lanzaremos su proceso contra el fichero access_log, que tiene un tamaño elevado. Y por ultimo en la pagina de administración de Yarn veremos como ha ido lanzando los mapers y los reducers.

Primero descargamos el programa y lo copiamos en el directorio practicas, dentro de /home/hadoop

```
1 hadoop@nodo1:~$ cd Downloads/
hadoop@nodo1:~/Downloads$ pwd
/home/hadoop/Downloads
hadoop@nodo1:~/Downloads$ ls
ContarPalabras.java  hadoop-2.10.2.tar.gz  hadoop-3.3.4.tar.gz
hadoop@nodo1:~/Downloads$ cd
hadoop@nodo1:~$ mkdir practicas
hadoop@nodo1:~$ cp Do
Documents/ Downloads/
hadoop@nodo1:~$ cp Downloads/ContarPalabras.java practicas/
hadoop@nodo1:~$ cd practicas
hadoop@nodo1:~/practicas$ ls
ContarPalabras.java
hadoop@nodo1:~/practicas$
```

8. MAPREDUCE DESDE JAVA

Paso 2. En ContarPalabras.java tenemos :

- Un objeto Mapper que extiende la clase org.apache.hadoop.mapreduce.Mapper, que es la que va a contar las palabras y por cada una que encuentre suma un 1.
- Al ejecutar Hadoop, se recibe cada línea del fichero de entrada como input. La función map devuelve por cada palabra (word) un (word,1) como salida.

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

8. MAPREDUCE DESDE JAVA

Paso 3. La función reduce recibe todos los valores que tienen la misma clave como entrada y devuelve la clave y el número de ocurrencias como salida

```
public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

8. MAPREDUCE DESDE JAVA

Paso 4. La clase main que lo único que hace es recibir un fichero y generar una salida de acuerdo.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Uso: ContarPalabras <in> <out>");
        System.exit(2);
    }
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(ContarPalabras.class);
    job.setMapperClass(TokenizerMapper.class);

    /**** Dejarlo tal cual ****/
    // job.setCombinerClass(IntSumReducer.class);

    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

8. MAPREDUCE DESDE JAVA

Paso 5. Compilaremos este fichero a manualmente con Hadoop:

a) update-alternatives –list java → donde esta el JDK de java

b) export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar → para que Hadoop sepa encontrar el JDK

c) hadoop com.sun.tolos.javac.Main ContarPalabras.java

d) Deberia generar 3 clases uno para el main, otro para el Reducer y otro para el Mapper.

```
hadoop@nodo1:~/practicas$ update-alternatives --list java
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
hadoop@nodo1:~/practicas$ export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
hadoop@nodo1:~/practicas$ hadoop com.sun.tools.javac.Main ContarPalabras.java
hadoop@nodo1:~/practicas$ ls -l
total 16
-rw-rw-r-- 1 hadoop hadoop 1754 feb 20 08:34 'ContarPalabras$IntSumReducer.class'
-rw-rw-r-- 1 hadoop hadoop 1751 feb 20 08:34 'ContarPalabras$TokenizerMapper.class'
-rw-rw-r-- 1 hadoop hadoop 1846 feb 20 08:34  ContarPalabras.class
-rw-rw-r-- 1 hadoop hadoop 2934 feb 20 08:31  ContarPalabras.java
hadoop@nodo1:~/practicas$
```

8. MAPREDUCE DESDE JAVA

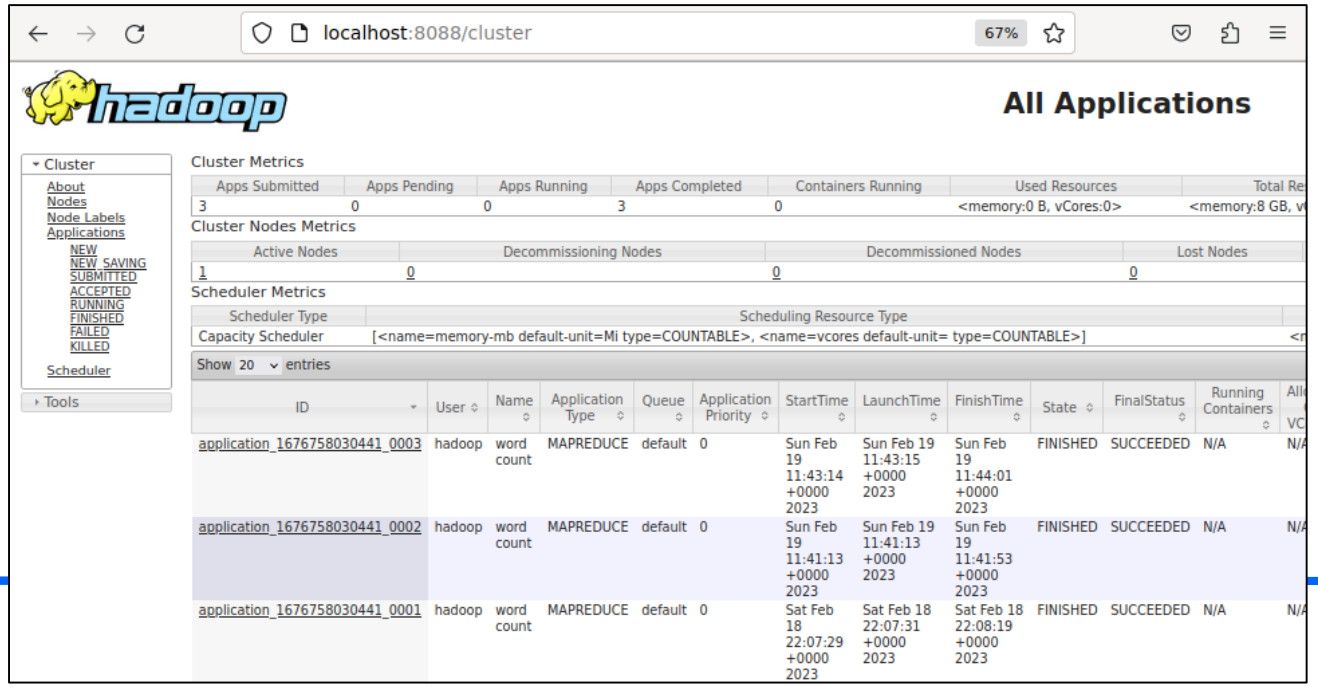
Paso 6. Hadoop solo admite ficheros JAR. Creamos un fichero jar que contenga los class generados

```
hadoop@nodo1:~/practicas$ jar cf ContarPalabras.jar Contar*.class
hadoop@nodo1:~/practicas$ ls -l
total 20
-rw-rw-r-- 1 hadoop hadoop 1754 feb 20 08:34 'ContarPalabras$IntSumReducer.class'
-rw-rw-r-- 1 hadoop hadoop 1751 feb 20 08:34 'ContarPalabras$TokenizerMapper.class'
-rw-rw-r-- 1 hadoop hadoop 1846 feb 20 08:34  ContarPalabras.class
-rw-rw-r-- 1 hadoop hadoop 3276 feb 20 09:08  ContarPalabras.jar
-rw-rw-r-- 1 hadoop hadoop 2934 feb 20 08:31  ContarPalabras.java
hadoop@nodo1:~/practicas$
```

8. MAPREDUCE DESDE JAVA

Paso 7. Antes de ejecutarlo debemos hacer dos pasos.

- Asegurarnos de que la página web desde la que se gestionan las aplicaciones y nodos, esta abierta para ver cómo se lanza
- Activar el proceso History Server, que nos permite crear un histórico con el cual luego poder buscar posibles problemas o todos los pasos que se han ido realizando a lo largo de la ejecución



8. MAPREDUCE DESDE JAVA

Paso 8. Para activar el history server usaremos el Shell- script mr-jobhistory-daemon.sh que se encuentra en el directorio /opt/hadoop/sbin

mr-jobhistory-daemon.sh start historyserver

De esta manera cuando lancemos una aplicación, pinchando en Application ID y en History tenemos un histórico de cuántos procesos Mapper ha ejecutado

```
hadoop@nodo1:~/practicas$ ls /opt/hadoop/sbin/
distribute-exclude.sh  mr-jobhistory-daemon.sh  start-secure-dns.sh  stop-secure-dns.sh
FederationStateStore   refresh-namenodes.sh     start-yarn.cmd       stop-yarn.cmd
hadoop-daemon.sh       slaves.sh                start-yarn.sh        stop-yarn.sh
hadoop-daemons.sh      start-all.cmd            stop-all.cmd         yarn-daemon.sh
hdfs-config.cmd        start-all.sh             stop-all.sh          yarn-daemons.sh
hdfs-config.sh         start-balancer.sh        stop-balancer.sh
httpfs.sh              start-dfs.cmd            stop-dfs.cmd
kms.sh                 start-dfs.sh             stop-dfs.sh
hadoop@nodo1:~/practicas$
```

8. MAPREDUCE DESDE JAVA

Paso 9. Lanzamos el proceso java:

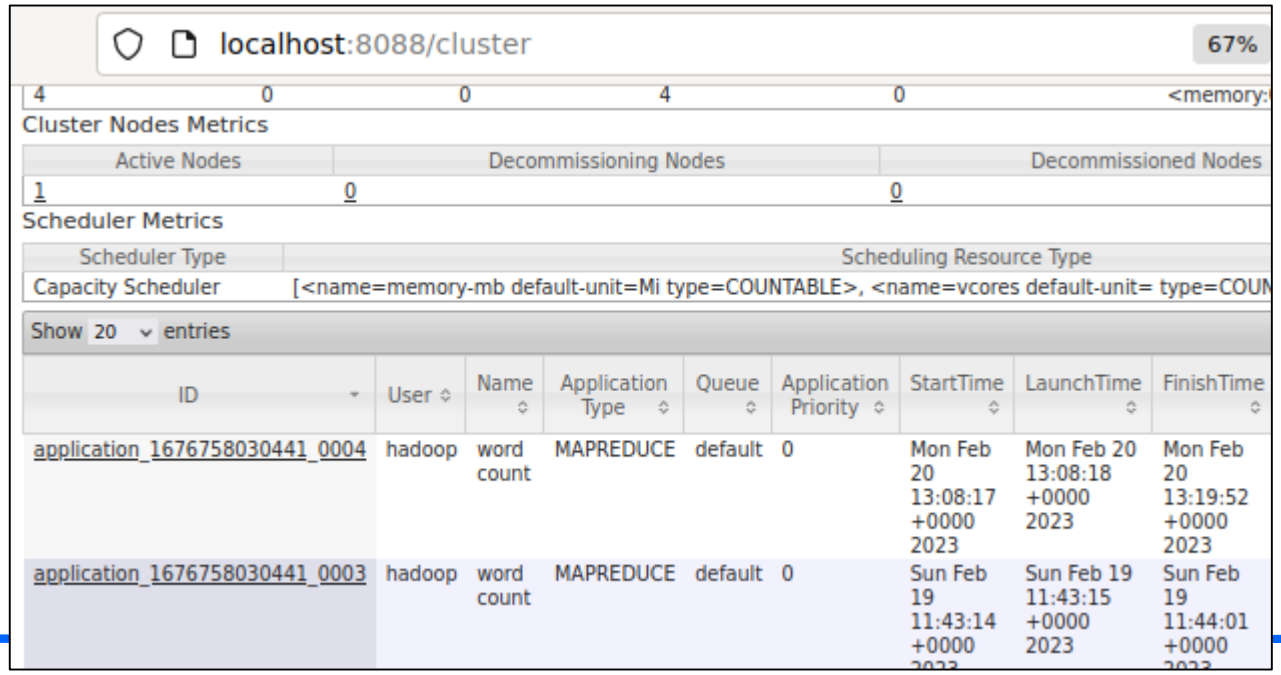
hadoop jar ContarPalabras.jar ContarPalabras /temporal/access_log /salida_java

Vemos que se conecta con el ResourceManager. Y lanza un job que ejecutará el proceso. Empieza a hacer los procesos Mappers. Cuando termine con estos, hará los procesos Reduce

```
hadoop@nodo1:~/practicas$ hdfs dfs -ls /temporal
Found 1 items
-rw-r--r--   1 hadoop supergroup  504941532 2023-02-05 13:50 /temporal/access_log
hadoop@nodo1:~/practicas$ hadoop jar ContarPalabras.jar ContarPalabras /temporal/access_log /salida_java
23/02/20 13:08:13 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
23/02/20 13:08:15 INFO input.FileInputFormat: Total input files to process : 1
23/02/20 13:08:15 INFO mapreduce.JobSubmitter: number of splits:4
23/02/20 13:08:16 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1676758030441_0004
23/02/20 13:08:17 INFO conf.Configuration: resource-types.xml not found
23/02/20 13:08:17 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
23/02/20 13:08:17 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
23/02/20 13:08:17 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
23/02/20 13:08:17 INFO impl.YarnClientImpl: Submitted application application_1676758030441_0004
23/02/20 13:08:17 INFO mapreduce.Job: The url to track the job: http://nodo1:8088/proxy/application_1676758030441_0004/
23/02/20 13:08:17 INFO mapreduce.Job: Running job: job_1676758030441_0004
23/02/20 13:08:35 INFO mapreduce.Job: Job job_1676758030441_0004 running in uber mode : false
23/02/20 13:08:35 INFO mapreduce.Job:  map 0% reduce 0%
23/02/20 13:09:51 INFO mapreduce.Job:  map 8% reduce 0%
23/02/20 13:09:57 INFO mapreduce.Job:  map 17% reduce 0%
23/02/20 13:09:58 INFO mapreduce.Job:  map 20% reduce 0%
23/02/20 13:10:30 INFO mapreduce.Job:  map 24% reduce 0%
23/02/20 13:10:36 INFO mapreduce.Job:  map 34% reduce 0%
```

8. MAPREDUCE DESDE JAVA

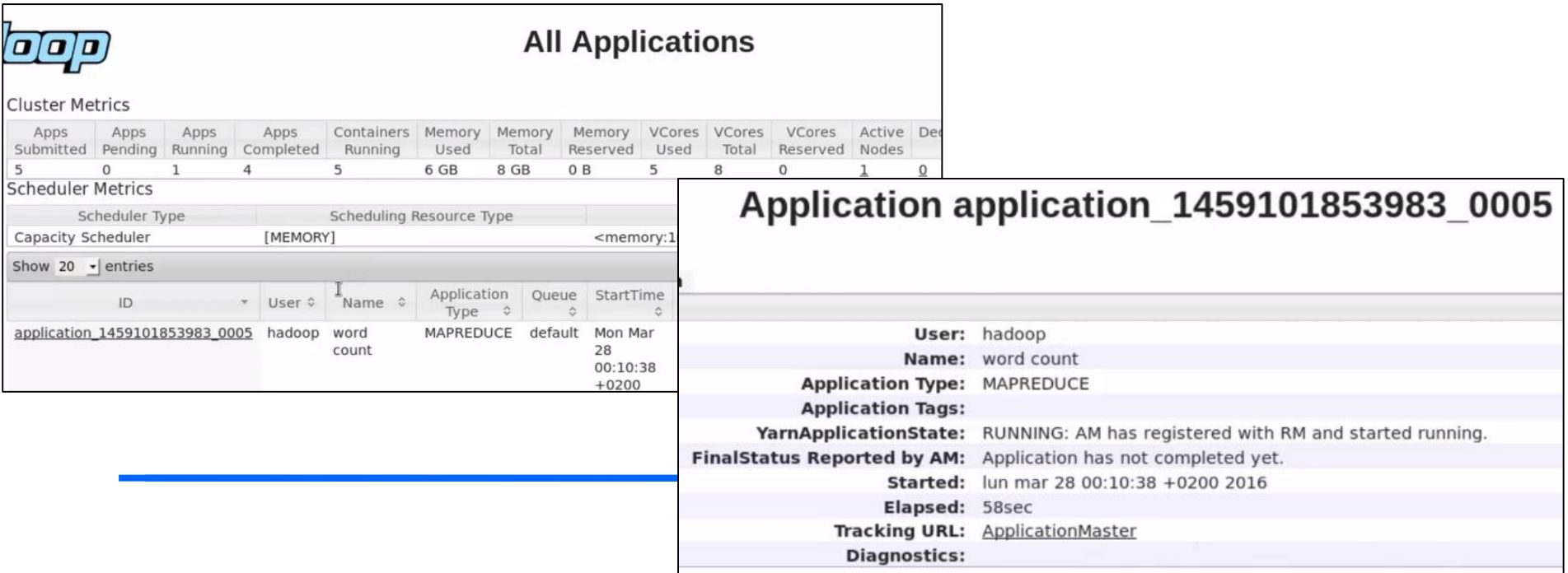
Paso 10. Si ahora refrescamos la pagina web y podemos comprobar que se ha lanzado el programa java ContarPalabras que tiene la etiqueta Word count. Como los procesos empiezan a ser bastante pesados (se lanzan varios mappers de 1GB cada uno), el refresco de la web puede costar. Teníamos que haber puesto un poquito más de recursos.



8. MAPREDUCE DESDE JAVA

Paso 11. Si tuviéramos recursos suficientes podríamos ver en la web que la aplicación esta RUNNING. Si hacemos click en Aplication ID se ve que ha arrancado una ApplicatiónMaster, es el proceso general con el que trabajaba MapReduce

Con Yarn teníamos el ResorceManager el NodeManager y el ApplicationMaster que se arrancaba por cada aplicación que se lanzaba



8. MAPREDUCE DESDE JAVA

Paso 12. Esperamos que termine.

```
23/02/20 13:13:51 INFO mapreduce.Job:  map 76% reduce 0%
23/02/20 13:13:56 INFO mapreduce.Job:  map 77% reduce 0%
23/02/20 13:13:59 INFO mapreduce.Job:  map 78% reduce 0%
23/02/20 13:14:08 INFO mapreduce.Job:  map 79% reduce 0%
                Combine output records=0
                Reduce input groups=2475645
                Reduce shuffle bytes=773627690
                Reduce input records=44778500
                Reduce output records=2475645
                Spilled Records=134335500
                Shuffled Maps =4
                Failed Shuffles=0
                Merged Map outputs=4
                GC time elapsed (ms)=9893
                CPU time spent (ms)=375970
                Physical memory (bytes) snapshot=1289498624
                Virtual memory (bytes) snapshot=9408188416
                Total committed heap usage (bytes)=901251072
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=504953820
        File Output Format Counters
                Bytes Written=57779668
[1]+  Terminated              firefox
hadoop@nodo1:~/practicas$
```

Primero hace los Mappers y después los Reduce. Al final obtenemos un resumen:

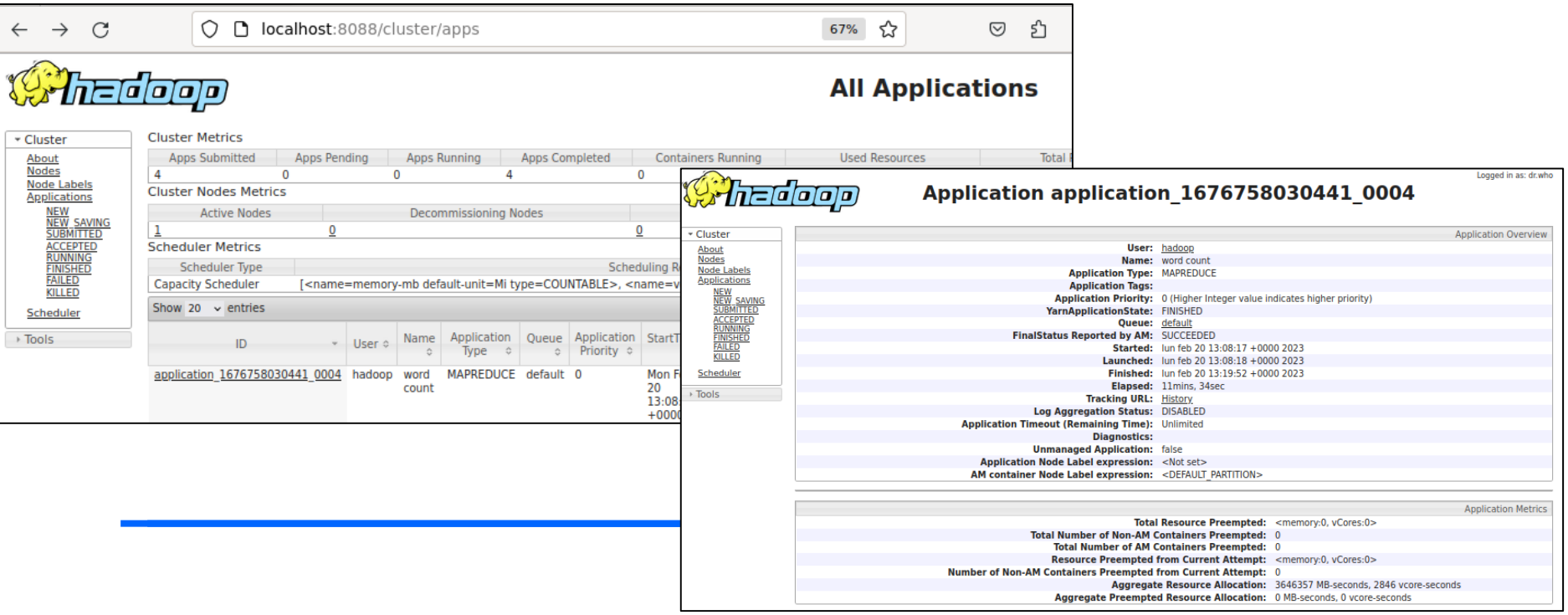
- nº de Jobs que ha hecho
- nº bytes leidos de los ficheros
- 4 mappers lanzados y 1 Reducer

Ha lanzado 4 procesos que se corresponden con el nº de bloques que tiene nuestro fichero

9. HISTORY

Paso 1. En nodo1:8088 observamos todas las aplicaciones lanzadas en Hadoop. Hacemos click en la Application ID que queremos comprobar.

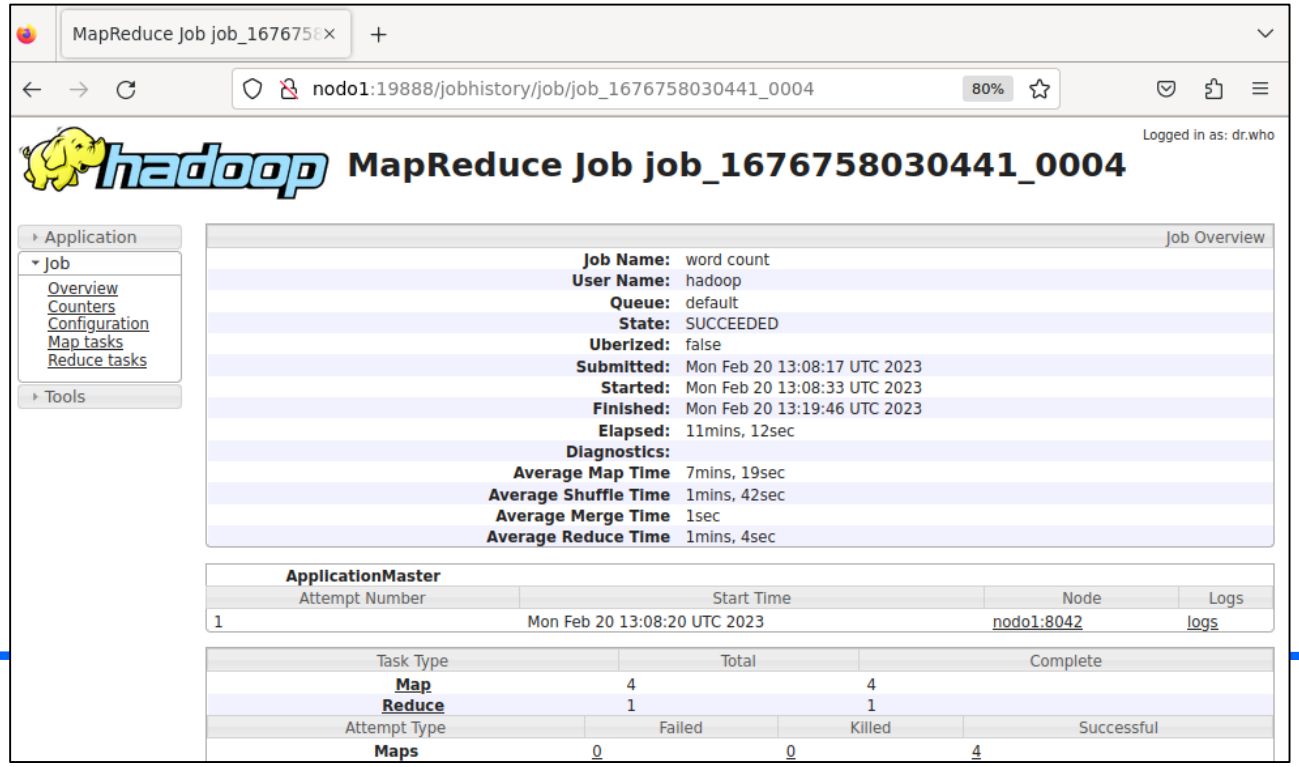
En Tracking URL tenemos el historial de los procesos que se han lanzado de esta aplicación. Si no hubiéramos puesto lo del History Server al pinchar aquí no saldría nada, y no podría ver los procesos que se han hecho



9. HISTORY

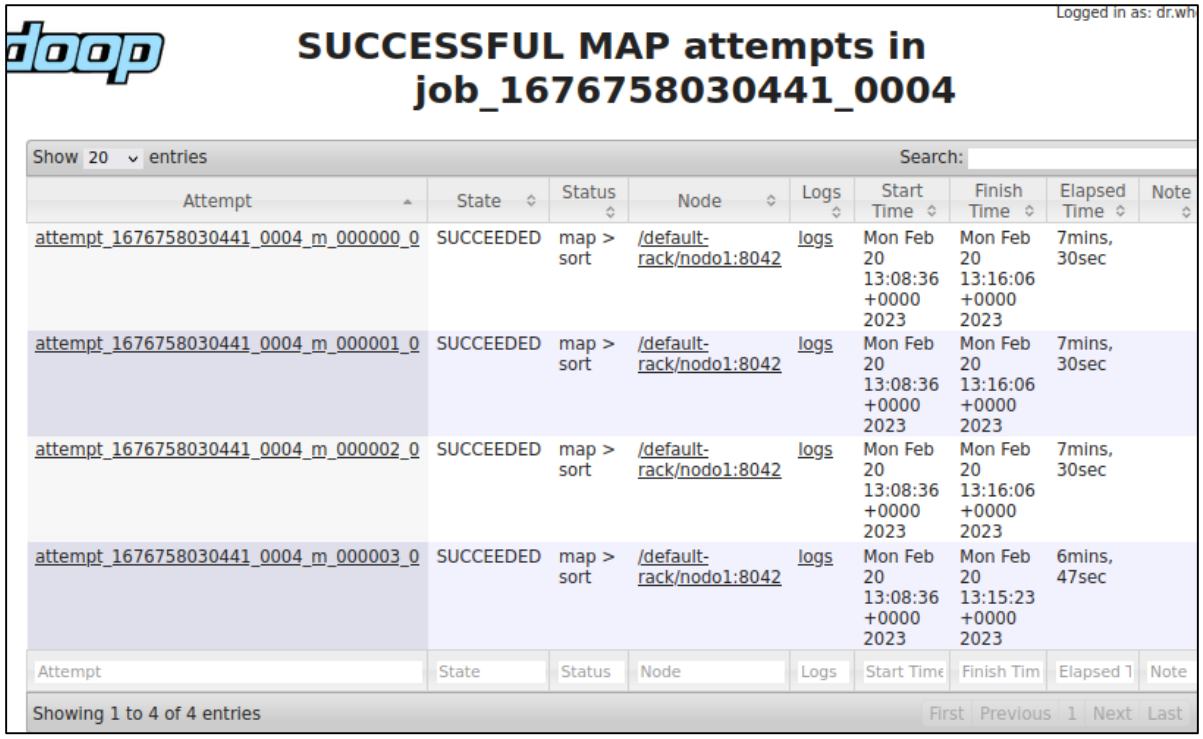
Paso 2. En la pagina de History, podemos ver los contadores realizados, los mappers y los reducers que se han hecho, etc

En Overview vemos que el resultado wordcount ha sido satisfactorio. El proceso ha utilizado 4 procesos mappers y un procesos reducer. También nos dice si hubiera alguno fallido o eliminado.



9. HISTORY

Paso 3. Si hacemos click en los satisfactorios (en el link del 4). Podemos ver los 4 map que se han hecho con sort y además en los nodos donde se ha ejecutado



Logged in as: dr.wh

SUCCESSFUL MAP attempts in job_1676758030441_0004

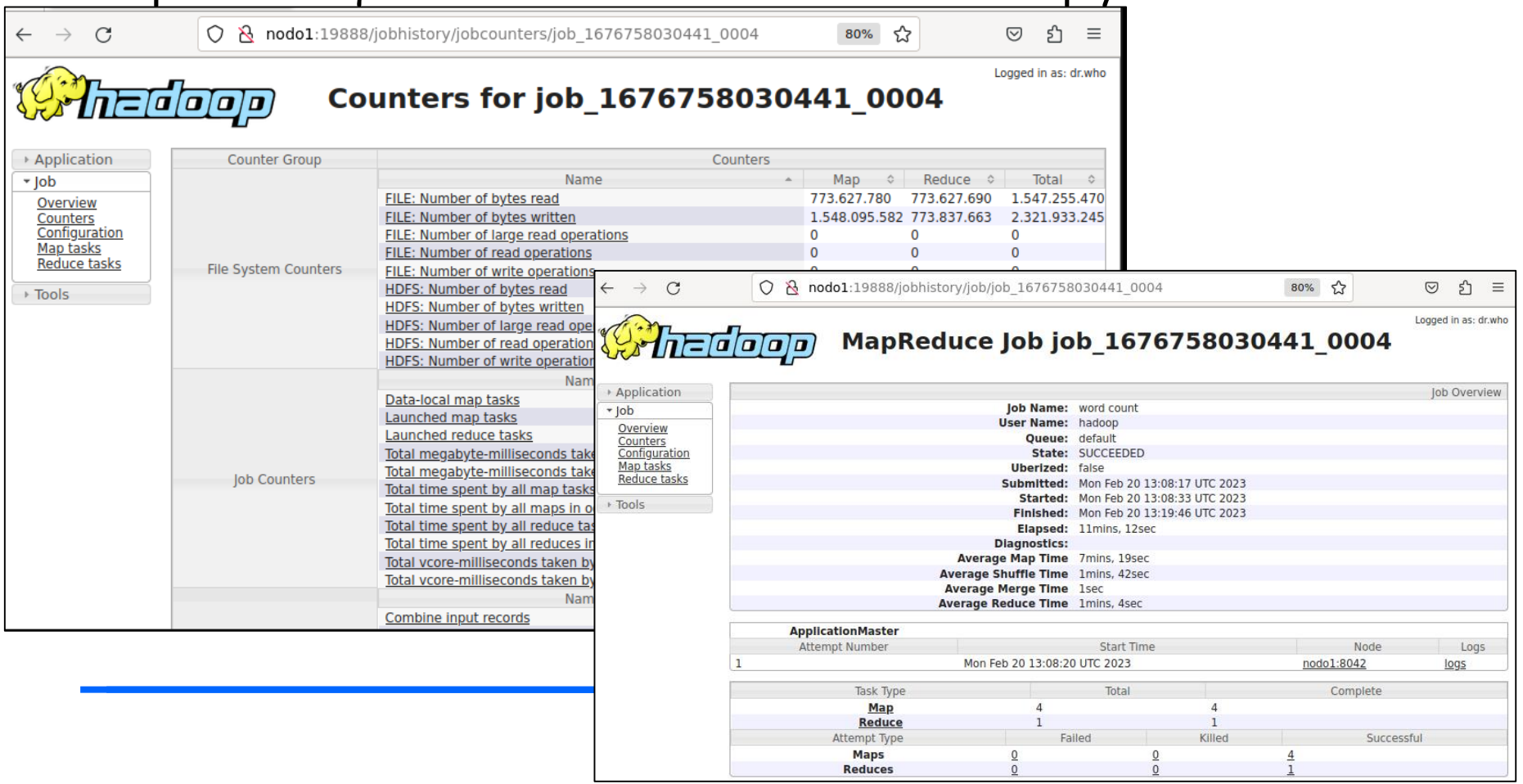
Show 20 entries Search:

Attempt	State	Status	Node	Logs	Start Time	Finish Time	Elapsed Time	Note
attempt_1676758030441_0004_m_000000_0	SUCCEEDED	map > sort	/default-rack/nodo1:8042	logs	Mon Feb 20 13:08:36 +0000 2023	Mon Feb 20 13:16:06 +0000 2023	7mins, 30sec	
attempt_1676758030441_0004_m_000001_0	SUCCEEDED	map > sort	/default-rack/nodo1:8042	logs	Mon Feb 20 13:08:36 +0000 2023	Mon Feb 20 13:16:06 +0000 2023	7mins, 30sec	
attempt_1676758030441_0004_m_000002_0	SUCCEEDED	map > sort	/default-rack/nodo1:8042	logs	Mon Feb 20 13:08:36 +0000 2023	Mon Feb 20 13:16:06 +0000 2023	7mins, 30sec	
attempt_1676758030441_0004_m_000003_0	SUCCEEDED	map > sort	/default-rack/nodo1:8042	logs	Mon Feb 20 13:08:36 +0000 2023	Mon Feb 20 13:15:23 +0000 2023	6mins, 47sec	

Showing 1 to 4 of 4 entries First Previous 1 Next Last

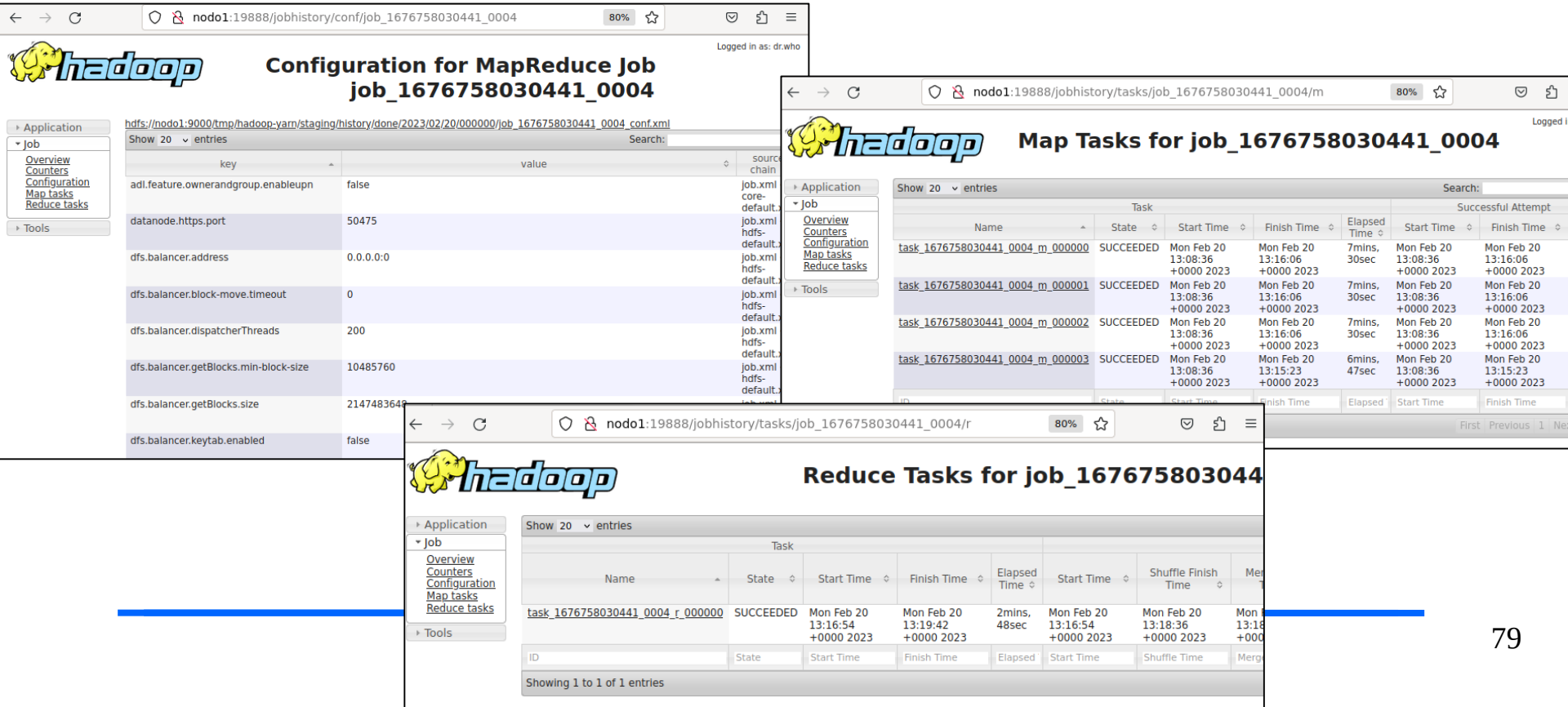
9. HISTORY

Paso 4. Si vamos al menú Job, en Overview nos indica el resumen de lo que se ha hecho. En counters podemos ver los contadores o todos los procesos que se han lanzado en la fase de map y en la fase Reduce



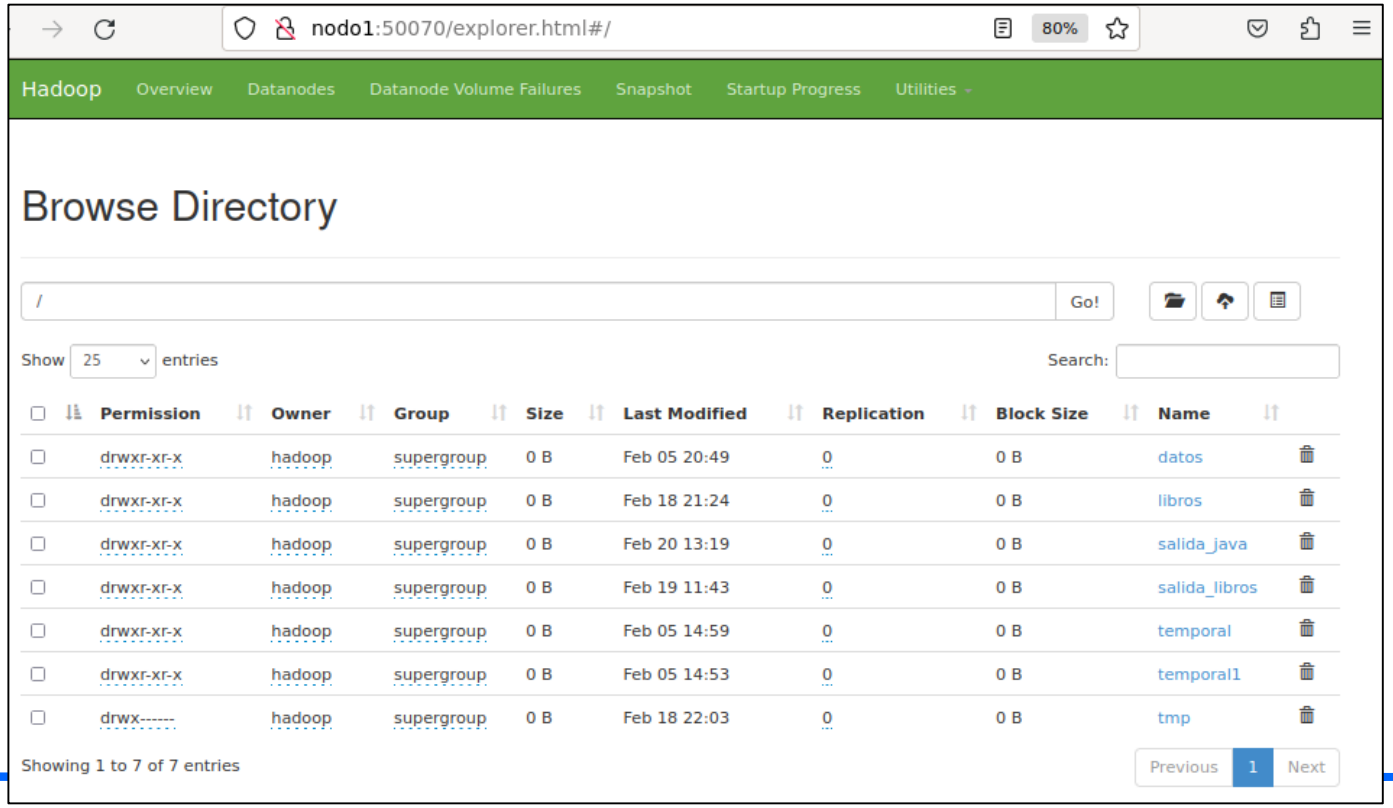
9. HISTORY

Paso 5. En **Configuration** vemos la configuración que se ha utilizado, las tareas Maps que se han utilizado y las tareas ReducersSi se hace click en una tarea ya sea de tipo mapa o de tipo Radius podéis averiguar en qué nodo se han ejecutado



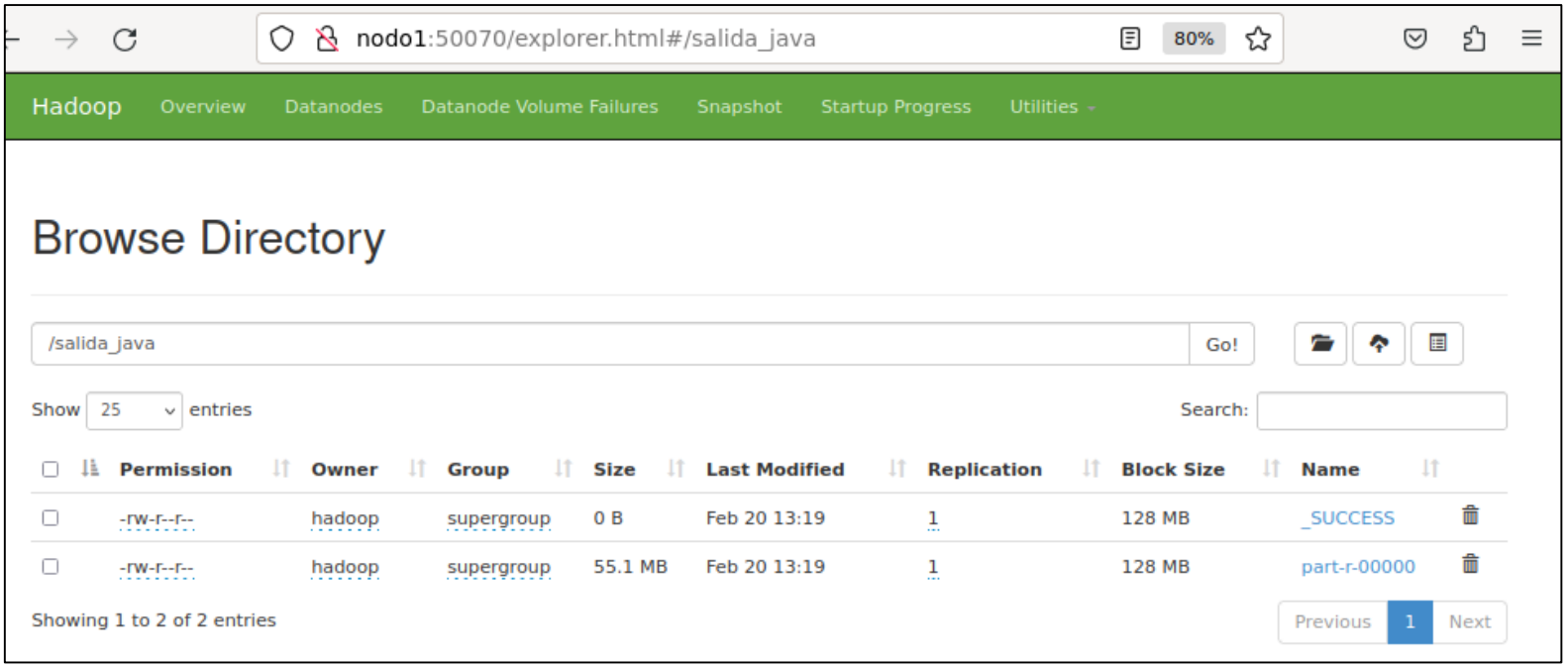
9. HISTORY

Paso 6. Si vamos del nodo1:8088 al nodo1:50070, entramos en la web de HDFS y dentro de Utilities/Browse the file system, podemos ver el directorio de salida salida_java



9. HISTORY

Paso 7. En /salida_java habrá dejado la información del número de veces que aparece cada palabra




```
130 hadoop@nodo1:~/practicas$ hdfs dfs -ls /salida_java
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2023-02-20 13:19 /salida_java/_SUCCESS
-rw-r--r--   1 hadoop supergroup   57779668 2023-02-20 13:19 /salida_java/part-r-00000
hadoop@nodo1:~/practicas$ hdfs dfs -get /salida_java/part-r-00000 /tmp/salida_java.txt
hadoop@nodo1:~/practicas$
```