
BIG DATA ZOOKEEPER

EDUARD LARA



1. INTRODUCCION

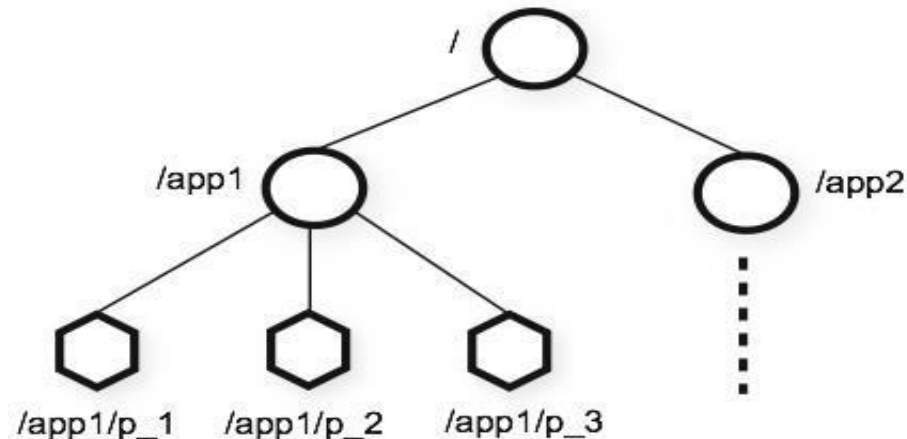
- ❑ ZooKeeper es otro producto de Apache, anterior a Hadoop. Su funcionalidad le hace muy útil dentro de Hadoop Big Data.
- ❑ Nos permite poner nuestro Cluster en alta disponibilidad
- ❑ Nos permite tener un servicio centralizado para:
 - ❑ Gestionar una configuración centralizada y coordinada de entornos distribuidos, no sólo Hadoop
 - ❑ Sincronización distribuida, de transacciones o configuraciones.
 - ❑ Gestión de nombres
 - ❑ Agrupaciones de servicios de forma común
 - ❑ Zookeeper es el producto mas adecuado para un entorno distribuido de procesamiento distribuido

1. INTRODUCCION

- ❑ Expone un conjunto simple de primitivas y variables que las aplicaciones distribuidas pueden UTILIZAR para implementar servicios de sincronización, mantenimiento de configuraciones, grupos, etc.
- ❑ Utiliza un modelo de datos basado en la estructura jerárquica de directorios.
- ❑ Funciona en Java. En la instalación veremos unos cuantos servidores en Java

1. INTRODUCCION

- ❑ ZooKeeper permite que los procesos distribuidos se coordinen entre sí a través de un espacio de nombres jerárquico centralizado que está organizado de forma similar a un sistema de archivos estándar.
- ❑ Distintos productos pueden mantener dentro de esta estructura jerárquica, su propia configuración y su propio entorno

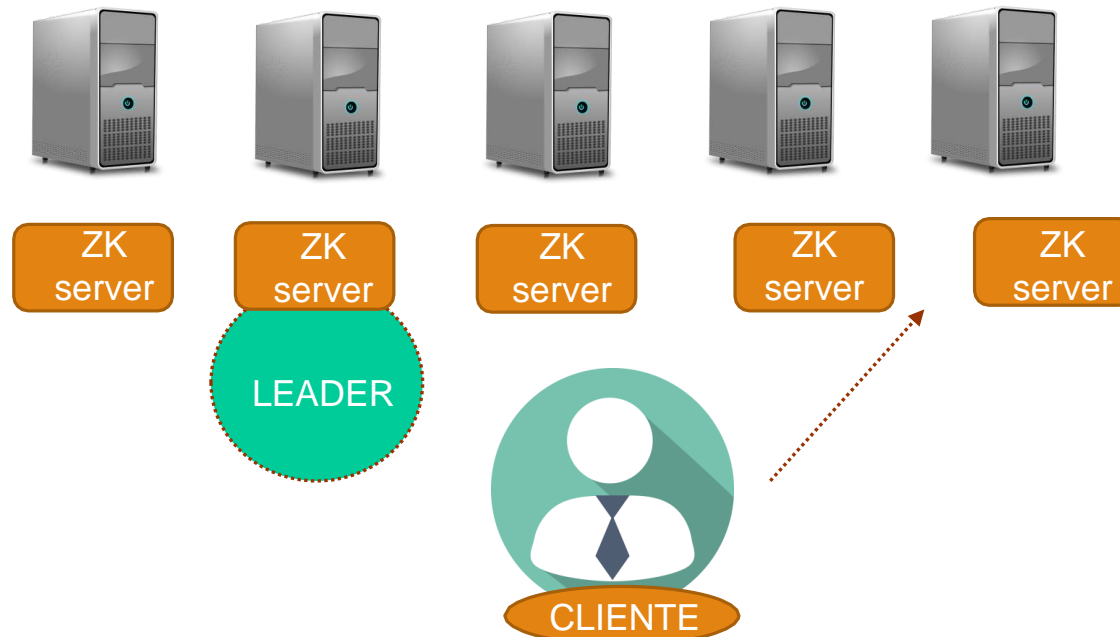


1. INTRODUCCION

- ❑ Los espacios de nombres o named spaces que están funcionando dentro de ZooKeeper consisten en registros de datos y se conocen como **znodes**.
- ❑ Estos nodos son similares a archivos y directorios y mantienen una nomenclatura que permite que los distintos servicios que estén dentro del mismo znode funcionan de manera cooperativa.
- ❑ Zookeeper mantiene todo en memoria, guardando sus datos en memoria, al contrario al contrario que otros productos que trabajan con una estructura en disco.
- ❑ Está muy orientado a grandes sistemas distribuidos donde el rendimiento es muy importante.
- ❑ Las características de alta disponibilidad impiden que haya un único punto de fallo.

1. INTRODUCCION

- ❑ Vamos a tener un conjunto de servidores ZooKeeper distribuidos a lo largo de nuestro clúster.
- ❑ Estos van a mantener una estructura jerárquica exactamente igual en todos los nodos, de manera que una cosa una cosa que se produce en un servidor es automáticamente replicado/distribuido al resto





1. INTRODUCCION


- ❑ Entre todos los Zookeepers que tengamos en un cluster se elige un líder. Maestro no es una palabra adecuada porque pero es un entorno maestro/esclavo.
- ❑ El líder que representa el punto de gestión centralizada
- ❑ Si se cae este servidor, automáticamente otro servidor del Clúster se convierte en el líder.
- ❑ El cliente se conecta a un servidor, hace su función y el Servicio Distribuido va a permitir mantener una réplica común en todos los nodos
- ❑ En Hadoop, vamos a conseguir que las configuraciones estén sincronizadas entre todos los servidores.
- ❑ Antes cada vez que tocábamos en algún sitio teníamos que copiar la configuración al resto de máquinas lo cual es pesado y nos puede llevar a errores

1. INTRODUCCION

- ❑ Un entorno Zookeeper consta de tres servidores que suelen ser suficientes para mantener la gestión de información y de configuración centralizada (no es excesivamente grande la información de configuración)
- ❑ Entre ellos se nombran un leader.
- ❑ Todas las operaciones se identifican con un id de transacción secuencial denominada zxid, lo que permite mantener una política común en los servicios Zookeeper
- ❑ Los nodos Zookeeper se implementan en números impares: 3,5,7... nodos, porque tiene que existir una mayoría para poder continuar
- ❑ Implementaremos Zookeeper en hadoop para mantener la alta disponibilidad: cuando uno de los nodos maestros se caiga, automáticamente otro entre en marcha.

2. INSTALACION ZOOKEEPER

Paso 1. Vamos a la página web de descarga de zookeeper
<https://zookeeper.apache.org/releases.html>

**Apache ZooKeeper™**

Welcome to Apache ZooKeeper™

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination.

What is ZooKeeper?

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Learn more about ZooKeeper on the [ZooKeeper Wiki](#).

Getting Started

Start by installing ZooKeeper on a single machine or a very small cluster.

1. [Learn about ZooKeeper](#) by reading the documentation.
2. [Download ZooKeeper](#) from the release page.

Getting Involved

Apache ZooKeeper is an open source volunteer project under the Apache Software Foundation. We encourage you to learn about the project and contribute your expertise. Here are some starter links:

**Apache ZooKeeper™**

2. INSTALACION ZOOKEEPER

Paso 2. Descargamos la versión 3.8.3. la ultima estable
Nos bajamos el fichero tar.gz

Apache ZooKeeper 3.8.3 (latest stable release)

[Apache ZooKeeper 3.8.3\(asc, sha512\)](#)

[Apache ZooKeeper 3.8.3 Source Release\(asc, sha512\)](#)



We suggest the following location for your download:

<https://dlcdn.apache.org/zookeeper/zookeeper-3.8.3/apache-zookeeper-3.8.3-bin.tar.gz>

Alternate download locations are suggested below.

It is essential that you [verify the integrity](#) of the downloaded file using the PGP signature (`.asc` file) or a hash (`.md5` or `.sha*` file).

HTTP

<https://dlcdn.apache.org/zookeeper/zookeeper-3.8.3/apache-zookeeper-3.8.3-bin.tar.gz>



apache-zookeeper-3.8.3-bin.tar.gz

Completada — 14,2 MB



[Mostrar todas las descargas](#)

2. INSTALACION ZOOKEEPER

Paso 3. Descomprimos el tar.gz, renombramos la carpeta y la movemos a /opt/hadoop

```
hadoop@nodo1:~/Downloads$ ls
access_log                empleados.txt              motivo_pais.csv
apache-hive-3.1.3-bin.tar.gz  hadoop-3.2.4.tar.gz      'MyJob$MapClass.class'
apache-zookeeper-3.8.3-bin.tar.gz  hadoop-3.3.6.tar.gz    'MyJob$Reduce.class'
cite75_99.txt              hue-4.10.0                MyJob.class
ContarPalabras.java        'hue-4.10.0(1).tgz'      MyJob.jar
countries.csv              hue-4.10.0.tgz            MyJob.java
deslizamientos.csv         hue-release-4.10.0.zip   tabla_deslizamientos.txt
hadoop@nodo1:~/Downloads$ tar xvf apache-zookeeper-3.8.3-bin.tar.gz
```

```
apache-zookeeper-3.8.3-bin/lib/jline-2.14.6.jar
apache-zookeeper-3.8.3-bin/lib/metrics-core-4.1.12.1.jar
apache-zookeeper-3.8.3-bin/lib/snappy-java-1.1.10.5.jar
hadoop@nodo1:~/Downloads$ mv apache-zookeeper-3.8.3-bin zookeeper
hadoop@nodo1:~/Downloads$ mv zookeeper/ /opt/hadoop
hadoop@nodo1:~/Downloads$ mv /opt/hadoop/zookeeper/ /opt/hadoop/zoo
hadoop@nodo1:~/Downloads$ ls /opt/hadoop/zoo
bin  conf  docs  lib  LICENSE.txt  NOTICE.txt  README.md  README_packaging.md
hadoop@nodo1:~/Downloads$
```

Si entramos vemos el directorio bin y conf que utilizaremos para la configuración

2. INSTALACION ZOOKEEPER

Paso 4. Esta parte que hemos hecho en el nodo1 tenemos que repetirla en todos aquellos nodos donde vamos a utilizar Zookeeper. Como estamos trabajando con tres nodos, instalaremos en los 3 zookeeper, ya que han de haber al menos tres nodos Zookeeper. Lo instalaremos en nodo1, nodo2 y nodo3. Lo copiaremos del nodo1 a los otros dos nodos.

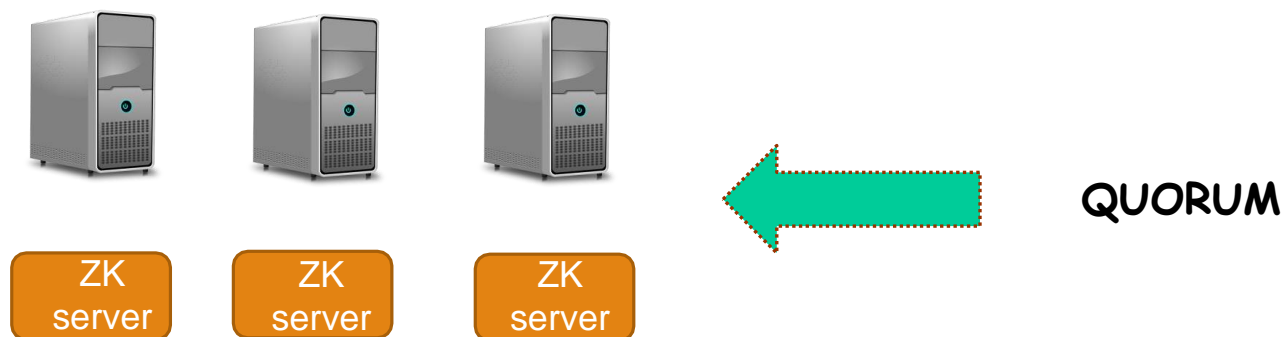
```
hadoop@nodo1:/opt/hadoop$ scp -r zoo nodo2:/opt/hadoop
```

```
hadoop@nodo1:/opt/hadoop$ scp -r zoo nodo3:/opt/hadoop
```

De esta manera tenemos las tres máquinas exactamente igual.

3. CONFIGURACION ZOOKEEPER

- ❖ Vamos a ver la teoría de cómo se configura zookeeper como servidor en clúster. Luego lo aplicaremos a Hadoop con algún pequeño cambio.
- ❖ Esto valido para Hadoop y para cualquier otro producto con el cual utilicemos Zookeeper
- ❖ Configuraremos 3 servidores zookeeper en nodo1, 2 y 3.
- ❖ Estos 3 servidores zookeeper se denominan quorum (conjunto de servidores replicados).



3. CONFIGURACION ZOOKEEPER

- ❖ Estos tres servidores van a mantener la configuración de Hadoop, y van a tener que ser gestionados o manejados de la misma manera.
- ❖ Los tres servidores van a tener asociados un fichero de configuración que llamaremos **zoo.cfg**
- ❖ En cada servidor se creará un directorio donde se van a guardar snapshots, logs, y cosas que necesita zookeeper. Aunque trabaja en memoria necesita dejar cosas en algún sitio.
- ❖ Utilizaremos el mismo directorio **/datos/zoo** que estamos utilizando para los namenode y los datanode.

3. CONFIGURACION ZOOKEEPER

Puertos de comunicación

- ❖ Estas máquinas se escuchan entre sí por 2 puertos, uno para configuración y el otro para gestión:
 - El 2888 se suele utilizar para las conexiones y comunicaciones entre ellos: Por ejemplo si se modifica un fichero en un servidor, permite sincronizar ese fichero con el resto de servidores automáticamente.
 - El 3888 sirve para determinar el líder
- ❖ Todo esto se configura en el fichero zoo.cfg
- ❖ Los clientes de Zookeeper se conectan a través del puerto 2181 (es el por defecto que conviene dejar)

3. CONFIGURACION ZOOKEEPER

Paso 1. Vamos a configurar cada uno de nuestros servidores. La configuración tiene que ser igual en todos. Vamos al directorio **/opt/hadoop/zoo**, donde hicimos la instalación de los binarios y visualizamos el directorio bin.

```
hadoop@nodo1:/opt/hadoop/zoo/bin$ ls
README.txt      zkCli.sh        zkServer.cmd    zkSnapshotComparer.cmd  zkSnapshotToolkit.sh
zkCleanup.sh    zkEnv.cmd       zkServer-initialize.sh  zkSnapshotComparer.sh    zkTxnLogToolkit.cmd
zkCli.cmd       zkEnv.sh        zkServer.sh       zkSnapshotToolkit.cmd    zkTxnLogToolkit.sh
hadoop@nodo1:/opt/hadoop/zoo/bin$
```

Tenemos algunos comandos que utilizaremos zkserver y el zkcli. Están tanto para Windows como para Linux aunque nosotros utilizaremos los de Linux.

3. CONFIGURACION ZOOKEEPER

Paso 2. Volvemos atrás y nos metemos en el directorio conf, donde creamos el fichero zoo.cfg

```
hadoop@nodo1:/opt/hadoop/zoo/conf$ ls  
configuration.xml  logback.xml  zoo  sample.cfg  
hadoop@nodo1:/opt/hadoop/zoo/conf$
```

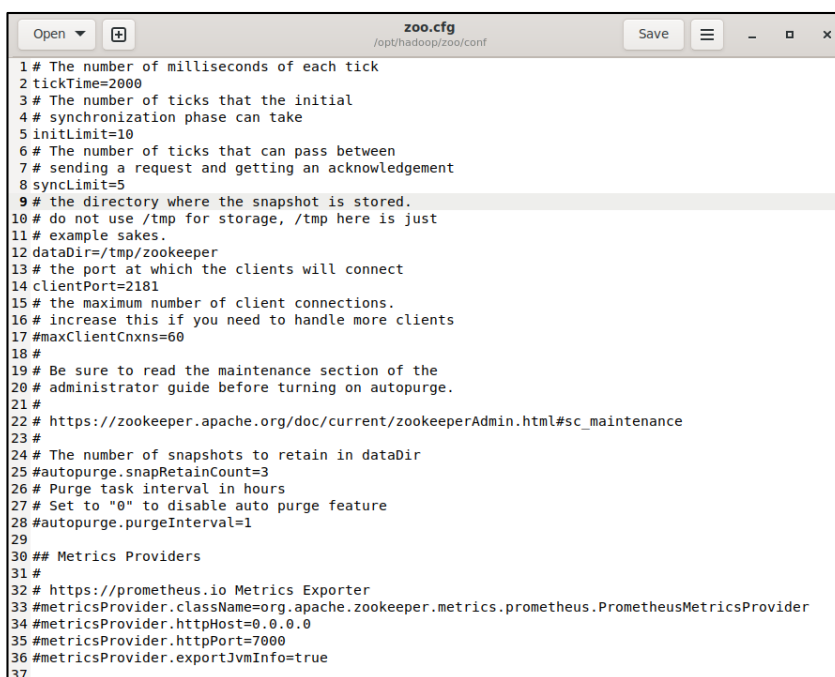
Copiamos el fichero zoo_sample.cfg como zoo.cfg.

```
hadoop@nodo1:/opt/hadoop/zoo/conf$ ls  
configuration.xml  logback.xml  zoo_sample.cfg  
hadoop@nodo1:/opt/hadoop/zoo/conf$ cp zoo_sample.cfg zoo.cfg  
hadoop@nodo1:/opt/hadoop/zoo/conf$ ls  
configuration.xml  logback.xml  zoo.cfg  zoo_sample.cfg  
hadoop@nodo1:/opt/hadoop/zoo/conf$
```

NOTA: No es necesario llamarlo así pero si no, cada vez que arrancamos el servidor tenemos que indicar el nombre

3. CONFIGURACION ZOOKEEPER

Paso 3. zoo.cfg tiene una serie de parámetros que podemos utilizar para configurar nuestro zookeeper

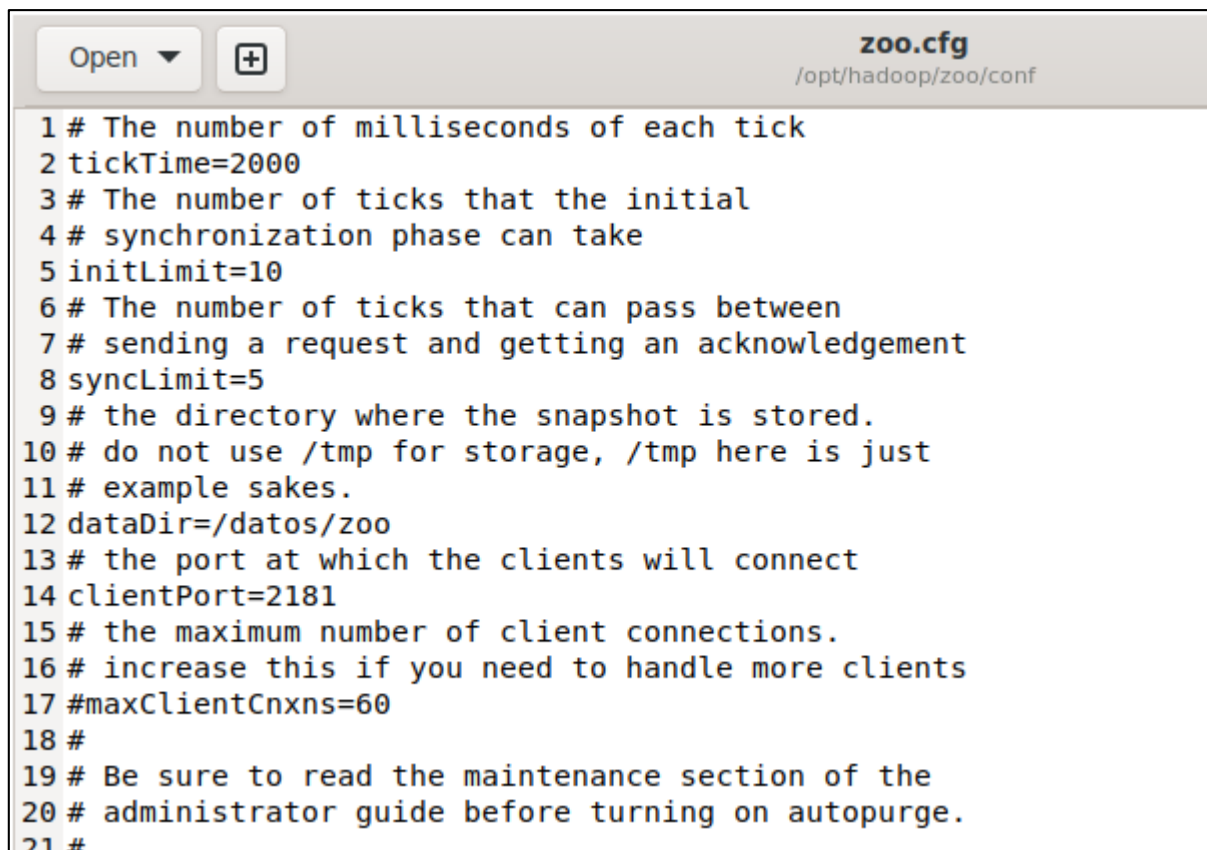


```
1 # The number of milliseconds of each tick
2 tickTime=2000
3 # The number of ticks that the initial
4 # synchronization phase can take
5 initLimit=10
6 # The number of ticks that can pass between
7 # sending a request and getting an acknowledgement
8 syncLimit=5
9 # the directory where the snapshot is stored.
10 # do not use /tmp for storage, /tmp here is just
11 # example sake.
12 dataDir=/tmp/zookeeper
13 # the port at which the clients will connect
14 clientPort=2181
15 # the maximum number of client connections.
16 # increase this if you need to handle more clients
17 #maxClientCnxns=60
18 #
19 # Be sure to read the maintenance section of the
20 # administrator guide before turning on autopurge.
21 #
22 # https://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc\_maintenance
23 #
24 # The number of snapshots to retain in dataDir
25 #autopurge.snapRetainCount=3
26 # Purge task interval in hours
27 # Set to "0" to disable auto purge feature
28 #autopurge.purgeInterval=1
29
30 ## Metrics Providers
31 #
32 # https://prometheus.io Metrics Exporter
33 #metricsProvider.className=org.apache.zookeeper.metrics.prometheus.PrometheusMetricsProvider
34 #metricsProvider.httpHost=0.0.0.0
35 #metricsProvider.httpPort=7000
36 #metricsProvider.exportJvmInfo=true
37
```

NOTA: Este fichero de configuración tiene que ser igual en el resto de los nodos zookeeper para mantener una sincronización de la infraestructura

3. CONFIGURACION ZOOKEEPER

Paso 3. Configuramos los siguientes parámetros: ticktime, initLimit, syncLimit, dataDir, clientPort



```
zoo.cfg
/opt/hadoop/zoo/conf

1 # The number of milliseconds of each tick
2 tickTime=2000
3 # The number of ticks that the initial
4 # synchronization phase can take
5 initLimit=10
6 # The number of ticks that can pass between
7 # sending a request and getting an acknowledgement
8 syncLimit=5
9 # the directory where the snapshot is stored.
10 # do not use /tmp for storage, /tmp here is just
11 # example sakes.
12 dataDir=/datos/zoo
13 # the port at which the clients will connect
14 clientPort=2181
15 # the maximum number of client connections.
16 # increase this if you need to handle more clients
17 #maxClientCnxns=60
18 #
19 # Be sure to read the maintenance section of the
20 # administrator guide before turning on autopurge.
21 #
```

3. CONFIGURACION ZOOKEEPER

Parámetros

- ❑ tickTime: Número de milisegundos de cada tick, que los servidores zookeeper se van haciendo entre ellos.
- ❑ initLimit: Número de ticks que la fase inicial de sincronización puede tomar.
- ❑ syncLimit: Número de ticks que puede pasar entre enviar una petición y recibir reconocimiento.
- ❑ dataDir: Directorio donde los servidores zookeeper van a dejar información de almacenamiento, logging, snapshot y cosas suyas internas. Lo vamos a crear y configurar en el directorio /datos/zoo
- ❑ clientPort: Puerto cliente 2181

3. CONFIGURACION ZOOKEEPER

Paso 4. En la parte final del documento, identificamos con esta nomenclatura los servidores zookeeper que vamos a tener: **server.x=nodox:2888:3888**

```
32 # https://prometheus.io Metrics Exporter
33 #metricsProvider.className=org.apache.zook
34 #metricsProvider.httpHost=0.0.0.0
35 #metricsProvider.httpPort=7000
36 #metricsProvider.exportJvmInfo=true
37 server.1=nodo1:2888:3888
38 server.2=nodo2:2888:3888
39 server.3=nodo3:2888:3888
```

NOTA: 2888 es el puerto para hablar entre ellos y 3888 para decidir el líder

3. CONFIGURACION ZOOKEEPER

Paso 4. Este fichero de configuración debe de ser exacto en cada uno de los servidores. Vamos a copiarlo al nodo2 y al nodo3

```
hadoop@nodo1:/opt/hadoop/zoo/conf$ scp zoo.cfg nodo2:/opt/hadoop/zoo//conf
zoo.cfg                                100% 1254      2.2MB/s   00:00
hadoop@nodo1:/opt/hadoop/zoo/conf$ scp zoo.cfg nodo3:/opt/hadoop/zoo//conf
zoo.cfg                                100% 1254      3.0MB/s   00:00
hadoop@nodo1:/opt/hadoop/zoo/conf$
```

Paso 5. Creamos en los 3 nodos el directorio /datos/zoo, con el que vamos a trabajar. Lo hacemos usando el comando ssh

```
hadoop@nodo1:~$ mkdir /datos/zoo
hadoop@nodo1:~$ ssh nodo2 mkdir /datos/zoo
hadoop@nodo1:~$ ssh nodo3 mkdir /datos/zoo
hadoop@nodo1:~$ █
```

4. ARRANCAR ZOOKEEPER

Paso 1. Antes de arrancar zookeeper, tenemos que habilitar el entorno en modo clúster para poder empezar a trabajar. Vamos al directorio /datos/zoo y tenemos que crear un fichero llamado myid, con un 1 dentro

```
hadoop@nodo1:~$ cd /datos/zoo/  
hadoop@nodo1:/datos/zoo$ echo 1 > myid  
hadoop@nodo1:/datos/zoo$ cat myid  
1  
hadoop@nodo1:/datos/zoo$ █
```

NOTA: Zookeeper identifica los servidores mediante este fichero. En cada uno de los servidores tendremos que poner un fichero llamado myid con el número de servidor. En el Nodo 2 myid contendrá un 2 y en el nodo 3 un 3. Si tuviéramos los nodos 1, 7 y 24, los id de los servidores serian 1, 2 y 3. Se hace por el número de servidor zookeeper no por el número que ocupe dentro del cluster₂₃



4. ARRANCAR ZOOKEEPER

Paso 2. Creamos los ficheros myid en los nodos 2 y 3 usando ssh

```
hadoop@nodo2:/datos/zoo$ echo 2 > myid
hadoop@nodo2:/datos/zoo$ ls
myid
hadoop@nodo2:/datos/zoo$ cat myid
2
hadoop@nodo2:/datos/zoo$ █
```

```
hadoop@nodo3:~$ cd /datos/zoo
hadoop@nodo3:/datos/zoo$ echo 3 > myid
hadoop@nodo3:/datos/zoo$ ls
myid
hadoop@nodo3:/datos/zoo$ cat myid
3
hadoop@nodo3:/datos/zoo$
```

Paso 3. Actualizamos el fichero de configuración del usuario .bashrc con el path de zookeeper, creando previamente la variable ZOOKEEPER_HOME

sudo nano .bashrc

```
export HADOOP_HOME=/opt/hadoop
export HIVE_HOME=/opt/hadoop/hive
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export ZOOKEEPER_HOME=/opt/hadoop/zoo
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$HIVE_HOME/bin:$ZOOKEEPER_HOME/bin
```


4. ARRANCAR ZOOKEEPER

Paso 4. Copiamos el fichero `.bashrc` en los nodos 2 y 3 para que este exactamente igual

```
hadoop@nodo1:~$ sudo nano .bashrc
hadoop@nodo1:~$ scp .bashrc nodo2:/home/hadoop
.bashrc
hadoop@nodo1:~$ scp .bashrc nodo3:/home/hadoop
.bashrc
```

Paso 5. Actualizamos las variables de entorno de `.bashrc`

```
hadoop@nodo1:~$ . .bashrc
hadoop@nodo1:~$ source .bashrc
hadoop@nodo1:~$ ssh nodo2 source /home/hadoop/.bashrc
hadoop@nodo1:~$ ssh nodo3 source /home/hadoop/.bashrc
hadoop@nodo1:~$ █
```

4. ARRANCAR ZOOKEEPER

Paso 6. El comando `zkServer.sh` (en `/opt/hadoop/zoo/bin`) permite ejecutar distintas operaciones sobre el servidor zookeeper. Si lo ejecutamos vemos las subopciones del comando: `start`, `stop`, `restart`, `status`, ect

```
hadoop@nodo1:~$ zkServer.sh
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/../conf/zoo.cfg
Usage: /opt/hadoop/zoo/bin/zkServer.sh [--config <conf-dir>] {start|start-foreground
|stop|version|restart|status|print-cmd}
hadoop@nodo1:~$
```

Paso 7. Ponemos la subopción `start` y lo ejecutamos.

```
hadoop@nodo1:~$ zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
hadoop@nodo1:~$
```

Indica que ha arrancado, pero da muy poca información. Localiza perfectamente el fichero `zoo.cfg`, sino tendríamos que indicarle que arrancara con otro fichero configuración.

4. ARRANCAR ZOOKEEPER

Paso 8. Ejecutamos zkServer.sh status:

```
hadoop@nodo1:~$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/./conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Error contacting service. It is probably not running.
1 hadoop@nodo1:~$
```

Indica que probablemente no se está ejecutando. Esto no es muy fiable: si no tenemos más de 2 servidores en el quorum de zookeeper, considera que el clúster no está funcionando → **Solución:** Se tiene que ejecutar el servicio zookeeper en cada servidor o nodo

Paso 9. Hacemos `ps -ef | grep zk`. Vemos que bajo este nombre no hay nada funcionando

```
1 hadoop@nodo1:~$ ps -ef | grep zk
hadoop      58574      1980  0 15:20 pts/2    00:00:00 grep --color=auto zk
hadoop@nodo1:~$
```

4. ARRANCAR ZOOKEEPER

Paso 10. En realidad el proceso se llama Quorum. Vemos el fichero QuorumPeerMain que en realidad es zookeeper. Indica que está funcionando y en cambio el status dice "No estoy funcionando" → Esto es porque todavía no hemos arrancado al menos dos nodos. En un cluster zookeeper tiene que haber siempre mayoría de nodos o quorum. Si hemos configurado en el fichero zoo.cfg 3 nodos, debemos tener al menos dos funcionando; como sólo tenemos uno, el clúster no está activo

```
hadoop@nodo1:~/datos/zoo$ ps -ef | grep Qu
root      516      2  0 20:57 ?        00:00:00 [iprt-VBoxWQueue]
hadoop    13305   1383  0 21:11 pts/2    00:00:00 /usr/lib/jvm/java-8-openjdk-amd64/bin/java
OnOutOfMemoryError -XX:OnOutOfMemoryError=kill -9 %p -cp /opt/hadoop/zoo/bin/./zookeeper-serv
hadoop/zoo/bin/./build/lib/*.jar:/opt/hadoop/zoo/bin/./lib/zookeeper-prometheus-metrics-3.8.
zoo/bin/./lib/snappy-java-1.1.10.5.jar:/opt/hadoop/zoo/bin/./lib/slf4j-api-1.7.30.jar:/opt/h
adoop/zoo/bin/./lib/simpleclient-common-0.9.0.jar:/opt/hadoop/zoo/bin/./lib/simpleclient-0.9.
-transport-native-epoll-4.1.94.Final.jar:/opt/hadoop/zoo/bin/./lib/netty-transport-classes-ep
olver-4.1.94.Final.jar:/opt/hadoop/zoo/bin/./lib/netty-handler-4.1.94.Final.jar:/opt/hadoop/z
in/./lib/netty-buffer-4.1.94.Final.jar:/opt/hadoop/zoo/bin/./lib/metrics-core-4.1.12.1.jar:/
oo/bin/./lib/jline-2.14.6.jar:/opt/hadoop/zoo/bin/./lib/jetty-util-ajax-9.4.52.v20230823.jar
.jar:/opt/hadoop/zoo/bin/./lib/jetty-server-9.4.52.v20230823.jar:/opt/hadoop/zoo/bin/./lib/j
b/jetty-http-9.4.52.v20230823.jar:/opt/hadoop/zoo/bin/./lib/javax.servlet-api-3.1.0.jar:/opt
bin/./lib/jackson-annotations-2.15.2.jar:/opt/hadoop/zoo/bin/./lib/commons-io-2.11.0.jar:/op
zoo/bin/./zookeeper-*.jar:/opt/hadoop/zoo/bin/./zookeeper-server/src/main/resources/lib/*.ja
se org.apache.zookeeper.server.quorum.QuorumPeerMain /opt/hadoop/zoo/bin/./conf/zoo.cfg
hadoop    19009   1952  0 21:17 pts/2    00:00:00 grep --color=auto Qu
hadoop@nodo1:~/datos/zoo$
```

4. ARRANCAR ZOOKEEPER

Paso 11. Si vamos a /opt/hadoop/zoo/logs. Encontramos el fichero zookeeper-hadoop-server-nodo1-out

```
hadoop@nodo1:/opt/hadoop/zoo/logs$ ls
zookeeper-hadoop-server-nodo1.out
hadoop@nodo1:/opt/hadoop/zoo/logs$
```

Si observamos su contenido vemos que hace referencia a que no esta consiguiendo la elección de un líder.

```
2023-11-15 21:36:16,596 [myid:] - WARN [QuorumConnectionThread-[myid=1]-48:o.a.z.s.q.QuorumCnxManager@401] - Cannot open
channel to 2 at election address nodo2/192.168.0.102:3888
java.net.ConnectException: Connection refused (Connection refused)
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:607)
    at org.apache.zookeeper.server.quorum.QuorumCnxManager.initiateConnection(QuorumCnxManager.java:384)
    at org.apache.zookeeper.server.quorum.QuorumCnxManager$QuorumConnectionReqThread.run(QuorumCnxManager.java:458)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:750)
hadoop@nodo1:/opt/hadoop/zoo/logs$ ls
zookeeper-hadoop-server-nodo1.out
```

4. ARRANCAR ZOOKEEPER

Paso 12. Vamos al nodo2 y comprobamos que el path esta correcto allí. Ejecutamos zkServer.sh start y comprobamos su estado

```
hadoop@nodo2:~$ echo $ZOOKEEPER_HOME
/opt/hadoop/zoo
hadoop@nodo2:~$ zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
hadoop@nodo2:~$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
hadoop@nodo2:~$
```

Vemos que esta en modo líder. Ya no se queja porque hay dos nodos funcionando. Ahora sí que pueden formar un clúster al ser tres nodos el numero mínimo.

4. ARRANCAR ZOOKEEPER

Paso 13. Vamos al nodo1 y observamos el status de zookeeper

```
hadoop@nodo2:~$ exit
logout
Connection to nodo2 closed.
hadoop@nodo1:/opt/hadoop/zoo/logs$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
hadoop@nodo1:/opt/hadoop/zoo/logs$
```

Ahora indica que está funcionando y que esta en modo follower.

NOTA: Normalmente aunque no siempre es seguro al 100% cuando arranca el segundo nodo se suele convertir en el líder y el nodo 1, el que inició primero, se suele poner como follower. Esto puede cambiar perfectamente.

4. ARRANCAR ZOOKEEPER

Paso 14. Vamos al nodo3 y arrancamos el zkServer.sh

```
hadoop@nodo3:~$ zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
hadoop@nodo3:~$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/./conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
hadoop@nodo3:~$ █
```

Si hacemos un status nos indica que también es un follower. Ahora tenemos tres nodos zookeeper:

nodo1 → follower

nodo2 → lider

nodo3 → follower

4. ARRANCAR ZOOKEEPER

Paso 15. Si vamos a /datos/zoo vemos que ya va dejando aquí sus cosas. Es totalmente interno y no debemos de tocar.

```
hadoop@nodo1:/opt/hadoop/zoo/logs$ cd /datos/zoo/  
hadoop@nodo1:/datos/zoo$ ls -al  
total 20  
drwxrwxr-x 3 hadoop hadoop 4096 nov 15 21:11 .  
drwxr-xr-x 4 hadoop hadoop 4096 nov 14 21:45 ..  
-rw-rw-r-- 1 hadoop hadoop 2 nov 15 21:10 myid  
drwxrwxr-x 2 hadoop hadoop 4096 nov 15 21:58 version-2  
-rw-rw-r-- 1 hadoop hadoop 5 nov 15 21:11 zookeeper_server.pid  
hadoop@nodo1:/datos/zoo$
```

```
hadoop@nodo3:~$ cd /datos/zoo  
hadoop@nodo3:/datos/zoo$ ls  
myid version-2 zookeeper_server.pid  
hadoop@nodo3:/datos/zoo$ ls -al  
total 20  
drwxrwxr-x 3 hadoop hadoop 4096 nov 15 22:16 .  
drwxr-xr-x 4 hadoop hadoop 4096 nov 14 21:46 ..  
-rw-rw-r-- 1 hadoop hadoop 2 nov 15 16:12 myid  
drwxrwxr-x 2 hadoop hadoop 4096 nov 15 22:16 version-2  
-rw-rw-r-- 1 hadoop hadoop 4 nov 15 22:16 zookeeper_server.pid  
hadoop@nodo3:/datos/zoo$
```

Paso 16. Hacemos JPS en cada uno de los servidores veréis que tenemos un proceso QuorumPeerMain que es cómo lo denomina zookeeper

```
hadoop@nodo3:/datos/zoo$ jps  
2578 Jps  
2150 NodeManager  
1995 DataNode  
2479 QuorumPeerMain  
hadoop@nodo3:/datos/zoo$
```

```
hadoop@nodo1:/opt/hadoop/zoo/logs$ jps  
85744 Jps  
3234 NameNode  
13305 QuorumPeerMain  
3564 SecondaryNameNode  
3932 ResourceManager  
hadoop@nodo1:/opt/hadoop/zoo/logs$
```

5. CLIENTE ZOOKEEPER

Paso 1. El cliente zkCli.sh nos permite conectarnos a nuestros servidores zookeeper en remoto para poder hacer algún tipo de operación.

Recordar que debemos tener arrancados el servidor zkServer en cada nodo/servidor

NOTA: El servidor zookeeper vale para muchos productos, y se tiene que adaptar a cada uno de ellos. Estos productos guardan determinados datos en la estructura jerárquica de directorios de zookeeper. Usaremos Zookeeper para tener alta disponibilidad en Hadoop con HDFS y con Yarn. Estos componentes rellenan información en el servidor zookeeper para poder hacer esa tarea.

5. CLIENTE ZOOKEEPER

Paso 2. Ejecutamos el cliente zookeeper con:

zkCli.sh (a secas) o zkCli.sh -server nodo1:2181

```
2023-11-16 11:54:01,034 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1011] - Socket connection established, initiating session, client: /127.0.0.1:37232, server: localhost/127.0.0.1:2181
2023-11-16 11:54:01,079 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1452] - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x10000605e0b0000, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTING) 0]
```

5. CLIENTE ZOOKEEPER

Paso 2. Una vez dentro podemos poner diferentes comandos, como help:

```
[zk: nodo1:2181(CONNECTED) 4] help
ZooKeeper -server host:port -client-configuration properties-file cmd args
  addWatch [-m mode] path # optional mode is one of [PERSISTENT, PERS
  addauth scheme auth
  close
  config [-c] [-w] [-s]
  connect host:port
  create [-s] [-e] [-c] [-t ttl] path [data] [acl]
  delete [-v version] path
  deleteall path [-b batch size]
  delquota [-n|-b|-N|-B] path
  get [-s] [-w] path
  getAcl [-s] path
  getAllChildrenNumber path
  getEphemerals path
  history
  listquota path
  ls [-s] [-w] [-R] path
  printwatches on|off
  quit
  reconfig [-s] [-v version] [[-file path] | [-members serverID=host:
  redo cmdno
  removewatches path [-c|-d|-a] [-l]
  set [-s] [-v version] path data
  setAcl [-s] [-v version] [-R] path acl
  setquota -n|-b|-N|-B val path
  stat [-w] path
  sync path
  version
  whoami
Command not found: Command not found help
[zk: nodo1:2181(CONNECTED) 5]
```

5. CLIENTE ZOOKEEPER

Paso 3. En zookeeper los znodes son unos registros conformados dentro de una estructura jerárquica similar a la de directorios, donde podemos ir configurando información para trabajar.

Por ejemplo para tener alta disponibilidad en Hadoop (HDFS, Yarn), se creará una estructura de directorios para hacer eso

Si hacemos ls / vemos que no tiene nada

```
[zk: localhost:2181(CONNECTED) 1] ls /  
[zookeeper]  
[zk: localhost:2181(CONNECTED) 2]
```

5. CLIENTE ZOOKEEPER

Paso 4. Aquí creamos un znode o directorio con **create /m1 v1** → Indicamos el nodo que quiero crear m1 y el contenido v1 que quiero poner en ese nodo

```
[zk: localhost:2181(CONNECTED) 1] ls /  
[zookeeper]  
[zk: localhost:2181(CONNECTED) 2] create /m1 v1  
Created /m1  
[zk: localhost:2181(CONNECTED) 3] ls /  
[m1, zookeeper]  
[zk: localhost:2181(CONNECTED) 4]
```

Al hacer un ls vemos que ha creado m1 y un genérico llamado zookeeper

Paso 5. Si recuperamos la información de m1 (con get) nos indica que tiene una variable llamada v1

```
[zk: localhost:2181(CONNECTED) 4] get /m1  
v1  
[zk: localhost:2181(CONNECTED) 5]
```

5. CLIENTE ZOOKEEPER

Paso 6. Creamos znodo m2 y le indicamos v2 como contenido. Si hacemos ls nos salen los dos.

```
[zk: localhost:2181(CONNECTED) 3] create /m2 v2
Created /m2
[zk: localhost:2181(CONNECTED) 4] ls /
[m1, m2, zookeeper]
[zk: localhost:2181(CONNECTED) 5]
```

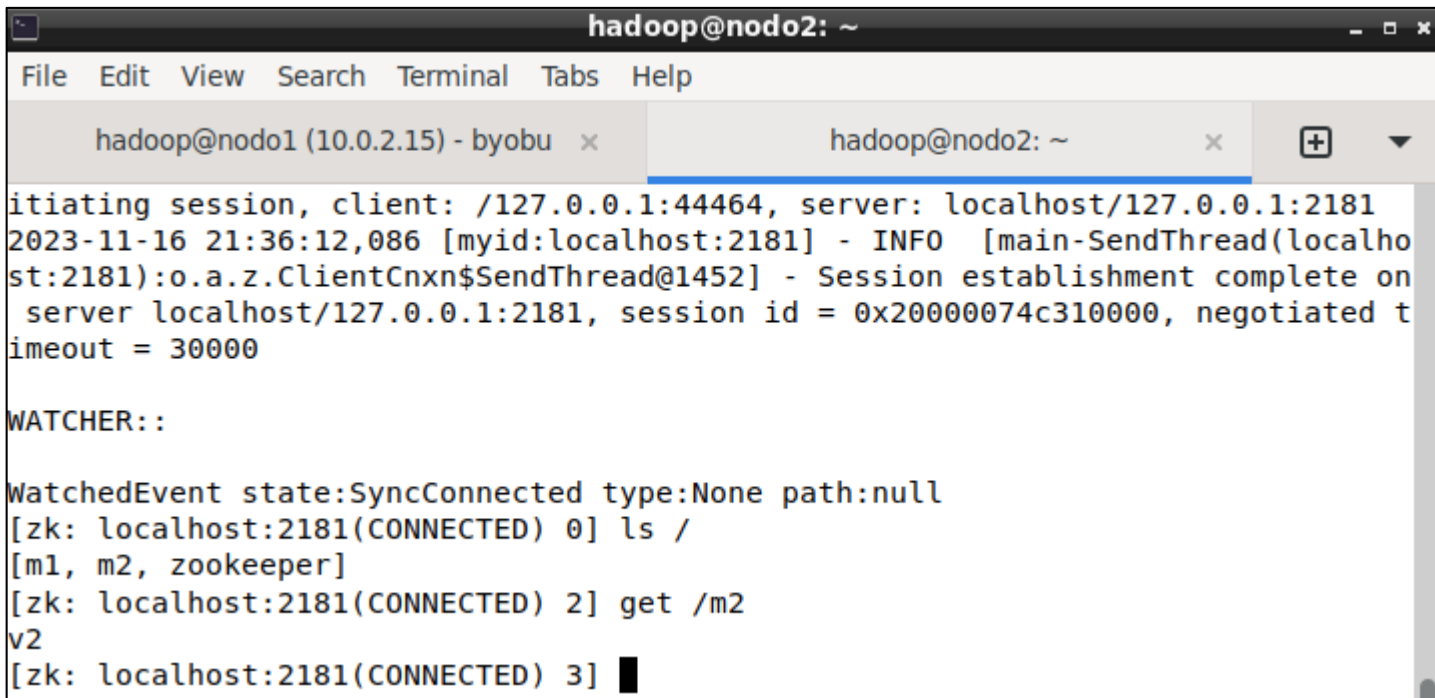
Paso 7. Si hacemos un get de m2 nos debería decir que tiene aquí una cosa llamada v2

```
[zk: localhost:2181(CONNECTED) 5] get /m2
v2
[zk: localhost:2181(CONNECTED) 6]
```

NOTA: Volveremos a ver esto cuando estemos trabajando con Yarn y HDFS en alta disponibilidad.

5. CLIENTE ZOOKEEPER

Paso 8. Vamos al nodo2 y nos conectamos con zkCli.sh al servidor 2, sin poner puerto ya que no hemos cambiado el puerto por defecto. Hacemos un ls y un get de m2 y nos da la misma información que hemos creado en el nodo1.



```
hadoop@nodo2: ~  
File Edit View Search Terminal Tabs Help  
hadoop@nodo1 (10.0.2.15) - byobu x hadoop@nodo2: ~ x + ▾  
Initiating session, client: /127.0.0.1:44464, server: localhost/127.0.0.1:2181  
2023-11-16 21:36:12,086 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1452] - Session establishment complete on  
server localhost/127.0.0.1:2181, session id = 0x20000074c310000, negotiated t  
imeout = 30000  
  
WATCHER::  
  
WatchedEvent state:SyncConnected type:None path:null  
[zk: localhost:2181(CONNECTED) 0] ls /  
[m1, m2, zookeeper]  
[zk: localhost:2181(CONNECTED) 2] get /m2  
v2  
[zk: localhost:2181(CONNECTED) 3] █
```


5. CLIENTE ZOOKEEPER

NOTA: Zookeeper sincroniza automáticamente los servidores. Cuando Yarn vaya a dejar cualquier cosa aquí la configuración será replicada automáticamente en el resto de los nodos zookeeper.

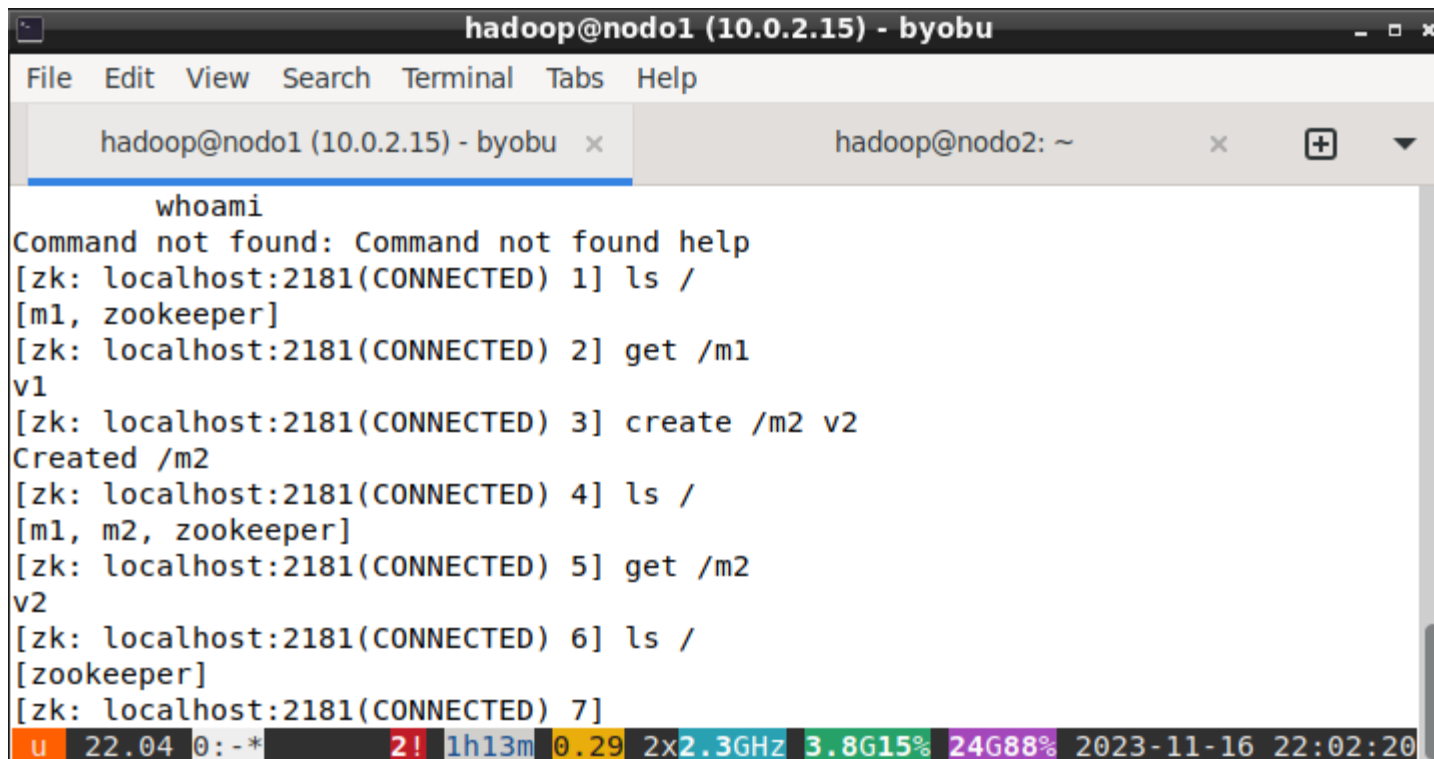
En alta disponibilidad, si uno de los nodos se cae de los maestros, otro que este pasivo, pasa a formar parte del trabajo automáticamente.

Paso 9. Desde nodo2 hacemos un delete de la información que tenemos en m1 y m2. Al hacer un ls está vacío

```
[zk: localhost:2181(CONNECTED) 3] delete /m1
[zk: localhost:2181(CONNECTED) 4] delete /m2
[zk: localhost:2181(CONNECTED) 5] ls /
[zookeeper]
[zk: localhost:2181(CONNECTED) 6] □
```

5. CLIENTE ZOOKEEPER

Paso 10. Si vamos a nodo1, cuando se sincronice, vemos que tampoco tenemos nada en el nodo 1



```
hadoop@nodo1 (10.0.2.15) - byobu
File Edit View Search Terminal Tabs Help
hadoop@nodo1 (10.0.2.15) - byobu x hadoop@nodo2: ~ x + v
whoami
Command not found: Command not found help
[zk: localhost:2181(CONNECTED) 1] ls /
[m1, zookeeper]
[zk: localhost:2181(CONNECTED) 2] get /m1
v1
[zk: localhost:2181(CONNECTED) 3] create /m2 v2
Created /m2
[zk: localhost:2181(CONNECTED) 4] ls /
[m1, m2, zookeeper]
[zk: localhost:2181(CONNECTED) 5] get /m2
v2
[zk: localhost:2181(CONNECTED) 6] ls /
[zookeeper]
[zk: localhost:2181(CONNECTED) 7]
u 22.04 0: -* 2! 1h13m 0.29 2x2.3GHz 3.8G15% 24G88% 2023-11-16 22:02:20
```

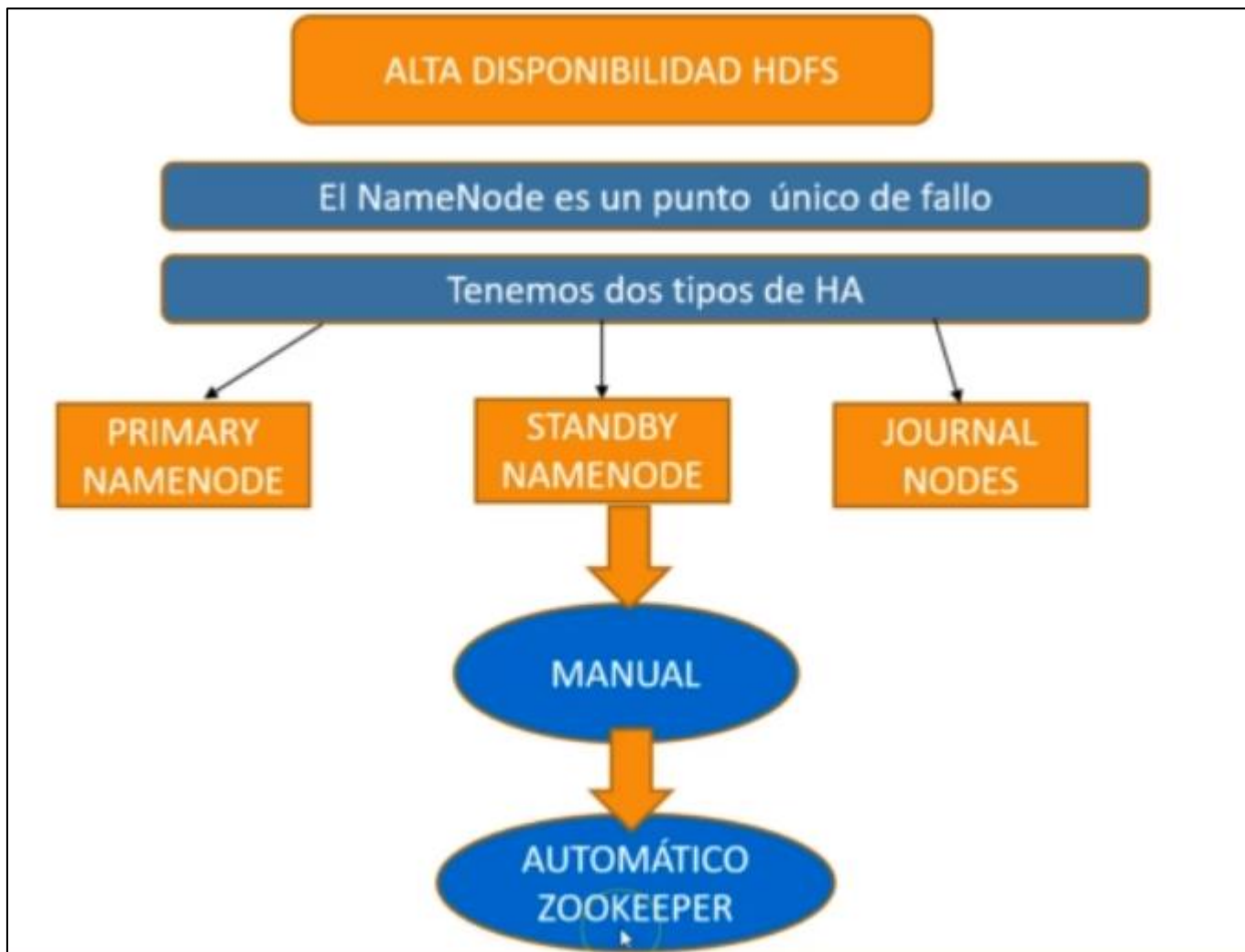
Zookeeper permite mantener información en directorios de manera sincronizada y trabajar con ella

6. ALTA DIPONIBILIDAD HDFS

- ❖ Una vez que hemos visto como se pueden configurar los servidores zookeeper, veremos cómo podemos utilizarlo para poder configurar la alta disponibilidad en HDFS
- ❖ Desde las primeras versiones de Hadoop la alta disponibilidad de HDFS, el namenode era un único punto de fallo, es decir que si se nos cae el namenode se nos cae todo el Cluster, ya que no podríamos acceder a los datos
- ❖ En la versión 2 se configuraron la posibilidad de tener alta disponibilidad en HDFS.
- ❖ Tenemos dos tipos de alta disponibilidad: Podemos tener al HDFS basada sobre todo en una actuación manual o basada en una actuación automática.

6. ALTA DIPONIBILIDAD HDFS

❖ Esquema alta disponibilidad HDFS



6. ALTA DIPONIBILIDAD HDFS

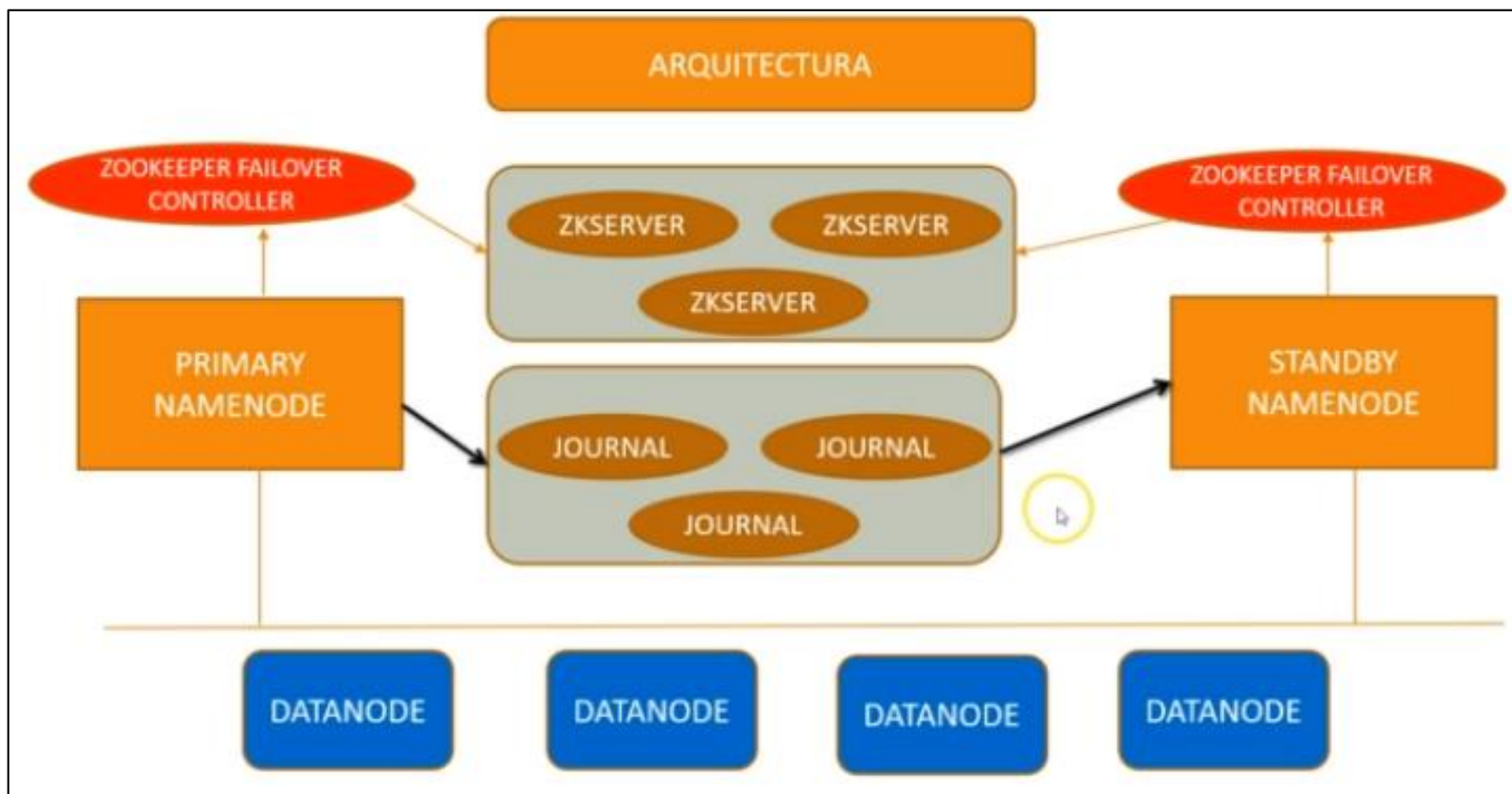
- ❖ En la alta disponibilidad HDFS vamos a tener 3 componentes que nos permiten trabajar con esta filosofía:
 - ❖ Un Primary Namenode (nodo primario)
 - ❖ Un Standby Namenode (nodo standby o secundario)
 - ❖ Journal nodes (nuevos componentes)
- ❖ El Primary Node escribe en los Journal Nodes y estos cambios que se van escribiendo en los nodos de tipo Journal son leídos por el standby.
- ❖ En un journal node se van copiando los ficheros edit de HDFS, donde se va guardando todas las transacciones, todos los log de HDFS

6. ALTA DIPONIBILIDAD HDFS

- ❖ Con esto ya tendríamos la posibilidad de tener un failover manual: Si se cae el nodo primario, de manera manual nos podemos pasar al nodo secundario, es decir podemos convertir el secundario en primario
- ❖ En entornos productivos esto es un problema porque quien va a estar pendiente de si se cae o no se cae, que luego lo levante a mano, etc
- ❖ Se puede utilizar un failover automático, de manera que sea el propio clúster el que cambia entre el nodo primario y secundario utilizando zookeeper
- ❖ Esto es donde entra la parte de zookeeper dentro de la alta disponibilidad de HDFS.

6. ALTA DIPONIBILIDAD HDFS

❖ Arquitectura alta disponibilidad con Zookeeper



6. ALTA DIPONIBILIDAD HDFS

- ❖ La arquitectura de alta disponibilidad es sencilla
- ❖ Tenemos un nodo primario y un nodo standby. No se llama secundario porque existe el secondary namenode utilizado como complemento al checkpoint.
- ❖ El standby es un namenode completo pero sin trabajar.
- ❖ Tanto el primary como el standby reciben peticiones por parte de los datanodes que tengamos en el clúster
- ❖ El Primary namenode escribe, en un número máximo de ficheros de Journal, los cambios aparte de su propia estructura y el standby namenode lee esta información
- ❖ El standby reciba esta información instantáneamente
- ❖ Con estos elementos ya tendríamos una HA manual: Si se cae el primary podemos convertir el standby en primary

6. ALTA DIPONIBILIDAD HDFS

- ❖ Si queremos una disponibilidad automática voy a tener que tener otros componentes.
- ❖ Aparece un elemento que es el zookeeper failover controler (ZFC), que es como un agente que controla la disponibilidad de los nodos.
- ❖ Estos son los que se relacionan con los servidores zookeeper. Cuando los servidores Zookeeper detectan que a través del ZFC tenemos algún problema en alguno de los primary, va a intentar saltar automáticamente al standby sin necesidad de intervención manual.
- ❖ Vamos a tener otros componentes dentro de nuestro cluster. Además de los típicos se añaden los journals, el standby, los ZF controlers y los zkservers

7. CONFIGURACION HA EN HDFS

Una vez vistas las características que tiene una alta disponibilidad dentro de Big Data a nivel de HDFS, vamos a ver todas las propiedades que tenemos que configurar antes de arrancarlo. Tenemos que tocar en dos ficheros: **core-site.xml** y **hdfs-site.xml**.

Paso 1. Dentro del **core-site.xml** le indicaremos un nombre con el que vamos a definir a nuestro cluster. Hemos puesto ha-cluster pero en realidad se puede poner cualquier nombre con el que identificar nuestro cluster

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <!-- <value>hdfs://nodo1:9000</value> -->
    <value>hdfs://ha-cluster</value>
  </property>
  <property>
    <name>hadoop.proxyuser.hue.hosts</name>
    <value>*</value>
  </property>
```

7. CONFIGURACION HA EN HDFS

Paso 2. Dentro de **hdfs-site.xml** vamos a configurar la mayoría de las propiedades de nuestro cluster de nuestra alta disponibilidad.

En la propiedad `dfs.nameservices` ponemos el mismo valor elegido que hemos puesto para definir nuestro clúster

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.nameservices</name>
  <value>ha-cluster</value>
</property>
```

7. CONFIGURACION HA EN HDFS

Paso 3. En la propiedad `dfs.ha.namenodes.ha-cluster` tenemos los nodos que participan del clúster, es decir cuáles van a ser el nodo activo y el nodo pasivo.

```
<property>
  <name>dfs.ha.namenodes.ha-cluster</name>
  <value>nodo1,nodo2</value>
</property>
```

Paso 4. Ponemos la propiedad `dfs.permissions` a `false`. Con `false` nos permite funcionar sin problemas con nuestro cluster

```
<property>
  <name>dfs.permissions</name>
  <value>>false</value>
</property>
```

7. CONFIGURACION HA EN HDFS

Paso 5. A continuación debemos definir los dos nodos y por qué puerto van a estar escuchando. Mediante la propiedad `dfs.namenode.rpc-address.ha-cluster.nodoX` vamos a indicarle porque puertos va a estar escuchando los nodos activo y pasivo de nuestro clúster.

```
<property>
  <name>dfs.namenode.rpc-address.ha-cluster.nodo1</name>
  <value>nodo1:9000</value>
</property>

<property>
  <name>dfs.namenode.rpc-address.ha-cluster.nodo2</name>
  <value>nodo2:9000</value>
</property>
```

7. CONFIGURACION HA EN HDFS

Paso 6. Debemos hacer lo mismo para el protocolo HTTP. Indicamos para el nodo1 y para el nodo2 porqué puertos van a estar escuchando.

```
<property>
  <name>dfs.namenode.http-address.ha-cluster.nodo1</name>
  <value>nodo1:50070</value>
</property>

<property>
  <name>dfs.namenode.http-address.ha-cluster.nodo2</name>
  <value>nodo2:50070</value>
</property>
```

7. CONFIGURACION HA EN HDFS

Paso 7. Definiremos dónde van a estar los journals. Los journals van a tener el fichero edits, de forma que estos datos se pueden ir pasando entre el activo y el pasivo.

Configuraremos 3 nodos de journal, nodo1, nodo2 y nodo3. Están escuchando por el puerto 8485. El valor empieza con qjournal y finaliza con ha-cluster, el nombre clúster

Los journals son obligatorios pueden estar en realidad en cualquier nodo del cláustro. Los hemos puesto en todos los nodos por sencillez. Cuando arranquemos todo el clúster automáticamente estos tres procesos Journal también se arrancan

```
<property>  
  <name>dfs.namenode.shared.edits.dir</name>  
  <value>qjournal://nodo3:8485;nodo2:8485;nodo1:8485/ha-cluster</value>  
</property>
```

7. CONFIGURACION HA EN HDFS

Paso 8. Debemos indicar en que directorio se van a guardar los ficheros edits en cada journal. Lo pondremos en /datos/jn. Aquí también tenemos el namenode, y los datanode por sencillez

```
<property>  
  <name>dfs.journalnode.edits.dir</name>  
  <value>/datos/jn</value>  
</property>
```

NOTA: Con estos dos últimos parámetros definimos qué nodos de journal tenemos y dónde vamos a dejar el fichero para trabajar luego.

7. CONFIGURACION HA EN HDFS

Paso 9. En la propiedad `client.failover`, indicamos la clase que está definiendo el failover. Y luego indicamos que queremos failover automático

```
<property>
  <name>dfs.client.failover.proxy.provider.ha-cluster</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
</property>

<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
```

NOTA: Indicamos que vamos a tener un zookeeper que va a estar cuidando de pasar el nodo standby a primary en caso de caída de este

7. CONFIGURACION HA EN HDFS

Paso 10. Si hacemos failover automático, tenemos que poner también la propiedad `ha.zookeeper.quorum` que básicamente indica en que nodos están los zookeepers y en qué puertos están escuchando. Creamos los zookeepers en los 3 nodos, que escuchan los clientes por el puerto 2181.

```
<property>  
  <name>ha.zookeeper.quorum</name>  
  <value>nodo1:2181,nodo2:2181,nodo3:2181</value>  
</property>
```

NOTA: HDFS lanzará el proceso `zkcontroller` en el nodo activo y nodo pasivo, pero lo hará automáticamente

7. CONFIGURACION HA EN HDFS

Paso 11. Luego hay que crear un fencing methods. Es el proceso que se percata de que el nodo activo se ha caído y por lo tanto hay que cambiar al pasivo. Hay dos métodos: SSH o Shell, pero ssh es el recomendado porque ya esta configurado).

```
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>
```

Paso 12. Como vamos a utilizar SSH necesitamos indicarle dónde están las claves privadas para que se puedan intercambiar las conexiones.

```
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/hadoop/.ssh/id_rsa</value>
</property>
```



8. ARRANCAR HDFS EN HA

Ya tenemos los ficheros configurados y ahora vamos a ver cómo montar el clúster en modo real de alta disponibilidad

Paso 1. Parar todo el clúster si lo tenemos arrancado

stop-dfs.sh

stop-yarn.sh

Paso 2. Borrarnos los directorios de /datos **SOLO SI QUEREMOS CREAR EL CLUSTER DESDE 0**

Paso 3. Copiamos los ficheros core-site.xml y hdfs-site.xml en todos los nodos mediante scp. Todos los nodos tienen que compartir los ficheros de configuración

```
hadoop@nodol1:/opt/hadoop/etc/hadoop$ scp hdfs-site.xml nodo2:/opt/hadoop/etc/hadoop/  
hdfs-site.xml                                100% 2722      4.1MB/s   00:00  
hadoop@nodol1:/opt/hadoop/etc/hadoop$ scp hdfs-site.xml nodo3:/opt/hadoop/etc/hadoop/  
hdfs-site.xml                                100% 2722      2.0MB/s   00:00  
hadoop@nodol1:/opt/hadoop/etc/hadoop$ scp core-site.xml nodo2:/opt/hadoop/etc/hadoop/  
core-site.xml                               100% 1079     649.4KB/s 00:00  
hadoop@nodol1:/opt/hadoop/etc/hadoop$ scp core-site.xml nodo3:/opt/hadoop/etc/hadoop/  
core-site.xml                               100% 1079     594.1KB/s 00:00  
hadoop@nodol1:/opt/hadoop/etc/hadoop$ █
```

8. ARRANCAR HDFS EN HA

Paso 4. Comprobamos que tenemos los procesos zookeeper funcionando en los 3 nodos mediante los comandos `jps` o `zkServer.sh status` (los arrancamos si es necesario).

```
hadoop@nodo2:~$ jps
2244 ZooKeeperMain
2023 QuorumPeerMain
3181 Jps
```

```
hadoop@nodo2:~$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/./conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
hadoop@nodo2:~$
```

```
127 hadoop@nodo1:/opt/hadoop/etc/hadoop$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/./conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
hadoop@nodo1:/opt/hadoop/etc/hadoop$ jps
233975 Jps
2287 QuorumPeerMain
hadoop@nodo1:/opt/hadoop/etc/hadoop$ █
```

```
hadoop@nodo3:~$ zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/hadoop/zoo/bin/./conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
hadoop@nodo3:~$ jps
1985 QuorumPeerMain
2738 Jps
hadoop@nodo3:~$
```

NOTA: QuorumPeerMain es el proceso de zookeeper

8. ARRANCAR HDFS EN HA

Paso 5. Ahora arrancaremos los journals en los tres nodos ya que lo indicamos en el fichero de configuración. Es lo primero que hay que hacer para montar un clúster HA. El journal es el proceso que va a recibir los cambios, los logs que se van realizando dentro de nuestro clúster

hdfs --daemon start journalnode

Es una manera corta de arrancar y parar un determinado proceso de nuestro clúster de manera individual, en vez de utilizar start-dfs o start-yarn.

```
hadoop@nodo1:/datos$ hadoop-daemon.sh start journalnode
WARNING: Use of this script to start HDFS daemons is deprecated.
WARNING: Attempting to execute replacement "hdfs --daemon start" instead.
hadoop@nodo1:/datos$ jps
270305 JournalNode
270497 Jps
2287 QuorumPeerMain
hadoop@nodo1:/datos$
```

8. ARRANCAR HDFS EN HA

Paso 5bis. Arrancamos los journals en el resto de nodos (nodo2 y nodo3)

```
hadoop@nodo2:~$ hadoop-daemon.sh start journalnode
WARNING: Use of this script to start HDFS daemons is deprecated.
WARNING: Attempting to execute replacement "hdfs --daemon start" instead.
hadoop@nodo2:~$ jps
3377 JournalNode
2023 QuorumPeerMain
3418 Jps
hadoop@nodo2:~$ █
```

```
hadoop@nodo3:~$ hadoop-daemon.sh start journalnode
WARNING: Use of this script to start HDFS daemons is deprecated.
WARNING: Attempting to execute replacement "hdfs --daemon start" instead.
journalnode is running as process 2794. Stop it first and ensure /tmp/hadoop-hadoop-journalnode.pid
file is empty before retry.
hadoop@nodo3:~$ jps
1985 QuorumPeerMain
2794 JournalNode
2958 Jps
hadoop@nodo3:~$ █
```

8. ARRANCAR HDFS EN HA

Paso 6. Los journals van a empezar a crear sus ficheros en el directorio /datos/jn. Comprobamos que en los 3 nodos tenemos el directorio/datos/jn

```
hadoop@nodo1:~$ cd /datos
hadoop@nodo1:/datos$ ls
jn namenode zoo
hadoop@nodo1:/datos$
```

```
hadoop@nodo2:~$ cd /datos
hadoop@nodo2:/datos$ ls
datanode jn zoo
hadoop@nodo2:/datos$
```

```
hadoop@nodo3:~$ cd /datos
hadoop@nodo3:/datos$ ls
datanode jn zoo
hadoop@nodo3:/datos$
```


8. ARRANCAR HDFS EN HA

Paso 7. Vamos al nodo1. Creamos de nuevo el HDFS del cluster. **SOLO SI QUEREMOS CREAR EL CLUSTER DESDE CERO. NO HACER SI QUEREMOS PRESERVAR LO REALIZADO HASTA AHORA**

→ **hdfs namenode -format**

Crea todo el sistema de ficheros de Hadoop. Crea el directorio /datos/namenode, además del jn del journal y zoo del zookeeper

```
2023-11-22 22:06:22,562 INFO namenode.FSImage: Allocated new BlockPoolId: BP-247006429-192.168.0.101-1700690782562
2023-11-22 22:06:22,562 INFO common.Storage: Will remove files: [/datos/namenode/current/seen_txid, /datos/namenode/c
2023-11-22 22:06:22,576 INFO common.Storage: Storage directory /datos/namenode has been successfully formatted.
2023-11-22 22:06:22,634 INFO namenode.FSImageFormatProtobuf: Saving image file /datos/namenode/current/fsimage.ckpt.0
2023-11-22 22:06:22,725 INFO namenode.FSImageFormatProtobuf: Image file /datos/namenode/current/fsimage.ckpt_00000000
2023-11-22 22:06:22,730 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-11-22 22:06:22,774 INFO namenode.FSNamesystem: Stopping services started for active state
2023-11-22 22:06:22,774 INFO namenode.FSNamesystem: Stopping services started for standby state
2023-11-22 22:06:22,778 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-11-22 22:06:22,779 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at nod01/192.168.0.101
*****/
hadoop@nod01:/datos/jn$
```

Paso 8. En el nodo1 arrancamos el namenode

hdfs --daemon start namenode

```
hadoop@nod01:~$ jps
88469 NameNode
14646 QuorumPeerMain
88621 Jps
17742 JournalNode
hadoop@nod01:~$
```



8. ARRANCAR HDFS EN HA

Paso 9. Vamos al nodo2, el cual en el fichero de configuración hemos decidido que será el nodo pasivo.

Tenemos que sincronizar este nodo con el namenode del nodo1, para poder convertir el nodo2 en el nodo pasivo.

hdfs namenode -bootstrapStandby

Comando que sincroniza con el namenode del nodo1, y convierte el nodo2 en nodo standby

```
2023-11-22 22:12:24,870 INFO common.Storage: Storage directory /datos/namenode has been successfully formatted.
2023-11-22 22:12:24,891 INFO common.Util: Assuming 'file' scheme for path /datos/namenode in configuration.
2023-11-22 22:12:24,891 INFO common.Util: Assuming 'file' scheme for path /datos/namenode in configuration.
2023-11-22 22:12:24,914 INFO namenode.FSEditLog: Edit logging is async:true
2023-11-22 22:12:24,969 INFO namenode.TransferFsImage: Opening connection to http://nodo1:50070/imagetransfer?getimage=tstrapstandby=true
2023-11-22 22:12:25,046 INFO common.Util: Combined time for file download and fsync to all disks took 0,00s. The file d
t/fsimage.ckpt_00000000000000000000 took 0,00s.
2023-11-22 22:12:25,047 INFO namenode.TransferFsImage: Downloaded file fsimage.ckpt_00000000000000000000 size 398 bytes.
2023-11-22 22:12:25,057 INFO ha.BootstrapStandby: Skipping InMemoryAliasMap bootstrap as it was not configured
2023-11-22 22:12:25,061 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at nodo2/192.168.0.102
*****/
hadoop@nodo2:/datos/jn$
```

8. ARRANCAR HDFS EN HA

Paso 9bis. Si todo es correcto se va a conectar al nodo1 y va a hacer una copia del momento en el que está el namenode

```
2023-11-21 21:29:08,703 INFO common.Util: Assuming 'file' scheme for path /d
configuration.
=====
About to bootstrap Standby ID nodo2 from:
    Nameservice ID: ha-cluster
    Other Namenode ID: nodo1
    Other NN's HTTP address: http://nodo1:50070
    Other NN's IPC address: nodo1/192.168.0.101:9000
    Namespace ID: 824775038
    Block pool ID: BP-1921597098-192.168.0.101-1700231278768
    Cluster ID: CID-761e915d-1e01-4668-bf6b-675232eb3a8f
    Layout version: -66
    isUpgradeFinalized: true
=====
Re-format filesystem in Storage Directory root= /datos/namenode; location= n
2023-11-21 21:29:18,186 INFO common.Storage: Will remove files: [/datos/name
```

Indica que ha conseguido conectarse al namenode del nodo1 por el puerto 9000 junto con el nombre del clúster
Se ve como ha sincronizado con el fichero FS Image.

8. ARRANCAR HDFS EN HA

Paso 10. En el proceso de sincronización se crea el directorio /datos/namenode. Si tenemos el directorio datanode de antes, lo podemos borrar

```
hadoop@nodo2:/datos$ ls
datanode  jn  namenode  zoo
hadoop@nodo2:/datos$ rm -r datanode
hadoop@nodo2:/datos$ ls
jn  namenode  zoo
hadoop@nodo2:/datos$
```

Paso 11. Arrancamos el namenode standby en nodo2.

hdfs --daemon start namenode

```
hadoop@nodo2:/datos$ hdfs --daemon start namenode
hadoop@nodo2:/datos$
```

8. ARRANCAR HDFS EN HA

Paso 12. Comprobamos con `jps` que tenemos todos los procesos funcionando.

```
hadoop@nodo1:~$ jps
127665 Jps
88469 NameNode
14646 QuorumPeerMain
17742 JournalNode
hadoop@nodo1:~$
```

```
hadoop@nodo2:/datos$ jps
2065 QuorumPeerMain
2165 JournalNode
2581 Jps
2487 NameNode
hadoop@nodo2:/datos$
```

NOTA: De esta manera ya tenemos funcionando el nodo activo que sería el nodo 1 y el nodo pasivo que sería el nodo 2 en standby

8. ARRANCAR HDFS EN HA

Paso 13. Los zkfc acompañan a los nodos activo y pasivo, controlando que estén vivos para hacer el cambio en caso de fallo. En el nodo1 preparamos y arrancamos el zkfc:

hdfs zkfc -formatZK → formatea el zkController

hdfs --daemon start zkfc → arranca el zkController

```
2023-11-21 22:20:45,995 INFO ha.ActiveStandbyElector: Session connected.
Proceed formatting /hadoop-ha/ha-cluster? (Y or N) Y
2023-11-21 22:21:13,530 INFO ha.ActiveStandbyElector: Recursively deleting /hadoop-ha/ha-cluster from ZK...
2023-11-21 22:21:13,554 INFO ha.ActiveStandbyElector: Successfully deleted /hadoop-ha/ha-cluster from ZK.
2023-11-21 22:21:13,576 INFO ha.ActiveStandbyElector: Successfully created /hadoop-ha/ha-cluster in ZK.
2023-11-21 22:21:13,581 WARN zookeeper.ClientCnxn: An exception was thrown while closing send thread for sessionid 0x200000ed4c90000, likely server has
EndOfStreamException: Unable to read additional data from server sessionid 0x200000ed4c90000, likely server has
    at org.apache.zookeeper.ClientCnxnSocketNIO.doIO(ClientCnxnSocketNIO.java:77)
    at org.apache.zookeeper.ClientCnxnSocketNIO.doTransport(ClientCnxnSocketNIO.java:350)
    at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1290)
2023-11-21 22:21:13,688 INFO zookeeper.ZooKeeper: Session: 0x200000ed4c90000 closed
2023-11-21 22:21:13,689 WARN ha.ActiveStandbyElector: Ignoring stale result from old client with sessionId 0x200000ed4c90000
2023-11-21 22:21:13,689 INFO zookeeper.ClientCnxn: EventThread shut down for session: 0x200000ed4c90000
2023-11-21 22:21:13,690 INFO tools.DFSZKFailoverController: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down DFSZKFailoverController at nodo1/192.168.0.101
*****/
hadoop@nodo1:~$
```

Mirar con jps si ha arrancado. Si no buscar en el directorio de logs para ver que ha pasado y si hay algún problema

```
hadoop@nodo1:~$ hdfs --daemon start zkfc
hadoop@nodo1:~$ jps
88469 NameNode
151015 Jps
14646 QuorumPeerMain
150878 DFSZKFailoverController
17742 JournalNode
hadoop@nodo1:~$
```



8. ARRANCAR HDFS EN HA

Paso 14. Vamos a hacer lo mismo en el nodo2, el nodo pasivo. Preparamos y arrancamos el zkfcController:

hdfs zkfc -formatZK

hdfs --daemon start zkfc

```
Proceed formatting /hadoop-ha/ha-cluster? (Y or N) 2023-11-21 22:39:01,505 INFO ha.ActiveStandbyElector: Se
Y
2023-11-21 22:39:07,834 INFO ha.ActiveStandbyElector: Recursively deleting /hadoop-ha/ha-cluster from ZK..
2023-11-21 22:39:07,910 INFO ha.ActiveStandbyElector: Successfully deleted /hadoop-ha/ha-cluster from ZK.
2023-11-21 22:39:07,959 INFO ha.ActiveStandbyElector: Successfully created /hadoop-ha/ha-cluster in ZK.
2023-11-21 22:39:08,074 INFO zookeeper.ZooKeeper: Session: 0x300000f22850000 closed
2023-11-21 22:39:08,075 WARN ha.ActiveStandbyElector: Ignoring stale result from old client with sessionId
2023-11-21 22:39:08,075 INFO zookeeper.ClientCnxn: EventThread shut down for session: 0x300000f22850000
2023-11-21 22:39:08,078 INFO tools.DFSZKFailoverController: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down DFSZKFailoverController at nodo2/192.168.0.102
*****/
hadoop@nodo2:/datos$
```

```
hadoop@nodo2:/datos$ hdfs --daemon start zkfc
hadoop@nodo2:/datos$ jps
2065 QuorumPeerMain
2165 JournalNode
3093 DFSZKFailoverController
2487 NameNode
3145 Jps
hadoop@nodo2:/datos$
```


8. ARRANCAR HDFS EN HA

Paso 15. Paramos todo el cluster con **stop-dfs.sh**

```
hadoop@nodo1:~$ stop-dfs.sh
Stopping namenodes on [nodo1 nodo2]
Stopping datanodes
Stopping journal nodes [nodo2 nodo3 nodo1]
Stopping ZK Failover Controllers on NN hosts [nodo1 nodo2]
hadoop@nodo1:~$
```

Como hadoop lee los ficheros de configuración y conoce la estructura creada, stop-dfs.sh sabe que tiene que parar los namenodes (del nodo 1 y 2), los datanodes, los journalnodes (de nodo1, 2 y 3), y los zkfailover controller.

8. ARRANCAR HDFS EN HA

Paso 16. Arrancamos de nuevo el clúster para comprobar que todo arranca satisfactoriamente con **start-dfs.sh**

Primero arranca los namenodes, los datanodes, los journalnodes y luego los zkfailover controllers. No arranca el secondary namenode, el proceso ayudante del namenode en HDFS, puesto que al estar trabajando en alta disponibilidad ya no hace falta

Los zkServers hay que arrancarlos por separado, es lo unico que no arranca start-dfs

```
hadoop@nodo1:~$ start-dfs.sh
Starting namenodes on [nodo1 nodo2]
Starting datanodes
Starting journal nodes [nodo2 nodo3 nodo1]
Starting ZK Failover Controllers on NN hosts [nodo1 nodo2]
hadoop@nodo1:~$
```

9. COMPROBACION CLUSTER FUNCIONA

Vamos a comprobar que una vez hemos configurado y arrancado nuestro clúster, funciona en HA

Paso 1. Ejecutamos jps en los 3 nodos. En el nodo1 tenemos el journalnode, el zkFailover Controller, el namenode, y el zookeeper (quorumPeermain). En el nodo2 debemos de tener lo mismo: el namenode, zookeeper, el journalnode, y el Failover. En el nodo3 tendremos el datanode, el zookeeper, y el journalnode

```
hadoop@nodo1:~$ jps
4817 QuorumPeerMain
6661 Jps
6102 JournalNode
6374 DFSZKFailoverController
5775 NameNode
hadoop@nodo1:~$
```

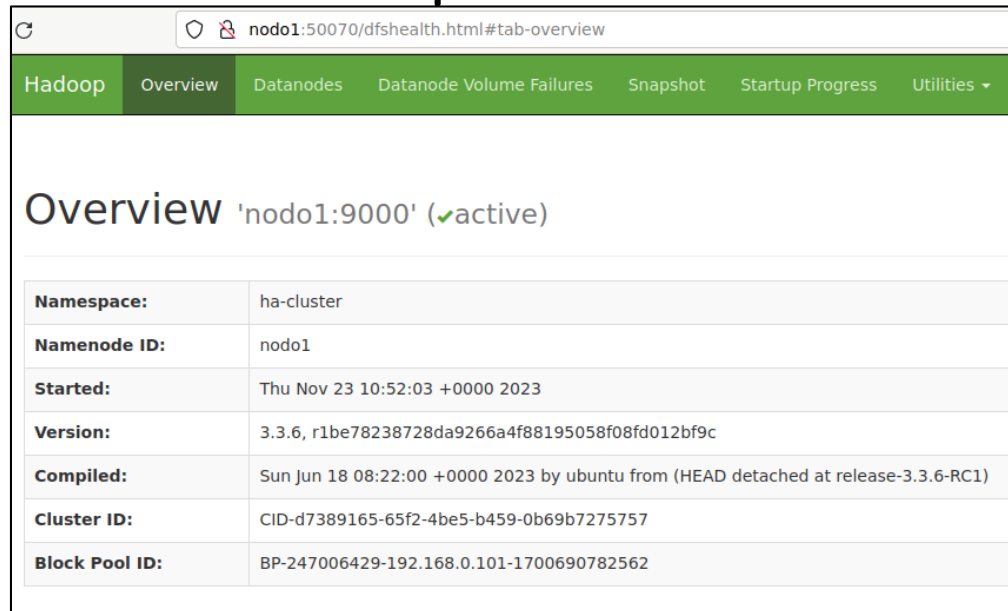
```
hadoop@nodo2:~$ jps
2016 QuorumPeerMain
2154 NameNode
2269 JournalNode
2431 DFSZKFailoverController
2495 Jps
hadoop@nodo2:~$
```

```
hadoop@nodo3:/datos$ jps
1990 QuorumPeerMain
2614 Jps
2550 JournalNode
2415 DataNode
hadoop@nodo3:/datos$
```

NOTA: Lo ideal sería tener los journalnode en sus maquinas independientes.

9. COMPROBACION CLUSTER FUNCIONA

Paso 2. Abrimos el navegador y vamos a la administración de hadoop-hdfs: nodo1:50070 (indicada en hdfs-site.xml). Indica que estamos en el cluster ha-cluster y en el nodo1, el cual además es el nodo primario o activo



Overview 'nodo1:9000' (✓active)

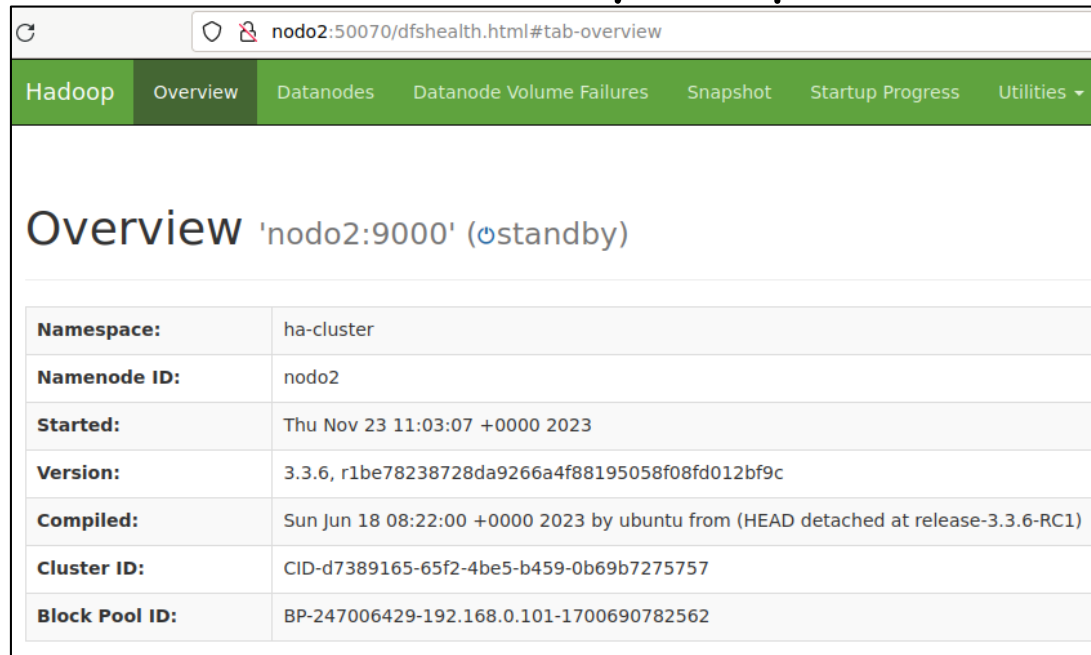
Namespace:	ha-cluster
Namenode ID:	nodo1
Started:	Thu Nov 23 10:52:03 +0000 2023
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 08:22:00 +0000 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-d7389165-65f2-4be5-b459-0b69b7275757
Block Pool ID:	BP-247006429-192.168.0.101-1700690782562

NameNode Storage

Storage Directory	Type	State
/datos/namenode	IMAGE_AND_EDITS	Active

9. COMPROBACION CLUSTER FUNCIONA

Paso 3. Abrimos una nueva pestaña y vamos a la web del nodo2 → nodo2:50070. Podemos comprobar que nodo2 está funcionando como standby del primario



Overview 'nodo2:9000' (standby)	
Namespace:	ha-cluster
Namenode ID:	nodo2
Started:	Thu Nov 23 11:03:07 +0000 2023
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 08:22:00 +0000 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-d7389165-65f2-4be5-b459-0b69b7275757
Block Pool ID:	BP-247006429-192.168.0.101-1700690782562

Los Journal Manager
está en los nodos
donde los hayamos
configurado

NameNode Journal Status

Current transaction ID: 361

Journal Manager	State
OJM to [192.168.0.103:8485, 192.168.0.102:8485, 192.168.0.101:8485]	Writing segment beginning at txid 361, 192.168.0.103:8485 (Written txid 361), 192.168.0.102:8485 (Written txid 361), 192.168.0.101:8485 (Written txid 361)
FileJournalManager(root=/datos/namenode)	EditLogFileOutputStream(/datos/namenode/current/edits_inprogress_00000000000000000361)

9. COMPROBACION CLUSTER FUNCIONA

Paso 4. También lo podemos ver desde línea de comandos:

hdfs haadmin -getServiceState nodo1

hdfs haadmin -getServiceState nodo2

hdfs haadmin -getAllServiceState

```
hadoop@nodo1:/datos$ hdfs haadmin -getServiceState nodo1
active
hadoop@nodo1:/datos$ hdfs haadmin -getServiceState nodo2
standby
hadoop@nodo1:/datos$ hdfs haadmin -getAllServiceState
nodo1:9000                                active
nodo2:9000                                standby
hadoop@nodo1:/datos$
```

9. COMPROBACION CLUSTER FUNCIONA

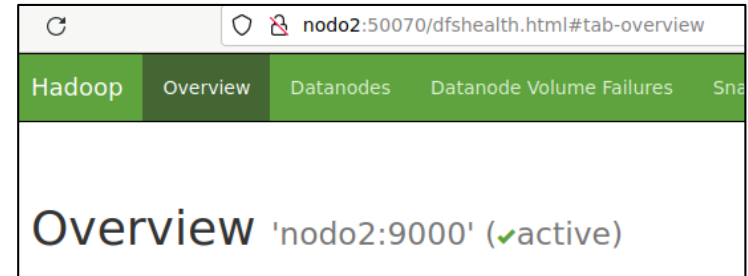
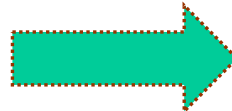
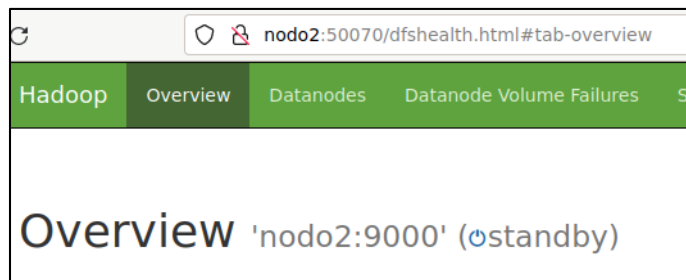
Paso 5. Vamos a hacer un failover manual, parando el proceso namenode del nodo1. Hacemos jps y kill -9 <idpro>

```
hadoop@nodo1:/datos$ jps
442803 Jps
402014 DFSZKFailoverController
442687 JournalNode
6363 QuorumPeerMain
401435 NameNode
hadoop@nodo1:/datos$ kill -9 401435
hadoop@nodo1:/datos$ jps
447745 Jps
402014 DFSZKFailoverController
442687 JournalNode
6363 QuorumPeerMain
hadoop@nodo1:/datos$
```

Si hacemos jps vemos que esta todo arrancado menos el namenode

9. COMPROBACION CLUSTER FUNCIONA

Paso 6. Vamos a la web del nodo2 y hacemos un reload. Si todo es correcto, nodo2 se ha convertido en activo, ya que ha tenido que conmutar al haber caído el Nodo1.

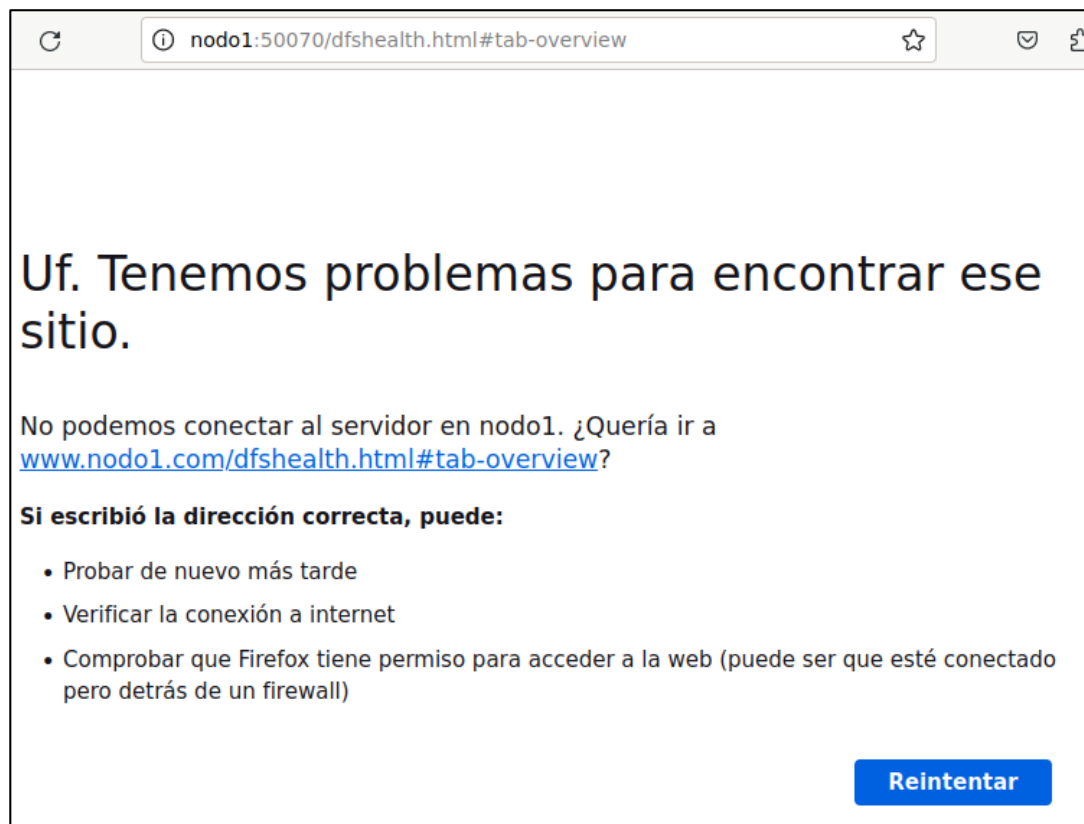


Current transaction ID: 422	
Journal Manager	State
QJM to [192.168.0.103:8485, 192.168.0.102:8485, 192.168.0.101:8485]	Writing segment beginning at txid 422. 192.168.0.103:8485 (Written txid 422), 192.168.0.102:8485 (Written txid 422), 192.168.0.101:8485 (Written txid 422)
FileJournalManager(root=/datos/namenode)	EditLogFileOutputStream(/datos/namenode/current/edits_inprogress_00000000000000000422)

En este momento ya se han pasado los datos de un nodo a otro. En maquina no tan rapidas, si hacemos reload, esta información puede ir cambiando de forma mas lenta

9. COMPROBACION CLUSTER FUNCIONA

Paso 7. Si vamos al nodo1, ya no nos podemos conectar puesto que lo hemos tumbado



9. COMPROBACION CLUSTER FUNCIONA

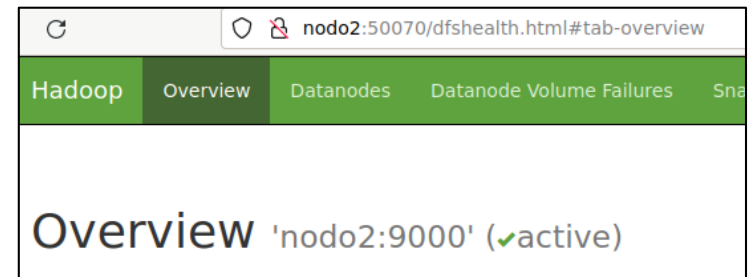
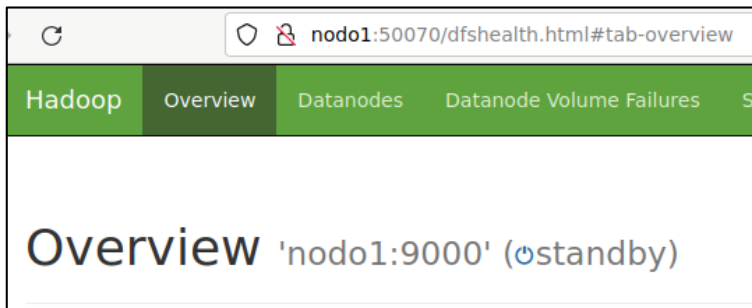
Paso 8. En el nodo1 probamos de rearrancar el namenode:

hdfs --daemon start namenode

```
hadoop@nodo1:/$ jps
402014 DFSZKFailoverController
442687 JournalNode
6363 QuorumPeerMain
459912 Jps
hadoop@nodo1:/$ hdfs --daemon start namenode
jps
460079 NameNode
402014 DFSZKFailoverController
442687 JournalNode
460205 Jps
6363 QuorumPeerMain
hadoop@nodo1:/$
```

9. COMPROBACION CLUSTER FUNCIONA

Paso 9. Hacemos un reload de la pantalla y podemos comprobar cómo el nodo1 se ha convertido en un standby. El nodo2 sigue siendo el activo



Paso 10. También podemos ver la situación del clúster mediante las opciones del comando `hdfs haadmin`:

```
hadoop@nodo1:/$ hdfs haadmin
Usage: haadmin [-ns <nameserviceId>]
       [-checkHealth <serviceId>]
       [-failover [--forcefence] [--forceactive] <serviceId> <serviceId>]
       [-getAllServiceState]
       [-getServiceState <serviceId>]
       [-help <command>]
       [-transitionToActive [--forceactive] <serviceId>]
       [-transitionToObserver <serviceId>]
       [-transitionToStandby <serviceId>]
```

9. COMPROBACION CLUSTER FUNCIONA

Paso 11. Con la opción `-getAllServiceState` vemos rápidamente el nodo que esta activo y el que esta en standby

```
hadoop@nodo1:/$ hdfs haadmin -getAllServiceState
nodo1:9000 standby
nodo2:9000 active
hadoop@nodo1:/$
```

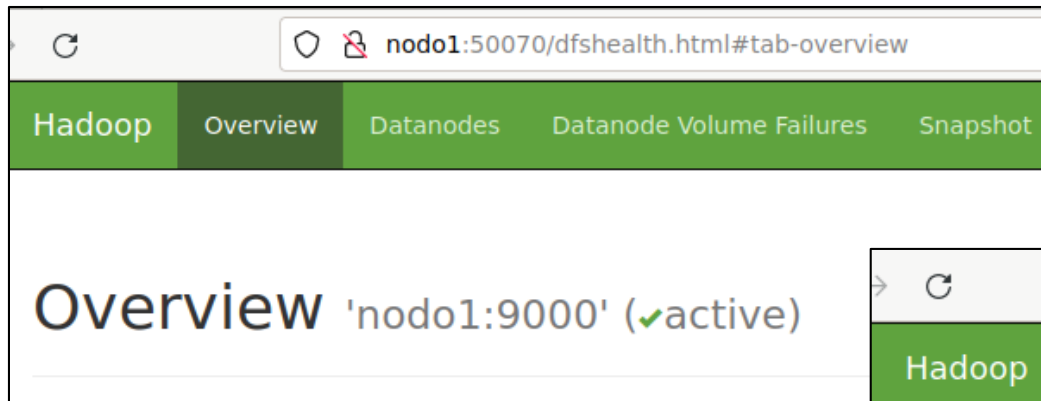
Paso 12. Con opción failover podemos hacer un failover manual. Provocaremos un failover del nodo2 (que en este momento está activo) al nodo1.

hdfs haadmin failover nodo2 nodo1

```
255 hadoop@nodo1:/$ hdfs haadmin -getAllServiceState
nodo1:9000 standby
nodo2:9000 active
hadoop@nodo1:/$ hdfs haadmin -failover nodo2 nodo1
Failover to NameNode at nodo1/192.168.0.101:9000 successful
hadoop@nodo1:/$ hdfs haadmin -getAllServiceState
nodo1:9000 active
nodo2:9000 standby
hadoop@nodo1:/$
```

9. COMPROBACION CLUSTER FUNCIONA

Paso 13. Si vamos a nivel de web nos indica exactamente lo mismo: nodo1 es el activo y el nodo2 es el standby



NOTA: Ya no existe uno de los puntos de fallo que tenía nuestro entorno, con la implantación de la alta disponibilidad HDFS. En este momento ya podemos tener una caída del sistema sin temor a tener pérdidas