节点规划

```
namenode1    namenode   resourcemanager  zkfc
namenode2    namenode   resourcemanager  zkfc   his

datanode1    datanode   nodemanager
datanode2    datanode   nodemanager
datanode3    datanode   nodemanager

journalnode1 journal node
journalnode2 journal node
journalnode3 journal node

zookeeper1   zookeeper
zookeeper2   zookeeper
zookeeper3   zookeeper
```

相关命令

docker

```
sudo service docker start       //启动docker
sudo service docker restart     //重启docker

docker start $(docker ps -a -q) //启动所有容器
docker stop $(docker ps -a -q) //  stop停止所有容器
```

容器启动停止

```
#启动
docker start namenode1 namenode2 datanode1 datanode2 datanode3 journalnode1
journalnode2 journalnode3 zookeeper1 zookeeper2 zookeeper3 mysql_hive

#重新启动
docker restart namenode1 namenode2 datanode1 datanode2 datanode3 journalnode1
journalnode2 journalnode3 zookeeper1 zookeeper2 zookeeper3 mysql_hive

#停止
docker stop namenode1 namenode2 datanode1 datanode2 datanode3 journalnode1
journalnode2 journalnode3 zookeeper1 zookeeper2 zookeeper3 mysql_hive
```

拉取镜像

先把这个下了

https://github.com/Marcel-Jan/docker-hadoop-spark

拉取journalnode镜像

```
docker pull colovu/hadoop-journalnode
```

拉取zookeeper镜像

```
 docker search zookeeper
 docker pull zookeeper
 docker images              //查看下载的本地镜像
 docker inspect zookeeper   //查看zookeeper详细信息
```

创建网络

```
docker network create  --subnet=172.21.0.0/16   --ip-range=172.21.1.0/24   --
gateway=172.21.1.1  net1



docker network create  --subnet=172.19.0.0/16   --ip-range=172.19.1.0/24   --
gateway=172.19.1.1  net2



docker network create  --subnet=172.20.0.0/16   --ip-range=172.20.1.0/24   --
gateway=172.20.1.1  net3
```

bridge网络只能有一个子网因此需要建立三个子网

创建容器

（不要用172.17网段，该网段已被使用）

```
#注意每个容器创建后配密码，我在每个容器里配密码时，密码与主机名相同
#密码配置命令
passwd

# namenode

sudo docker run -it -h namenode1 --name namenode1 --net=net1  --ip 172.21.1.2
bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

sudo docker run -it -h namenode2 --name namenode2 --net=net2  --ip 172.19.1.2
bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

# datanode

sudo docker run -it -h datanode1 --name datanode1 --net=net1  --ip 172.21.1.3
bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

sudo docker run -it -h datanode2 --name datanode2 --net=net2  --ip 172.19.1.3
bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit
```

```
sudo docker run -it -h datanode3 --name datanode3 --net=net3  --ip 172.20.1.3
bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

# journalnode

sudo docker run -it -h journalnode1 --name journalnode1 --net=net1  --ip
172.21.1.4 bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

sudo docker run -it -h journalnode2 --name journalnode2 --net=net2  --ip
172.19.1.4 bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

sudo docker run -it -h journalnode3 --name journalnode3 --net=net3  --ip
172.20.1.4 bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8 /bin/bash
exit

# zookeeper

sudo docker run -it -h zookeeper1 --name zookeeper1 --net=net1  --ip 172.21.1.5
zookeeper:latest /bin/bash
 cd ..
 mv apache-zookeeper-3.9.1-bin zookeeper
 exit

sudo docker run -it -h zookeeper2 --name zookeeper2 --net=net2  --ip 172.19.1.5
zookeeper:latest /bin/bash
 cd ..
 mv apache-zookeeper-3.9.1-bin zookeeper
 exit

sudo docker run -it -h zookeeper3 --name zookeeper3 --net=net3  --ip 172.20.1.5
zookeeper:latest /bin/bash
 cd ..
 mv apache-zookeeper-3.9.1-bin zookeeper
 exit
```

后面的/bin/bash的作用是表示**载入容器后运行bash**,docker中必须要保持一个进程的运行，要不然整个容器启动后就会马上kill itself， 这个/bin/bash就表示启动容器后启动bash。

创建一个网络，把这些容器都加进去保证能互相通信

```
 docker network create  --subnet=172.22.0.0/16   --ip-range=172.22.1.0/24   --
gateway=172.22.1.1  cloud_computing_network
```

```
docker network connect  cloud_computing_network namenode1 --ip 172.22.1.0
docker network connect  cloud_computing_network namenode2 --ip 172.22.1.2

docker network connect  cloud_computing_network datanode1 --ip 172.22.1.3
docker network connect  cloud_computing_network datanode2 --ip 172.22.1.4
docker network connect  cloud_computing_network datanode3 --ip 172.22.1.5

docker network connect  cloud_computing_network journalnode1 --ip 172.22.1.6
docker network connect  cloud_computing_network journalnode2 --ip 172.22.1.7
docker network connect  cloud_computing_network journalnode3 --ip 172.22.1.8

docker network connect  cloud_computing_network zookeeper1 --ip 172.22.1.9
docker network connect  cloud_computing_network zookeeper2 --ip 172.22.1.10
docker network connect  cloud_computing_network zookeeper3 --ip 172.22.1.11
```

补全zookeeper中的文件

由于zookeeper的三个容器里java/bin缺少jps，所以从windows里直接拷过来

zookeeper中/zookeeper/conf文件夹现在没东西，东西在/conf中，要复制过去

```
#先要退出容器，回到ubuntu宿主机，因为源路径有空格所以要加引号
docker cp "/mnt/d/Program Files/Java/jdk-17/bin/jps.exe"
zookeeper1:/opt/java/openjdk/bin
docker exec -it zookeeper1 /bin/bash
cp -r /conf /zookeeper
cd /opt/java/openjdk/bin
mv jps.exe jps
chown root:root jps
exit
docker cp zookeeper1:/conf zookeeper1:/zookeeper

docker cp "/mnt/d/Program Files/Java/jdk-17/bin/jps.exe"
zookeeper2:/opt/java/openjdk/bin
docker exec -it zookeeper2 /bin/bash
cp -r /conf /zookeeper
cd /opt/java/openjdk/bin
mv jps.exe jps
chown root:root jps
exit
docker cp zookeeper2:/conf zookeeper2:/zookeeper

docker cp "/mnt/d/Program Files/Java/jdk-17/bin/jps.exe"
zookeeper3:/opt/java/openjdk/bin
docker exec -it zookeeper3 /bin/bash
cp -r /conf /zookeeper
cd /opt/java/openjdk/bin
mv jps.exe jps
chown root:root jps
exit
docker cp zookeeper3:/conf zookeeper3:/zookeeper
```

进入容器

```
docker exec -it 容器名 /bin/bash

docker exec -it namenode1 /bin/bash
docker exec -it namenode2 /bin/bash

docker exec -it datanode1 /bin/bash
docker exec -it datanode2 /bin/bash
docker exec -it datanode3 /bin/bash

docker exec -it journalnode1 /bin/bash
docker exec -it journalnode2 /bin/bash
docker exec -it journalnode3 /bin/bash

docker exec -it zookeeper1 /bin/bash
docker exec -it zookeeper2 /bin/bash
docker exec -it zookeeper3 /bin/bash

docker exec -it mysql_hive /bin/bash
```

下载各种工具

在每个容器中下载允许连接https的包及vim、ssh

```
#别用下面这个
apt-get update
apt-get install vim
#报错E: Unable to locate package vim



#这样可以
# 先通过vscode换源，进入/etc/apt/sources.list
#以下是替换的文本内容，直接粘过去（zookeeper的三个容器不用换源，跳到从第21行开始执行）

deb https://mirrors.ustc.edu.cn/ubuntu/ focal main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-security main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-updates main restricted universe
multiverse
deb https://mirrors.ustc.edu.cn/ubuntu/ focal-backports main restricted universe
multiverse



#粘完在每个容器中执行下面语句
cd /usr/lib/apt/methods
ln -s http https
cd /
apt-get update
apt-get install apt-transport-https
apt-get install vim
apt-get install openssh-server
```

[解决更新源强制使用https但出现apt-transport-https not found - 系统安装/使用问题 - ParrotSec中文社区 (parrotsec-cn.org)]

创建hadoop要用到的文件夹

在三个journalnode下的home目录中创建文件夹

```
cd home
mkdir journaldata
```

在namenode2下创建文件夹

```
mkdir /hisdata/
cd hisdata
mkdir staging
mkdir done
mkdir done_intermediate
```

在三个datanode下创建文件夹

```
mkdir hisdata
cd hisdata
mkdir ContinerLogs
```

配置ip主机映射

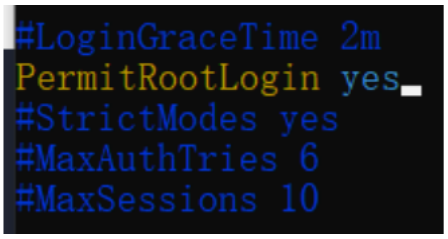
```
#映射如下
172.22.1.0 namenode1
172.22.1.2 namenode2

172.22.1.3 datanode1
172.22.1.4 datanode2
172.22.1.5 datanode3

172.22.1.6 journalnode1
172.22.1.7 journalnode2
172.22.1.8 journalnode3

172.22.1.9 zookeeper1
172.22.1.10 zookeeper2
172.22.1.11 zookeeper3


172.21.1.2 namenode1
172.19.1.2 namenode2

172.21.1.3 datanode1
172.19.1.3 datanode2
172.20.1.3 datanode3

172.21.1.4 journalnode1
172.19.1.4 journalnode2
172.20.1.4 journalnode3

172.21.1.5 zookeeper1
172.19.1.5 zookeeper2
172.20.1.5 zookeeper3
```

改/etc/hosts,添加下面内容（所有hadoop容器都要配）

```
#由于直接改hosts，重启容器后会清空，因此直接在entrypoint.sh里写
#注意不要照抄！！！！！在哪个节点里把哪个删了，因为启动时本节点的写进hosts了，这里是补全集群里其他节点的
vim /entrypoint.sh

echo "172.22.1.0 namenode1" >> /etc/hosts
echo "172.22.1.2 namenode2" >> /etc/hosts

echo "172.22.1.3 datanode1" >> /etc/hosts
echo "172.22.1.4 datanode2" >> /etc/hosts
echo "172.22.1.5 datanode3" >> /etc/hosts

echo "172.22.1.6 journalnode1" >> /etc/hosts
echo "172.22.1.7 journalnode2" >> /etc/hosts
echo "172.22.1.8 journalnode3" >> /etc/hosts

echo "172.22.1.9 zookeeper1" >> /etc/hosts
echo "172.22.1.10 zookeeper2" >> /etc/hosts
echo "172.22.1.11 zookeeper3" >> /etc/hosts
```

配ssh

```
#先在每个容器中打开ssh服务，并允许root方式登录（因为默认不允许）
#检查ssh服务是否开启
service ssh status

#打开ssh服务
service ssh start

vi /etc/ssh/sshd_config
#放开下图中黄蓝行注释，并改为yes
```



```
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
```

```
#添加如下内容
RSAAuthentication yes # 启用 RSA 认证
PubkeyAuthentication yes # 启用公钥私钥配对认证方式
AuthorizedKeysFile .ssh/authorized_keys # 公钥文件路径（和上面生成的文件同）
#退出编辑

#创建.ssh文件夹
mkdir /root/.ssh

#将ssh加入启动项
vim /root/startup_run.sh

#startup_run.sh中添加以下内容
#!/bin/bash
```

```
LOGTIME=$(date "+%Y-%m-%d %H:%M:%S")
echo "[$LOGTIME] startup run..." >>/root/startup_run.log
service ssh start >>/root/startup_run.log

#退出编辑，设置访问权限
chmod +x /root/startup_run.sh

#将脚本加入到启动文件中
vim /root/.bashrc


#/root/.bashrc中追加如下语句
# startup run
if [ -f /root/startup_run.sh ]; then
      /root/startup_run.sh
fi


#重启ssh
service ssh restart
```

```
#namenode1中
ssh-keygen #一路回车

#将公钥复制到其他容器
scp ~/.ssh/id_rsa.pub root@namenode2:~/.ssh/authorized_keys

scp ~/.ssh/id_rsa.pub root@datanode1:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub root@datanode2:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub root@datanode3:~/.ssh/authorized_keys

scp ~/.ssh/id_rsa.pub root@journalnode1:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub root@journalnode2:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub root@journalnode3:~/.ssh/authorized_keys

scp ~/.ssh/id_rsa.pub root@zookeeper1:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub root@zookeeper2:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub root@zookeeper3:~/.ssh/authorized_keys
```

编辑hadoop的配置文件

修改core-site.xml

注意fs.defaultFS在haooop2用的8020端口，但是这里因为我配了rpc，且使用了高可用，因此要写集群的服务名

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

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

<configuration>

    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://nameservice</value>
    </property>

    <property>
        <name>hadoop.http.staticuser.user</name>
        <value>root</value>
    </property>

    <property>
        <name>hadoop.tmp.dir</name>
        <value>/hadoop</value>
    </property>

<!-- ZooKeeper集群的地址和端口。注意，数量一定是奇数，且不少于三个节点-->
    <property>
        <name>ha.zookeeper.quorum</name>
        <value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</value>
    </property>

  <!--修改core-site.xml中的ipc参数,防止出现连接journalnode服务ConnectException,默认
10s-->
  <property>
      <name>ipc.client.connect.max.retries</name>
      <value>100</value>
  </property>
  <property>
      <name>ipc.client.connect.retry.interval</name>
      <value>10000</value>
  </property>

  <property>
      <name>hadoop.proxyuser.root.hosts</name>
      <value>*</value>
  </property>
  <property>
      <name>hadoop.proxyuser.root.groups</name>
      <value>*</value>
  </property>
```

```
</configuration>
```

修改hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

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

<configuration>

<!--指定hdfs的nameservice服务组名字，再次声明hdp1只是一个名字，不是hdp1这个节点，大家觉得别扭可以改成别的，但一定要和core-site.xml中的保持一致 -->
    <property>
        <name>dfs.nameservices</name>
        <value>nameservice</value>
    </property>

    <property>
        <name>dfs.ha.namenodes.nameservice</name>
        <value>namenode1,namenode2</value>
    </property>

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

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

    <property>
        <name>dfs.namenode.handler.count</name>
        <value>50</value>
    </property>
    <!-- 非常重要，因为hadoop搭建在docker内网，我们要在windows中访问hadoop的datanode上传文件 -->
    <property>
        <name>dfs.client.use.datanode.hostname</name>
        <value>true</value>
    </property>
    <property>
        <name>dfs.datanode.use.datanode.hostname</name>
        <value>true</value>
    </property>

    <!-- namenode1的RPC通信地址 -->
    <property>
        <name>dfs.namenode.rpc-address.nameservice.namenode1</name>
        <value>namenode1:9000</value>
    </property>

    <!-- namenode1的http通信地址 -->
    <property>
        <name>dfs.namenode.http-address.nameservice.namenode1</name>
        <value>namenode1:9870</value>
    <!-- 注意，hadoop3中页面访问端口不再是50070而是9870 -->
    </property>

    <!-- namenode2的RPC通信地址 -->
    <property>
        <name>dfs.namenode.rpc-address.nameservice.namenode2</name>
        <value>namenode2:9000</value>
    </property>

    <!-- namenode2的http通信地址 -->
    <property>
        <name>dfs.namenode.http-address.nameservice.namenode2</name>
        <value>namenode2:9870</value>
    </property>

    <!-- 指定NameNode的edits元数据在JournalNode上的存放位置，这个配置本质上的目的就是在zk上使用一个zk节点，保存namenode同步数据，应该注意这里的zk节点说的是zkCli里面的 -->
    <property>
        <name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://journalnode1:8485;journalnode2:8485;journalnode3:8485/nameservice</value>
    </property>

    <!-- 指定JournalNode在本地磁盘存放数据的位置 -->
    <property>
        <name>dfs.journalnode.edits.dir</name>
        <value>/home/journaldata</value>
    </property>

    <!-- 开启NameNode死亡自动切换 -->
    <property>
        <name>dfs.ha.automatic-failover.enabled.nameservice</name>
        <value>true</value>
    </property>

    <!-- 指定该集群出故障时，哪个实现类负责执行故障切换 -->
    <property>
        <name>dfs.client.failover.proxy.provider.nameservice</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
</value>
    </property>

    <!-- 配置隔离机制方法，多个机制用换行分割，即每个机制暂用一行-->
    <!-- 注意一定要加shell(/bin/true)且和前面换行，否则namenode无法宕掉之后自动切换-->
    <property>
        <name>dfs.ha.fencing.methods</name>
        <value>
            sshfence
            shell(/bin/true)
        </value>
    </property>

    <!-- 使用sshfence隔离机制时需要ssh免登陆，因此需要指定公钥路径，大家集群中ssh之后的公钥
文件在哪里就指定那里，一般在用户主目录下的 .ssh文件夹里面-->
    <property>
        <name>dfs.ha.fencing.ssh.private-key-files</name>
        <value>/root/.ssh/authorized_keys</value>
    </property>

    <!-- 配置sshfence隔离机制超时时间 -->
    <property>
        <name>dfs.ha.fencing.ssh.connect-timeout</name>
        <value>30000</value>
    </property>

    <!-- namenode与datanode存储多路径配置，且更改datanode磁盘写入规则 -->
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///${hadoop.tmp.dir}/dfs/data</value>
    </property>

    <property>
        <name>dfs.datanode.fsdataset.volume.choosing.policy</name>

<value>org.apache.hadoop.hdfs.server.datanode.fsdataset.AvailableSpaceVolumeChoo
singPolicy</value>
    </property>

    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///${hadoop.tmp.dir}/dfs/name</value>
    </property>

    <!-- 配置心跳报告周期和检查时间 -->
    <property>
        <name>dfs.heartbeat.interval</name>
        <value>3</value>
    </property>

    <property>
        <name>dfs.namenode.heartbeat.recheck-interval</name>
        <value>3000</value>
    </property>

</configuration>
```

在所有hdfs-site.xml上配置

```
<configuration>
<property>
    <name>dfs.client.use.datanode.hostname</name>
    <value>true</value>
</property>
<property>
    <name>dfs.datanode.use.datanode.hostname</name>
    <value>true</value>
</property>
</configuration>
```

```
#在windows的C:\Windows\System32\drivers\etc\hosts文件中添加
#注意，写自己的wsl地址
172.31.2.42 datanode1
172.31.2.42 datanode2
172.31.2.42 datanode3
#注意，三个datanode都映射到wsl的地址
```

注意，在三个datanode中还要加入datanode的http访问端口，下面三个配置不能每个datanode都放，是datanode几，就加value位置对应的那个，目的是指定该datanode的http通信端口。只在三个datanode上加！！

```
<property>
    <name>dfs.datanode.http.address</name>
    <value>datanode1:9864</value>
</property>

<property>
    <name>dfs.datanode.http.address</name>
    <value>datanode2:9874</value>
</property>

<property>
    <name>dfs.datanode.http.address</name>
    <value>datanode3:9884</value>
</property>
```

修改mapred-site.xml

```
<configuration>
```

```xml
<!-- 指定mr框架为yarn方式 -->
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>

    <!-- 历史服务器的外部服务端口 -->
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>namenode2:10020</value>
    </property>

    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>namenode2:19888</value>
    </property>

    <!--历史服务器的MR相关日志文件存储，该值指向一个hdfs路径，下面两个配置也是-->
    <property>
        <name>yarn.app.mapreduce.am.staging-dir</name>
        <value>/hisdata/staging</value>
    </property>

    <!--已结束的MR-日志-->
    <property>
        <name>mapreduce.jobhistory.done-dir</name>
        <value>/hisdata/done</value>
    </property>

    <!-- 正在进行的MR-日志 -->
    <property>
        <name>mapreduce.jobhistory.intermediate-done-dir</name>
        <value>/hisdata/done_intermediate</value>
    </property>
</configuration>
```

修改yarn-site.xml

```xml
<?xml version="1.0"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>
```

```xml
<!-- Site specific YARN configuration properties -->

<!-- 开启RM高可用 -->
    <property>
        <name>yarn.resourcemanager.ha.enabled</name>
        <value>true</value>
    </property>

    <property>
        <name>yarn.resourcemanager.ha.automatic-failover.enabled</name>
        <value>true</value>
    </property>

    <property>
        <name>yarn.resourcemanager.ha.automatic-failover.embedded</name>
        <value>true</value>
    </property>

    <property>
        <name>yarn.resourcemanager.connect.retry-interval.ms</name>
        <value>2000</value>
    </property>

    <!-- 指定RM的cluster id -->
    <property>
        <name>yarn.resourcemanager.cluster-id</name>
        <value>yrc</value>
    </property>

  <!-- 指定RM的名字,对应主备namenode -->
    <property>
        <name>yarn.resourcemanager.ha.rm-ids</name>
        <value>rm1,rm2</value>
    </property>

    <!-- 分别指定RM的地址 -->
    <property>
        <name>yarn.resourcemanager.hostname.rm1</name>
        <value>namenode1</value>
    </property>

    <property>
        <name>yarn.resourcemanager.hostname.rm2</name>
        <value>namenode2</value>
    </property>

  <!-- 指定zk集群地址 -->
    <property>
        <name>yarn.resourcemanager.zk-address</name>
        <value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</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>

    <!-- 阻止磁盘检查 -->
    <property>
        <name>yarn.nodemanager.vmem-check-enabled</name>
        <value>false</value>
    </property>


  <!--配置外部端口-->
    <!-- yarn对applicationmaster服务的端口 -->
    <property>
        <name>yarn.resourcemanager.scheduler.address.rm1</name>
        <value>namenode1:8030</value>
    </property>

    <property>
        <name>yarn.resourcemanager.scheduler.address.rm2</name>
        <value>namenode2:8030</value>
    </property>

  <!-- yarn对nodemanager服务的地址 -->
    <property>
        <name>yarn.resourcemanager.resource-tracker.address.rm1</name>
        <value>namenode1:8031</value>
    </property>

    <property>
        <name>yarn.resourcemanager.resource-tracker.address.rm2</name>
        <value>namenode2:8031</value>
    </property>

    <!--yarn对客户端服务的端口 -->
    <property>
        <name>yarn.resourcemanager.address.rm1</name>
        <value>namenode1:8032</value>
    </property>

    <property>
        <name>yarn.resourcemanager.address.rm2</name>
        <value>namenode2:8032</value>
    </property>

  <!-- yarn对管理员服务的地址 -->
    <property>
        <name>yarn.resourcemanager.admin.address.rm1</name>
        <value>namenode1:8033</value>
    </property>
    <property>
        <name>yarn.resourcemanager.admin.address.rm2</name>
        <value>namenode2:8033</value>
    </property>

    <!-- yarn web UI服务的地址  -->
    <property>
        <name>yarn.resourcemanager.webapp.address.rm1</name>
        <value>namenode1:8088</value>
    </property>

    <property>
        <name>yarn.resourcemanager.webapp.address.rm2</name>
        <value>namenode2:8088</value>
    </property>

<!--开启Continer日志聚合,否则无法在web页面查看mr的日志-->
    <property>
        <name>yarn.log-aggregation-enable</name>
        <value>true</value>
    </property>

    <!--Continer日志路径，MR任务结束后将从mapred-site.xml配置的MR日志路径中聚合结果存放在
这个目录中，使得日志可以背访问、使用-->
    <property>
        <name>yarn.nodemanager.remote-app-log-dir</name>
        <value>/hisdata/ContinerLogs</value>
    </property>

    <!-- 以下三项配置是修改hadoop对历史MR任务的保存态度，使得重启后yarn-ui的历史MR记录不会丢
失 -->
    <property>
        <!-- 开启作业保留 -->
        <name>yarn.resourcemanager.recovery.enabled</name>
        <value>true</value>
    </property>

    <property>
        <!-- 指定保存与加载历史任务的类，总计三个可选项，但高可用集群必须使用ZK -->
        <name>yarn.resourcemanager.store.class</name>

 <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</v
alue>
    </property>

    <property>
        <!-- yarn重启后等待历史任务被加载的时间 -->
        <name>yarn.resourcemanager.work-preserving-recovery.scheduling-wait-
ms</name>
        <value>10000</value>
    </property>

    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>4096</value>
    </property>

</configuration>
```

修改workers

注意hadoop3中datanode节点已经不再记录在slaves文件中，而是workers文件中

```
datanode1
datanode2
datanode3
```

修改capacity-scheduler.xml

这个文件中有默认值，我们找到如下的配置并修改为对应的值

```
<property>
    <name>yarn.scheduler.capacity.resource-calculator</name>

 <value>org.apache.hadoop.yarn.util.resource.DominantResourceCalculator</value>
</property>
```

同步各容器中的配置文件

发送给其余hadoop节点(路径最前面要加/表示从根节点开始)

```
scp -r /opt/hadoop-3.2.1 root@namenode2:/opt

scp -r /opt/hadoop-3.2.1 root@datanode1:/opt
scp -r /opt/hadoop-3.2.1 root@datanode2:/opt
scp -r /opt/hadoop-3.2.1 root@datanode3:/opt

scp -r /opt/hadoop-3.2.1 root@journalnode1:/opt
scp -r /opt/hadoop-3.2.1 root@journalnode2:/opt
scp -r /opt/hadoop-3.2.1 root@journalnode3:/opt
```

编辑zookeeper的配置文件（每个zookeeper容器）

设置zookeeper的环境变量

```
vscode进入root/.bashrc
```

 在文件末尾添加如下语句(这个每台机器都要配，复制zookeeper文件夹到另一个虚拟机并没有复制.bashrc文件):

```
export ZOOKEEPER_HOME=/zookeeper
export PATH=$ZOOKEEPER_HOME/bin:$PATH
export JAVA_HOME=/opt/java/openjdk
export PATH=$PATH:$JAVA_HOME:$JAVA_HOME/bin

#配置开机启动
zkServer.sh start
```

刷新配置

```
$ source ~/.bashrc
```

在/etc/profile里检查环境变量配置

```
export JAVA_HOME=/opt/openjdk
export ZOOKEEPER_HOME=/zookeeper

export PATH=$PATH:$JAVA_HOME/bin:$ZOOKEEPER_HOME/bin
```

4.配置zookeeper（在zookeeper1上）

进入到配置目录/conf/，修改下面文件(vscode可能权限不够，用vim)

```
zoo.cfg
```

修改如下内容，指定server的id存储于/data文件夹下：

```
dataDir=/data
```

在zoo.cfg文件末尾添加如下内容：

```
server.1=zookeeper1:2888:3888;2181
server.2=zookeeper2:2888:3888;2181
server.3=zookeeper3:2888:3888;2181
```

注意：配置为主机名称，需要先在/etc/hosts文件添加ip与主机名的映射关系，如下（这个，三个容器都要配置）：

```
172.22.1.9 zookeeper1
172.22.1.10 zookeeper2
172.22.1.11 zookeeper3
```

5.在data目录下修改myid文件(没有则创建)（这个，三个容器都要配置）：

进入data目录

```
$ cd /data
```

修改myid

```
$ vi myid
```

在第1台机器里面填写的内容：

```
1
```

在第2台机器里面填写的内容（等第7步填写）：

```
2
```

在第3台机器里面填写的内容（等第7步填写）：

```
3
```

6.将文件发送到其他节点

```
scp -r /conf root@zookeeper2:/
scp -r /conf root@zookeeper3:/
```

7.在其他容器上修改myid

集群，启动！

启动zookeeper

三台机器都完成以上配置后，在每个容器上分别用如下命令启动zookeeper

```
zkServer.sh start
```

三台都启动好后，分别用如下命令查看zookeeper状态

```
zkServer.sh status
```

启动zookeeper报错

```
ZooKeeper JMX enabled by default
Using config: /conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Error contacting service. It is probably not running.
```

解决：

Zookeeper 启动失败【Cannot open channel to 3 at election address...】_月亮给我抄代码的博客-CSDN博客

```
#下载lsof、iptable、工具
apt-get install lsof ufw net-tools telnet
```

```
lsof -i:2181 //查看某个端口信息
lsof -i //查看所有的监听端口
netstat -nlpt
#查看log
/logs

#在 zoo.cfg 文件中授权监听所有 IP：（找了好久，莫名其妙，这个方法有用）
quorumListenOnAllIPs=true
#之后重启容器，再在每个zookeeper容器分别执行
zkServer.sh start
```

启动journalnode

```
#三个journalnode上做
hadoop-daemon.sh start journalnode
#用下面这个也行，新支持的启动方式，和上面那个二选一，我用的上面那个
hdfs --daemon start namenode
```

```
#报错
-bash: hadoop-daemon.sh: command not found
```

多半是环境变量没配

可在如下文件内容添加变量（注意，既然journalnode没配变量，那说明剩下的hadoop节点也没配，那就都配了吧，不只是journalnode容器）

- ~/.bashrc
- ~/.profile
- /etc/bash.bashrc
- /etc/environment
- /etc/profile

在上面五个文件下都添加hadoop及java环境变量

这里要万万注意第三行，/bin:/usr/bin:/usr/sbin三个路径必须加进环境变量，不然上面配的ssh自启动，以及source命令等，在启动hadoop集群（执行start-all.sh）时会报一堆命令找不到的错误，直接寄寄。

```
export JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64/"
export HADOOP_HOME=/opt/hadoop-3.2.1
export export
PATH=/bin:/usr/bin:/usr/sbin:$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$JAVA_HOME
/bin
```

刷新配置

```
source   ~/.bashrc
source   ~/.profile
source   /etc/profile
source   /etc/environment
source   /etc/bash.bashrc
```

再start一遍，jps测试显示如下

```
root@journalnode1:/# jps
14188 Jps
14126 JournalNode
```

格式化namenode

在主namenode，即namenode1上做

```
hdfs namenode -format
```

格式化ZKFC

```
#执行这句前一定先确保zookeeper集群容器中zookeeper服务已经全部启动
hdfs zkfc -formatZK
```

启动hdfs

```
start-dfs.sh
```

```
#报错
Starting namenodes on [namenode1 namenode2]
ERROR: Attempting to operate on hdfs namenode as root
ERROR: but there is no HDFS_NAMENODE_USER defined. Aborting operation.
Starting datanodes
ERROR: Attempting to operate on hdfs datanode as root
ERROR: but there is no HDFS_DATANODE_USER defined. Aborting operation.
Starting journal nodes [journalnode2 journalnode1 journalnode3]
ERROR: Attempting to operate on hdfs journalnode as root
ERROR: but there is no HDFS_JOURNALNODE_USER defined. Aborting operation.
Starting ZK Failover Controllers on NN hosts [namenode1 namenode2]
ERROR: Attempting to operate on hdfs zkfc as root
ERROR: but there is no HDFS_ZKFC_USER defined. Aborting operation.
```

解决：

解决方案一（这里采用了方案一，注意，所有hadoop容器都要配置）：
输入如下命令，在环境变量中添加下面的配置

```
vi /etc/profile
```

然后向里面加入如下的内容

```
export HDFS_NAMENODE_USER=root
export HDFS_DATANODE_USER=root
export HDFS_SECONDARYNAMENODE_USER=root
export YARN_RESOURCEMANAGER_USER=root
export YARN_NODEMANAGER_USER=root
export HDFS_JOURNALNODE_USER=root
export HDFS_ZKFC_USER=root
```

向root/bashrc中添加下面这句

```
source /etc/profile
```

输入如下命令使改动生效

```
source /etc/profile
```

为了避免后面启动的时候每次都要输入`source /etc/profile`让命令生效，我在所有hadoop容器内的~/.bashrc文件里都添加了如下语句

```
source /etc/profile
```

解决方案二：
将start-dfs.sh，stop-dfs.sh(在hadoop安装目录的sbin里)两个文件顶部添加以下参数

```
HDFS_DATANODE_USER=root
HADOOP_SECURE_DN_USER=hdfs
HDFS_NAMENODE_USER=root
HDFS_SECONDARYNAMENODE_USER=root
```

将start-yarn.sh，stop-yarn.sh(在hadoop安装目录的sbin里)两个文件顶部添加以下参数

```
YARN_RESOURCEMANAGER_USER=root
HADOOP_SECURE_DN_USER=yarn
YARN_NODEMANAGER_USER=root
```

启动standby namenode

即namenode2

上面步骤中主namenode格式化后，会在之前core-site.xml中hadoop.tmp.dir配置的目录下生成hdfs的数据文件，我们要做的就是用命令把这个目录下所有文件拷贝到另一台备用namenode节点所在的机器的相同目录下

因此，登录备用namenode节点运行如下命令，手工启动备用节点，会自动拷贝主namenode文件，这个使用注意运行命令后提示出现"(Y or N) "时一定要选Y让备用节点后续处于交互式随着主namenode启动而启动，注意一定要确保主namenode格式化无异常

```
hdfs namenode -bootstrapStandby
```

启动yarn

要在主resourcemanager所在容器里做，这里把resourcemanager放在了namenode1

```
start-yarn.sh
```

```
##下面这个不用执行了！！！！！
在namenode2上确定是否启动了备份的resourcemanager，如果没起resourcemanager需要手动启，命令如下
yarn-daemon.sh start resourcemanager
```

启动历史服务器

在namenode2上，因为his节点配在了namenode2

```
mr-jobhistory-daemon.sh start historyserver
```

添加外界访问端口

https://zhuanlan.zhihu.com/p/155973403)

格式为 宿主机端口:容器端口

```
#hdfs

#namenode1
9870:9870
"9870/tcp":[{"HostIp":"","HostPort":"9870"}]

#namenode1 为hive做准备，hive用10000端口
10000:10000
"10000/tcp":[{"HostIp":"","HostPort":"10000"}]

#namenode2
9871:9870
"9870/tcp":[{"HostIp":"","HostPort":"9871"}]

#datanode1
9864:9864
"9864/tcp":[{"HostIp":"","HostPort":"9864"}]

#datanode2
9874:9874
"9874/tcp":[{"HostIp":"","HostPort":"9874"}]

#datanode3
9884:9884
"9884/tcp":[{"HostIp":"","HostPort":"9884"}]

#yarn

#namenode1
8091:8088
"8088/tcp":[{"HostIp":"","HostPort":"8091"}]
```

```
#namenode2
8090:8088
"8088/tcp":[{"HostIp":"","HostPort":"8092"}]
```

关闭正在运行要修改的容器

```
docker stop 容器id
#或者用下面命令直接把所有容器全关了
docker stop $(docker ps -a -q)
```

查看要修改的容器id

```
docker ps -a
```

修改容器的端口映射文件（注意要先su到root）

```
cd /var/lib/docker/containers/
ls
找到和containerID一样的那个文件夹cd进去
ls
vi hostconfig.json
```

修改hostconfig.json文件
在PortBindings里面添加 "443/tcp":[{"HostIp":"","HostPort":"443"}]，多个端口就逗号隔开，如

```
 "443/tcp":[{"HostIp":"","HostPort":"443"}], "445/tcp":
[{"HostIp":"","HostPort":"445"}]
```

 查看config.v2.json ExposedPorts里是否有需要放开的端口

```
vi config.v2.json
```

如果没有要把端口加进去，否则做了端口映射外部也没办法访问

```
#namenode的ExposePorts中应有
"9870/tcp":{},"8088/tcp":{}
```

重启docker服务

```
service docker restart
```

使用 `docker ps -a` 查看是否已经修改成功

为了在vscode中修改文件，我对wsl做了如下配置:

打开ubuntu20.04

root用户下

```
vi /etc/ssh/ssh_config

#添加以下内容
Host *
User root
cd /

vi /etc/ssh/ssh_config
把PermitRootLogin改为yes并放开该行注释
```

hadoop访问测试

```
#可以先修改一下访问权限，全打开
hdfs dfs -chmod -R 777 /
```

查看wsl的ip：

```
#wsl中
ifconfig
找eth0下的ip
```

用wsl的ip http访问

```
namenode1 :9870
namenode2 :9871

namenode1 :8091
namenode2 :8092
```

运行内置WordCount例子

把license作为需要统计的文件(在namenode1中)

```
cd /opt/hadoop-3.2.1
cat LICENSE.txt > file1.txt
hadoop fs -mkdir /input
```

这里mkdir的时候一开始报错

```
safemode: Call From namenode1/172.22.1.0 to namenode1:8020 failed on connection
exception: java.net.ConnectException: Connection refused; For more details see:
http://wiki.apache.org/hadoop/ConnectionRefused
```

我寻思，咋会报这个错，后来发现：

hadoop的HDFS中fs.defaultFS在core-site.xml 中配置，默认端口是8020，但是由于其接收Client连接的RPC端口，所以如果在hdfs-site.xml中配置了RPC端口9000，所以fs.defaultFS端口变为9000。因为我的确配了RPC，因此改成9000端口

```
    <name>fs.defaultFS</name>
    <value>hdfs://namenode1:9000</value>
```

但是后面发现还是报错，继续找，发现每个容器的core-site.xml 里都多了很多行重复的

```
<name>fs.defaultFS</name><value>hdfs://容器名:8020</value>
```

使用以下命令测试

```
hdfs getconf -confKey fs.defaultFS
```

输出为

```
hdfs://容器名:8020
```

那确实是这儿的问题，因此把所有容器里多的这些行全部删除，再执行，没问题了。

继续测试

然后发现，又报错了

```
hdfs haadmin -transitionToActive --forcemanual nn2
```

使用下面命令查看namenode1的状态

```
hdfs haadmin -getServiceState  namenode1
```

结果发现namenode1是standby，再测namenode2，是active

下面手动切换活节点到namenode1

```
hdfs haadmin -transitionToActive --forcemanual namenode1
```

发现不行，经过查询得知，当active节点正常时，使用hdfs haadmin -transitionToActive命令对两个namenode节点切换都不起作用，需要手动把active的节点转为standby，原来standby的才能转为active

执行下面语句

```
hdfs haadmin -transitionToStandby -forcemanual namenode2
```

正常

继续测试

```
#输入
hadoop fs -put file1.txt /input

#输出下面信息表示正常
2023-12-07 01:14:10,689 INFO sasl.SaslDataTransferClient: SASL encryption trust
check: localHostTrusted = false, remoteHostTrusted = false
```

查看 HDFS 中 input 文件夹里的内容

```
#输入
hadoop fs -ls /input

#输出下面信息为正常
Found 1 items
-rw-r--r--   3 root supergroup     150569 2023-12-07 01:14 /input/file1.txt
```

运行wordcount 例子程序

```
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.1.jar wordcount
/input/file1.txt /output
```

一直卡在

```
Running job: job_1701907184808_0003
```

我猜是我c盘不足了，不管了，直接生成镜像传上去

后续：清了一下c盘，可以了，然后运行wordcount 例子程序，报错说namenode处于safemode安全模式。

不用担心，这是因为NameNode在启动的时候首先进入安全模式，如果datanode丢失的block达到一定的比例（1-dfs.safemode.threshold.pct），则系统会一直处于安全模式状态即只读状态。dfs.safemode.threshold.pct（缺省值0.999f）表示HDFS启动的时候，如果DataNode上报的block个数达到了元数据记录的block个数的0.999倍才可以离开安全模式，否则一直是这种只读模式。如果设为1则HDFS永远是处于SafeMode。

过一会儿再测，之前那个错没了，但是又报错

```
[2023-12-07 11:57:21.174]Container exited with a non-zero exit code 1. Error
file: prelaunch.err.
Last 4096 bytes of prelaunch.err :
Last 4096 bytes of stderr :
Error: Could not find or load main class
org.apache.hadoop.mapreduce.v2.app.MRAppMaster

Please check whether your etc/hadoop/mapred-site.xml contains the below
configuration:
<property>
  <name>yarn.app.mapreduce.am.env</name>
  <value>HADOOP_MAPRED_HOME=${full path of your hadoop distribution directory}
</value>
</property>
<property>
  <name>mapreduce.map.env</name>
  <value>HADOOP_MAPRED_HOME=${full path of your hadoop distribution directory}
</value>
</property>
<property>
  <name>mapreduce.reduce.env</name>
  <value>HADOOP_MAPRED_HOME=${full path of your hadoop distribution directory}
</value>
```

```
</property>
```

原来，在hadoop3.x后还要在mapred-site.xml和yarn-site.xml中加入以下配置

注意，value的值是namenode1容器中输入`hadoop classpath`命令后得到的输出，注意将英文冒号:改为英文逗号,

```
<property>
    <name>mapreduce.application.classpath</name>
    <value>
        /etc/hadoop:/opt/hadoop-3.2.1/share/hadoop/common/lib/*,
        /opt/hadoop-3.2.1/share/hadoop/common/*,
        /opt/hadoop-3.2.1/share/hadoop/hdfs,
        /opt/hadoop-3.2.1/share/hadoop/hdfs/lib/*,
        /opt/hadoop-3.2.1/share/hadoop/hdfs/*,
        /opt/hadoop-3.2.1/share/hadoop/mapreduce/lib/*,
        /opt/hadoop-3.2.1/share/hadoop/mapreduce/*,
        /opt/hadoop-3.2.1/share/hadoop/yarn,
        /opt/hadoop-3.2.1/share/hadoop/yarn/lib/*,
        /opt/hadoop-3.2.1/share/hadoop/yarn/*
    </value>
</property>
```

在mapred-site.xml中还添加了以下语句

```
<property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
</property>
<property>
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
</property>
<property>
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
</property>
```

配完再运行wordcount 例子程序，又收获了一个错误

```
[2023-12-07 12:31:45.399]Container exited with a non-zero exit code 1. Error
file: prelaunch.err.
Last 4096 bytes of prelaunch.err :
Last 4096 bytes of stderr :
log4j:WARN No appenders could be found for logger
(org.apache.hadoop.mapreduce.v2.app.MRAppMaster).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
info.
For more detailed output, check the application tracking page:
http://namenode2:8088/cluster/app/application_1701953691933_0002 Then click on
links to logs of each attempt.
. Failing the application.
```

使用命令查看日志

```
yarn logs -applicationId application_1701953691933_0002
```

结果发现报了空指针异常

```
2023-12-07 13:15:49,721 ERROR [Listener at 0.0.0.0/41465]
org.apache.hadoop.mapreduce.v2.app.MRAppMaster: Error starting MRAppMaster
org.apache.hadoop.yarn.exceptions.YarnRuntimeException:
java.lang.NullPointerException
        at
org.apache.hadoop.mapreduce.v2.app.rm.RMCommunicator.register(RMCommunicator.jav
a:178)
        at
org.apache.hadoop.mapreduce.v2.app.rm.RMCommunicator.serviceStart(RMCommunicator
.java:122)
        at
org.apache.hadoop.mapreduce.v2.app.rm.RMContainerAllocator.serviceStart(RMContai
nerAllocator.java:280)
        at
org.apache.hadoop.service.AbstractService.start(AbstractService.java:194)
        at
org.apache.hadoop.mapreduce.v2.app.MRAppMaster$ContainerAllocatorRouter.serviceS
tart(MRAppMaster.java:979)
        at
org.apache.hadoop.service.AbstractService.start(AbstractService.java:194)
        at
org.apache.hadoop.service.CompositeService.serviceStart(CompositeService.java:12
1)
        at
org.apache.hadoop.mapreduce.v2.app.MRAppMaster.serviceStart(MRAppMaster.java:129
3)
        at
org.apache.hadoop.service.AbstractService.start(AbstractService.java:194)
        at
org.apache.hadoop.mapreduce.v2.app.MRAppMaster$6.run(MRAppMaster.java:1761)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at
org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1
730)
        at
org.apache.hadoop.mapreduce.v2.app.MRAppMaster.initAndStartAppMaster(MRAppMaster
.java:1757)
        at
org.apache.hadoop.mapreduce.v2.app.MRAppMaster.main(MRAppMaster.java:1691)
Caused by: java.lang.NullPointerException
        at
org.apache.hadoop.mapreduce.v2.app.client.MRClientService.getHttpPort(MRClientSe
rvice.java:177)
        at
org.apache.hadoop.mapreduce.v2.app.rm.RMCommunicator.register(RMCommunicator.jav
a:159)
        ... 14 more
```

然后我发现，我配yarn web UI服务的地址的时候写的是

```
yarn.resourcemanager.web.address.rm1
yarn.resourcemanager.web.address.rm2
```

网上不少教程写的是webapp不是web，于是进行修改

```
yarn.resourcemanager.webapp.address.rm1
yarn.resourcemanager.webapp.address.rm2
```

配完后再测试运行又卡死在

```
2023-12-07 14:13:33,591 INFO mapreduce.Job: Running job: job_1701955768500_0003
```

难道是...内存又不足了?

在yarn-site.yml添加以下内容，允许多申请点空间

```
    <property>
            <name>yarn.nodemanager.resource.memory-mb</name>
            <value>4096</value>
    </property>
```

之后重启服务

```
stop-all.sh
start-all.sh
```

运行wordcount 例子程序

如果/output文件夹已存在会报错，此时要先删除/output文件夹

```
hadoop fs -rm -r /output
```

```
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.1.jar wordcount
/input/file1.txt /output
```

成功!

如果运行成功显示应类似下面的

查看 HDFS 中的 /output 文件夹的内容

```
hadoop fs -ls /output
Found 2 items
-rw-r--r--   2 root supergroup          0 2020-09-14 11:09 /output/_SUCCESS
-rw-r--r--   2 root supergroup      35324 2020-09-14 11:09 /output/part-r-00000
```

查看`part-r-00000`文件的内容，就是运行的结果

```
 hadoop fs -cat /output/part-r-00000
```

搭建hive

将hive解压后的文件夹上传到namenode1、namenode2

hive版本3.1.3

```
docker cp "/mnt/d/陈嘉瑞的文件夹/同济大学/课程文件、文稿/大三/云计算技术/hive"
namenode1:/usr/local/
```

修改配置文件（在namenode1:/usr/local/hive/conf中）

```
cd /usr/local/hive/conf
cp hive-default.xml.template hive-site.xml
vim hive-site.xml

#添加以下内容
<property>
    <name>system:java.io.tmpdir</name>
    <value>/tmp/hive/java</value>
</property>
<property>
    <name>system:user.name</name>
    <value>${user.name}</value>
</property>
```

将hive发送到namenode2

```
scp -r /usr/local/hive root@namenode2:/usr/local
```

配置hive相关环境变量(namenode1 namenode2都要配)

```
vim /etc/profile

#文本最后添加
export HIVE_HOME="/usr/local/hive"
export PATH=$PATH:$HIVE_HOME/bin
```

配置后执行source /etc/profile 生效

```
source /etc/profile
```

注意，还要在两个namenode上放开10000端口（hive访问默认端口），以及到宿主机的端口映射（namenode1:10000 namenode2:10001），然后重启容器，已经加了就行

配置mysql

```
#拉取镜像
docker pull mysql:8.0.18
#建立容器
docker run --name mysql_hive -p 4306:3306 --net cloud_computing_network --ip
172.22.1.12 -v /root/mysql:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456 -d
mysql:8.0.18
#进入容器
docker exec -it mysql_hive bash
#进入mysql
mysql -uroot -p
#密码上面建立容器时候已经设置123456
#建立hive数据库
create database hive;
#修改远程连接权限
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY '123456';
```

回去namenode1容器，修改关联数据库的配置

```
docker exec -it namenode1 /bin/bash
vim /usr/local/hive/conf/hive-site.xml
```

搜索关键词修改数据库url、驱动、用户名，url根据上面建容器时候地址

注意驱动别写com.mysql.jdbc.Driver，这是mysql-connector-java-5.x的，这里用的是8.2.0，正确写法是com.mysql.cj.jdbc.Driver

```
#还请注意hive配置文件里面使用&amp;作为分隔，高版本myssql需要SSL验证，在这里设置关闭
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>123456</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://172.22.1.12:3306/hive?
createDatabaseIfNotExist=true&amp;useSSL=false</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.cj.jdbc.Driver</value>
</property>
<property>
    <name>hive.metastore.schema.verification</name>
    <value>false</value>
    <property>
```

mysql驱动从windows上传到hive的lib下，只传.jar文件

```
exit
docker cp "/mnt/d/陈嘉瑞的文件夹/同济大学/课程文件、文稿/大三/云计算技术/mysql-connector-j-8.2.0/mysql-connector-j-8.2.0.jar" namenode1:/usr/local/hive/lib
```

jar包配置修改

对hive的lib文件夹下的部分文件做修改，不然初始化数据库的时候会**报错**

```
#进入namenode1
#slf4j这个包hadoop及hive两边只能有一个，这里删掉hive这边
cd /usr/local/hive/lib
rm log4j-slf4j-impl-2.17.1.jar

#guava这个包hadoop及hive两边只删掉版本低的那个，把版本高的复制过去，这里删掉hive，复制hadoop的过去
#因为hadoop里是guava-27.0-jre.jar，hive里是guava-19.0-jre.jar
cp /opt/hadoop-3.2.1/share/hadoop/common/lib/guava-27.0-jre.jar /usr/local/hive/lib
rm /usr/local/hive/lib/guava-19.0.jar

#把文件hive-site.xml第3223行的特殊字符删除&#8（不在这一行也上下差不了几行，删了就行）
vim /usr/local/hive/conf/hive-site.xml
```

初始化元数据库

```
cd /usr/local/hive/bin
schematool -initSchema -dbType mysql
```

成功后提示:

```
root@namenode1:/usr/local/hive/bin# schematool -initSchema -dbType mysql
Metastore connection URL:        jdbc:mysql://172.22.1.12:3306/hive?createDatabaseIfNotExist=true&useSSL=false
Metastore Connection Driver :    com.mysql.cj.jdbc.Driver
Metastore connection User:       root
Starting metastore schema initialization to 3.1.0
Initialization script hive-schema-3.1.0.mysql.sql
Initialization script completed
schemaTool completed
```

验证hive+mysql

验证配置是否成功（namenode1上）

我们先创建一个数据文件放到`/usr/local`下

```
cd /usr/local
vim test.txt

#加入以下几行
1,jack
2,hel
3,nack
```

进入hive交互界面进行测试

```
root@namenode1:/opt/hadoop-3.2.1# hive
Hive Session ID = a631a76d-12cc-42c2-b9ac-0454803ac3d3

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-
common-3.1.3.jar!/hive-log4j2.properties Async: true
Hive Session ID = b8cc922b-e54f-449d-8f83-6a28a1405606
Hive-on-MR is deprecated in Hive 2 and may not be available in the future
versions. Consider using a different execution engine (i.e. spark, tez) or using
Hive 1.X releases.
hive> create table customer(
    >     id       int
    > ,name     string
    > )
    > row format delimited
    > fields terminated by ',';
OK
Time taken: 0.777 seconds
hive> load data local inpath '/usr/local/test.txt' into table customer;
Loading data to table default.customer
OK
Time taken: 0.405 seconds
hive> select * from customer;
OK
1       jack
2       hel
3       nack
Time taken: 1.173 seconds, Fetched: 3 row(s)
```

启动Hiveserver2

修改hadoop的一些权限配置（所有hadoop上都要配）

```
vim /opt/hadoop-3.2.1/etc/hadoop/core-site.xml
```

加入以下配置

```
<property>
    <name>hadoop.proxyuser.root.hosts</name>
    <value>*</value>
</property>
<property>
    <name>hadoop.proxyuser.root.groups</name>
    <value>*</value>
</property>
```

重启hadoop服务

```
stop-all.sh
start-all.sh
```

后台启动hiveserver2

```
#注意，hive --service hiveserver2 命令将在前台启动hiveserver2导致窗口不能进行其他操作所以要后台启动
nohup hiveserver2 >/dev/null 2>/dev/null &

#输出，不一定和下面一样
[1] 25393
#其中[1]表示进程的标识号（job number），25393是该进程的PID（进程ID）
```

验证

通过beeline连接并查询

查看10000端口运行正常，beeline命令，!connect连接，结果查询正常

```
root@namenode1:/opt/hadoop-3.2.1#  netstat -ntulp |grep 10000
tcp        0      0 0.0.0.0:10000           0.0.0.0:*               LISTEN      25393/java
root@namenode1:/opt/hadoop-3.2.1# beeline
Beeline version 3.1.3 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000/default
Connecting to jdbc:hive2://localhost:10000/default
Enter username for jdbc:hive2://localhost:10000/default: root
Enter password for jdbc:hive2://localhost:10000/default: ******（这里写mysql密码123456）
Connected to: Apache Hive (version 3.1.3)
Driver: Hive JDBC (version 3.1.3)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000/default> select * from customer;
INFO  : Compiling command(queryId=root_20231207162208_393ff362-a45e-4ba4-9945-d28cb325c687): select * from test
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:test.id, type:int, comment:null), FieldSchema(name:test.name, type:string, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=root_20231207162208_393ff362-a45e-4ba4-9945-d28cb325c687); Time taken: 1.391 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20231207162208_393ff362-a45e-4ba4-9945-d28cb325c687): select * from test
INFO  : Completed executing command(queryId=root_20231207162208_393ff362-a45e-4ba4-9945-d28cb325c687); Time taken: 0.002 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+----------+------------+
| test.id  | test.name  |
+----------+------------+
| 1        | jack       |
| 2        | hel        |
| 3        | nack       |
+----------+------------+
3 rows selected (1.722 seconds)
0: jdbc:hive2://localhost:10000/default>
```

将容器打包为镜像

首先在wsl2上登录docker

```
docker login
```

发现报错

```
Error saving credentials: error storing credentials - err: exec: "docker-credential-desktop.exe": executable file not found in $PATH, out:
```

解决：先把下面这个文件删了，再登录

```
rm -rf ~/.docker/config.json
```

提交镜像

```
#容器打包为镜像
docker commit containerId或名字 imagesId或名字:版本

#镜像打tag
docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]

#镜像上传docker hub
docker push[dockerHub push地址]:[dockerHub上的命名]
```

注意提交前要将所提交的容器先停止运行

```
// docker commit 容器名 本地镜像名:版本号
// docker tag 本地镜像名:版本号 /用户名/仓库名:版本号
// docker push /仓库名/镜像名:版本号
//没有版本号默认latest

docker commit namenode1 namenode1:v1
docker tag namenode1:v1 singularitycjr/2023_cloudcomputing_namenode1:v1
docker push singularitycjr/2023_cloudcomputing_namenode1:v1
docker commit namenode2 namenode2:v1
docker tag namenode2:v1 singularitycjr/2023_cloudcomputing_namenode2:v1
docker push singularitycjr/2023_cloudcomputing_namenode2:v1

docker commit datanode1 datanode1:v1
docker tag datanode1:v1 singularitycjr/2023_cloudcomputing_datanode1:v1
docker push singularitycjr/2023_cloudcomputing_datanode1:v1

docker commit datanode2 datanode2:v1
docker tag datanode2:v1 singularitycjr/2023_cloudcomputing_datanode2:v1
docker push singularitycjr/2023_cloudcomputing_datanode2:v1

docker commit datanode3 datanode3:v1
docker tag datanode3:v1 singularitycjr/2023_cloudcomputing_datanode3:v1
docker push singularitycjr/2023_cloudcomputing_datanode3:v1

docker commit journalnode1 journalnode1:v1
docker tag journalnode1:v1 singularitycjr/2023_cloudcomputing_journalnode1:v1
docker push singularitycjr/2023_cloudcomputing_journalnode1:v1

docker commit journalnode2 journalnode2:v1
docker tag journalnode2:v1 singularitycjr/2023_cloudcomputing_journalnode2:v1
docker push singularitycjr/2023_cloudcomputing_journalnode2:v1

docker commit journalnode3 journalnode3:v1
docker tag journalnode3:v1 singularitycjr/2023_cloudcomputing_journalnode3:v1
docker push singularitycjr/2023_cloudcomputing_journalnode3:v1

docker commit zookeeper1 zookeeper1:v1
docker tag zookeeper1:v1 singularitycjr/2023_cloudcomputing_zookeeper1:v1
docker push singularitycjr/2023_cloudcomputing_zookeeper1:v1

docker commit zookeeper2 zookeeper2:v1
docker tag zookeeper2:v1 singularitycjr/2023_cloudcomputing_zookeeper2:v1
docker push singularitycjr/2023_cloudcomputing_zookeeper2:v1

docker commit zookeeper3 zookeeper3:v1
docker tag zookeeper3:v1 singularitycjr/2023_cloudcomputing_zookeeper3:v1
docker push singularitycjr/2023_cloudcomputing_zookeeper3:v1

docker commit mysql_hive mysql_hive:v1
docker tag mysql_hive:v1 singularitycjr/2023_cloudcomputing_mysql:v1
docker push singularitycjr/2023_cloudcomputing_mysql:v1
```

```
#v2版本

docker commit namenode1 namenode1:v2
docker tag namenode1:v2 singularitycjr/2023_cloudcomputing_namenode1:v2
docker push singularitycjr/2023_cloudcomputing_namenode1:v2

docker commit namenode2 namenode2:v2
docker tag namenode2:v2 singularitycjr/2023_cloudcomputing_namenode2:v2
docker push singularitycjr/2023_cloudcomputing_namenode2:v2
docker commit datanode1 datanode1:v2
docker tag datanode1:v2 singularitycjr/2023_cloudcomputing_datanode1:v2
docker push singularitycjr/2023_cloudcomputing_datanode1:v2

docker commit datanode2 datanode2:v2
docker tag datanode2:v2 singularitycjr/2023_cloudcomputing_datanode2:v2
docker push singularitycjr/2023_cloudcomputing_datanode2:v2

docker commit datanode3 datanode3:v2
docker tag datanode3:v2 singularitycjr/2023_cloudcomputing_datanode3:v2
docker push singularitycjr/2023_cloudcomputing_datanode3:v2

docker commit journalnode1 journalnode1:v2
docker tag journalnode1:v2 singularitycjr/2023_cloudcomputing_journalnode1:v2
docker push singularitycjr/2023_cloudcomputing_journalnode1:v2

docker commit journalnode2 journalnode2:v2
docker tag journalnode2:v2 singularitycjr/2023_cloudcomputing_journalnode2:v2
docker push singularitycjr/2023_cloudcomputing_journalnode2:v2

docker commit journalnode3 journalnode3:v2
docker tag journalnode3:v2 singularitycjr/2023_cloudcomputing_journalnode3:v2
docker push singularitycjr/2023_cloudcomputing_journalnode3:v2

docker commit zookeeper1 zookeeper1:v2
docker tag zookeeper1:v2 singularitycjr/2023_cloudcomputing_zookeeper1:v2
docker push singularitycjr/2023_cloudcomputing_zookeeper1:v2

docker commit zookeeper2 zookeeper2:v2
docker tag zookeeper2:v2 singularitycjr/2023_cloudcomputing_zookeeper2:v2
docker push singularitycjr/2023_cloudcomputing_zookeeper2:v2

docker commit zookeeper3 zookeeper3:v2
docker tag zookeeper3:v2 singularitycjr/2023_cloudcomputing_zookeeper3:v2
docker push singularitycjr/2023_cloudcomputing_zookeeper3:v2

docker commit mysql_hive mysql_hive:v2
docker tag mysql_hive:v2 singularitycjr/2023_cloudcomputing_mysql:v2
docker push singularitycjr/2023_cloudcomputing_mysql:v2
```

配置好后启动容器

```
docker restart namenode1 namenode2 datanode1 datanode2 datanode3 journalnode1
journalnode2 journalnode3 zookeeper1 zookeeper2 zookeeper3 mysql_hive

docker exec -it namenode1 /bin/bash

start-all.sh

ssh namenode2

mr-jobhistory-daemon.sh start historyserver

#注意如果两个namenode jps后都没有zkfc，还要执行下面这句
```

```
hadoop-daemon.sh   start  zkfc

nohup hiveserver2 >/dev/null 2>/dev/null &

#其他，下面这个不用执行，测试用
hadoop-daemon.sh   stop zkfc
```

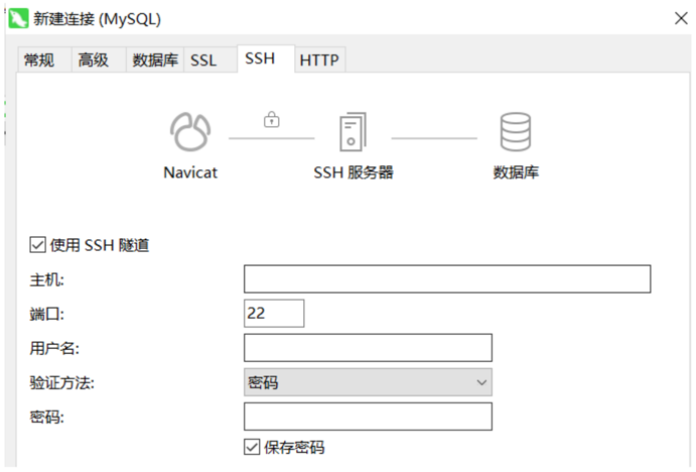
外界访问hive

查看hive元数据

这是在mysql数据库上

```
#WSL宿主机上

#ssh_config是配客户端的，sshd_config是配服务端的
sudo vi /etc/ssh/sshd_config

#下面这几句改成yes，之前是no
PermitRootLogin yes
PasswordAuthentication yes


#重启ssh服务
sudo systemctl enable ssh.service
```

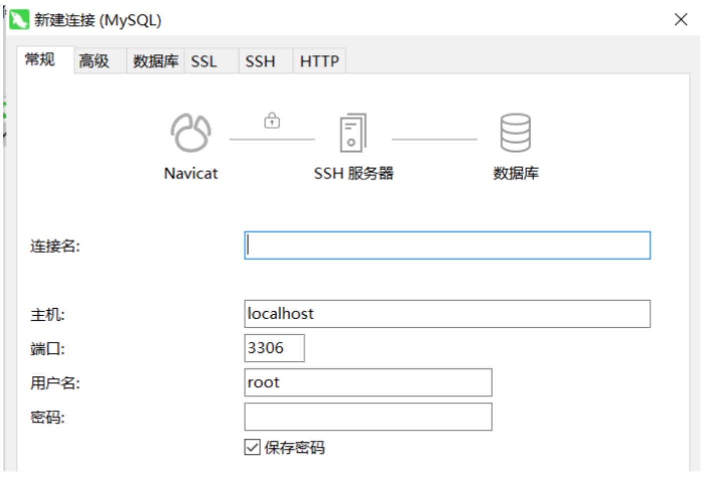
以Navicat为例：


新建连接 (MySQL)

☑ 使用 SSH 隧道

主机:	
端口:	22
用户名:	
验证方法:	密码
密码:	
	☑ 保存密码

这里填wsl地址（wsl里查eth0网卡)，端口默认22 用户名密码就填wsl的，我填的是用户名root密码root



连接名:

主机: localhost

端口: 3306

用户名: root

密码:

☑ 保存密码

这里填连接名、mysql的用户名和密码

我的wsl主机ip是172.31.2.42

查看hive数据库表内容

在idea中，name随便填，host填wsl的ip，user填hive，密码不填

hive默认user是hive，密码是无，端口10000（看hive的10000具体映射到了wsl哪个端口）

建表

```
#注意结束语句位置加分号
#注意，指定的location是文件夹，不是文件，因此指定后会在saledata下创建五个文件夹。要将数据csv

CREATE DATABASE IF NOT EXISTS SaleSystem;

-- 创建外部表 Customer，指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SaleSystem.customer (
    customerID  INT,
    customerName VARCHAR(255),
    Phone VARCHAR(255),
    email VARCHAR(255),
    address VARCHAR(255)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/saledata/customers'
tblproperties ("skip.header.line.count"="1");

-- 创建外部表 Product，指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SaleSystem.product (
    productId INT,
    productName VARCHAR(255),
    unitPrice  DECIMAL(10, 2)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/saledata/products'
tblproperties ("skip.header.line.count"="1");
```

```
-- 创建外部表 Orders，指定数据存储在 HDFS 上的路径
#order是关键字，这里用orders
CREATE EXTERNAL TABLE IF NOT EXISTS SaleSystem.orders (
    orderId INT,
    customerId INT,
    orderDate  DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/saledata/orders'
tblproperties ("skip.header.line.count"="1");

-- 创建外部表 OrderDetails，指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SaleSystem.orderDetail (
    orderDetailID INT,
    orderId INT,
    productId INT,
    quantity INT,
    unitPrice  DECIMAL(10, 2)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/saledata/orderDetails'
tblproperties ("skip.header.line.count"="1");

-- 创建外部表 Payment，指定数据存储在 HDFS 上的路径
CREATE EXTERNAL TABLE IF NOT EXISTS SaleSystem.payment (
    paymentID  INT,
    orderId INT,
    amount  DECIMAL(10, 2),
    paymentMethod  VARCHAR(255),
    paymentDate   DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/saledata/payments'
tblproperties ("skip.header.line.count"="1");
```

```
#其他hivesql语句

#查看所有表
use SaleSystem;
show tables;

#查看所有数据库
show databases;

#仅删除表中数据，保留表结构
truncate table table_name;

#删除表
```

```
drop table if exists table_name;

#删除库
drop database if exists database_name;
```

注意

外网访问hadoop

WebHDFS提供了访问HDFS的RESTful接口，是内置组件，并且默认开启，运行于NameNode和DataNode中，对HDFS文件的读写，将会重定向到文件所在的DataNode，并且会完全利用HDFS的带宽。； WebHDFS访问时，首先访问NameNode获知文件所在的DataNode地址，然后重定向到目标DataNode获取文件内容；

HDFS 中的文件夹和文件名都是存放在 NameNode 上，操作不需要和 DataNode 通信，因此可以正常创建文件夹和创建文件说明本地和远程 NameNode 通信没有问题。

文件夹和文件名都是存放在 NameNode 上的，我本地可以通过公网访问 NameNode，所以创建文件夹和文件都可以，但是当我写数据的时候，NameNode 和DataNode 是通过内网通信的，NameNode 会返回给我 DataNode 的内网 IP，我本地就访问不了了。

当hadoop集群使用内网搭建，使用外网访问hadoop，上传或下载文件是没法直接访问datanode。

因此要配置以hostname方式访问namenode

```
#在hdfs-site.xml中添加

<configuration>
    <property>
        <name>dfs.client.use.datanode.hostname</name>
        <value>true</value>
    </property>
    <property>
        <name>dfs.datanode.use.datanode.hostname</name>
        <value>true</value>
    </property>
</configuration>
```

```
在windows的C:\Windows\System32\drivers\etc\hosts文件中添加
172.31.2.42 datanode1
172.31.2.42 datanode2
172.31.2.42 datanode3
#注意，三个datanode都映射到wsl的地址
```

注意，在三个datanode中还要加入datanode的http访问端口，下面三个配置不能每个datanode都放，是datanode几，就加value位置对应的那个，目的是指定该datanode的http通信端口。只在三个datanode上加！！

```
<property>
    <name>dfs.datanode.http.address</name>
    <value>datanode1:9864</value>
</property>
```

```
<property>
    <name>dfs.datanode.http.address</name>
    <value>datanode2:9874</value>
</property>

<property>
    <name>dfs.datanode.http.address</name>
    <value>datanode3:9884</value>
</property>
```

我一开始datanode2和3分别用的9865和9866，后来发现，这两个端口已经被hadoop别的默认服务占掉了不能用，所以改换9874和9884

配置文件

无论何时，保证hadoop的hadoop-3.2.1文件夹内各节点配置文件一致

如果entrypoint.sh或其他配置文件错误导致容器无法进入，可以将配置文件先拷贝到wsl宿主机，修改后再放回容器中

以entrypoint.sh为例

```
sudo docker cp journalnode3:/entrypoint.sh /entrypoint
sudo cd entrypoint
sudo vi entrypoint.sh
sudo docker cp /entrypoint/entrypoint.sh journalnode3:/
```

注意配置文件里变量和值是否一致，比如下面这个例子，第二个property里，应该是rm2对应namenode2，配置错误导致namenode1一直无法启动yarn。我真的是...我服了

```
<!-- yarn对nodemanager服务的地址 -->
<property>
    <name>yarn.resourcemanager.resource-tracker.address.rm1</name>
    <value>namenode1:8031</value>
</property>

<property>
    <name>yarn.resourcemanager.resource-tracker.address.rm1</name>
    <value>namenode2:8031</value>
</property>
```

用户组

在开始所有配置之前（最好是安装完docker就做）记得将要操作docker的linux用户加入用户组，然后重启docker服务

```
 sudo gpasswd -a cjr docker
 sudo service docker restart
```

fs.defaultFS与fs.default.name

使用 fs.default.name 还是 使用 fs.defaultFS， 要首先判断是否开启了 NN 的HA (namenode 的highavaliable)，如果开启了NN HA，那么就用fs.defaultFS，在单一namenode的情况下，就用fs.default.name

namenode和sourcemanager的active检测

如，检测namenode1上的namenode是否为active

```
hdfs haadmin -getServiceState  namenode1
```

如，检测namenode1上的sourcemanager是否为active

```
yarn rmadmin -getServiceState rm1
```

将namenode2的namenode切为standby

```
hdfs haadmin -transitionToStandby -forcemanual namenode2
```

将namenode2的sourcemanager切为standby（在yarn-site.xml文件中，namenode2对应rm2）

```
yarn rmadmin -transitionToStandby -forcemanual rm2
```

```
yarn rmadmin -transitionToActive -forcemanual rm1
```

```
yarn --daemon start resourcemanager
yarn --daemon stop resourcemanager
```

```
hdfs  --daemon start namenode
hdfs  --daemon stop namenode
```

切换用下面两段

```
yarn --daemon start resourcemanager
yarn --daemon stop resourcemanager
```

```
hdfs haadmin -transitionToStandby -forcemanual namenode1
hdfs haadmin -transitionToStandby -forcemanual namenode2
```

宕掉namenode后监测时间

```
timeout = 2 * heartbeat.recheck-interval + 10 * dfs.heartbeat.interval
```

vim中常用快捷键

/查找内容 从光标所在行向下查找

?查找内容 从光标所在行向上查找

:行号 按行寻找

:wq 保存文件

[Linux 之 Vim 命令使用（详细总结） - 知乎 (zhihu.com)](

密码

我的容器密码和ubuntu的用户密码都和用户名、容器名相同，好记

#!/usr/bin/env bash

这个命令不完全是注释，它实际上指定了bash的目录，表示使用/usr/bin/env文件，去动态查找bash的目录

然后系统就会去~/.bashrc里配的环境变量里，从环境变量路径找bash，这就是为什么我们前面要在~/.bashrc里加/bin:/usr/bin:/usr/sbin，service命令对应文件在/usr/sbin里

重新格式化

重新格式化hadoop

删除namenode和datanode中tmp文件夹下的所有内容

删除journalnode上存储数据路径下所有文件夹和文件

```
#namenode
rm -r /hadoop/dfs/name/*
```

```
#datanode
rm -r /hadoop/dfs/data/*
rm -r  /hadoop/nm-local-dir
```

```
#journalnode
rm -r  /home/journaldata/nameservice/*
```

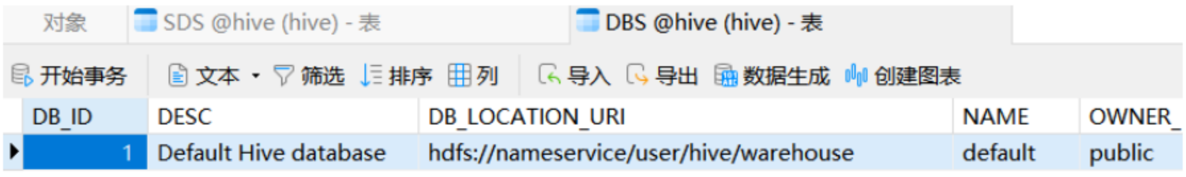
重新格式化hive

```
#namenode1中
hadoop fs -rmr /user/
```

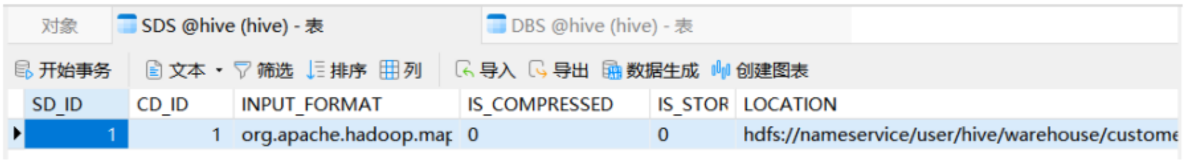
```
#mysql_hive中
drop database hive;
```

然后再重走一遍格式化hive

注意，如果使用高可用HA则zookeeper存储的元数据中DBS和SDS两张表的hds路径应该是集群名（我这里是nameservice），而不是如hdfs://namenode1:9000/...的形式



DB_ID	DESC	DB_LOCATION_URI	NAME	OWNER_
1	Default Hive database	hdfs://nameservice/user/hive/warehouse	default	public



SD_ID	CD_ID	INPUT_FORMAT	IS_COMPRESSED	IS_STOR	LOCATION
1	1	org.apache.hadoop.map	0	0	hdfs://nameservice/user/hive/warehouse/custome

其他命令

```
#hive删库
drop database if exists database_name;

#hive强制删库
drop database if exists database_name cascade;
如：drop database if exists SaleSystem cascade;

#查看是否处于安全模式
hdfs dfsadmin -safemode get

#hadoop离开安全模式
hadoop dfsadmin -safemode leave
```

参考

原生Hadoop配置高可用：

https://blog.csdn.net/dudadudadd/article/details/109858613

基于Docker搭建Hadoop+Hive - 知乎

基于Docker搭建Hadoop+Hive - 知乎 (zhihu.com)

大数据Hadoop之——Hadoop 3.3.4 HA（高可用）原理与实现（QJM） - 掘金 (juejin.cn)