

安装拓扑结构图：

注意：

Zookeeper-myid 的设置与 zoo.cfg 有关

Hadoop-slaves 的设置与/etc/hosts 有关

通用环境准备：

java 编译环境准备：

JAVA 安装：

Java-1.8.0_161

添加 JAVA 环境变量

防火墙关闭

结点时间同步准备：

软件安装

所有服务器安装软件：

```
yum -y install ntp
```

服务器端（192.168.6.210）配置：

客户端（192.168.0.102）配置

客户端同步时间

```
ntpdate -u 192.168.0.101
```

补充说明：

不同机器在安装系统时，可能存在时区选择不一致的问题，常见的如 CST EDT 等，此时需要将时区选择一致。

这里统一设置为 CST

```
sudo mv /etc/localtime /etc/localtime.bak
```

```
sudo ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

ssh 无密码访问准备：

为普通结点创建管理账户

以下操作需要通过 ssh 登录相应主机后，在结点上执行，默认 username 和 passwd 均为 hadoop，在下面的参数中也使用 hadoop

新建账户：

```
sudo useradd -d /home/{username} -m {username}
```

为账户创建密码：

```
sudo passwd {username}
```

给账户提升 root 权限：

```
echo "{username} ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/{username}
```

```
sudo chmod 0440 /etc/sudoers.d/{username}
```

为 namenode 结点创建方便的管理方法

注：以下操作都是针对 name-node 结点，hadoop 用户。

生成 ssh 公私钥对：

```
ssh-keygen
```

注：使用普通用户 hadoop，所有密码位空，直接回车就好

将生成的 key 复制到普通结点：

```
ssh-copy-id hadoop@node0
```

```
ssh-copy-id hadoop@node1
```

```
ssh-copy-id hadoop@node2
```

首次需要输入正确的密码

注：若是无法执行成功可以先执行下面的步奏，与 node0, node1,node2 无法解析为 IP 有关，在/etc/hosts 中添加即可

生成新的 ssh 配置文件：

```
vi ~/.ssh/config(格式如下：)
```

```
Host node1
```

```
    Hostname node1
```

```
    User hadoop
```

```
Host node2
```

```
    Hostname node2
```

```
    User hadoop
```

```
Host node0
```

```
    Hostname node0
```

```
    User hadoop
```

注：这里假设有 3 个用户

环境变量准备：

```
export JAVA_HOME=/lovelsl/java/jdk
export JRE_HOME=/lovelsl/java/jdk/jre
export HADOOP_HOME=/lovelsl/hadoop/hadoop
export HBASE_HOME=/lovelsl/hbase/hbase
export HIVE_HOME=/lovelsl/hive/hive
export KYLIN_HOME=/lovelsl/kylin/kylin
export SPARK_HOME=/lovelsl/spark/spark
export SQOOP_HOME=/lovelsl/sqoop/sqoop

export HADOOP_CLASSPATH=$HADOOP_HOME/lib/*
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HIVE_HOME/lib/*:$HIVE_HOME/conf
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin:$HADOOP_HOME/bin:$HBASE_HOME/bin:$HIVE_HOME/bin:$KYLIN_HOME/bin:$SPARK_HOME/bin:$SQOOP_HOME/bin
```

HADOOP（HA）完全分布式

hadoop 账户配置

ip	192.168.6.210	192.168.6.211	192.168.6.222
hostname	node0	node1	node2
user	hadoop	hadoop	hadoop
ssh	√	√	√
ntp	Server	Client	Client
zookeeper—myid	0	1	2

hadoop-slaves	node0 node1 node2	node0 node1 node2	node0 node1 node2
namenode	√ (zk 选举)		√ (zk 选举)
datanode	√	√	√
resourcemanager	√ (zk 选举)		√ (zk 选举)
nodemanager	√	√	√

注意：

Zookeeper-myid 的设置与 zoo.cfg 有关

Hadoop-slaves 的设置与/etc/hosts 有关

hadoop 配置文件解析

HDFS 的守护进程包括：NameNode，SecondaryNamenode，DataNode

YARN 的守护进程：ResourceManager，NodeManager，WebAppProxy

MapReduce 的相关守护进程：Map Reduce Job History Server

配置 hadoop 相关守护进程的 java 运行参数

此处的设置，主要针对 hadoop 安装包下的 etc/hadoop/hadoop-env.sh，etc/hadoop/mapred-env.sh，etc/hadoop/yarn-env.sh 三处相关。

- JAVA 环境变量设置
- 通过这些文件，可以配置 hadoop 运行时的运行环境参数，如调整 java 的 gc，堆等，设置方法，参考下表

Daemon	环境变量
NameNode	HADOOP_NAMENODE_OPTS
DataNode	HADOOP_DATANODE_OPTS
Secondary Namenode	HADOOP_SECONDARYNAMENODE_OPTS
ResourceManager	YARN_RESOURCEMANAGER_OPTS
NodeManager	YARN_NODEMANAGER_OPTS
WebAppProxy	YARN_PROXYSERVER_OPTS
Map Reduce Job History Server	HADOOP_JOB_HISTORYSERVER_OPTS

更具体的设置，参考 hadoop 官方文档，这里不做细致描述

链接地址：

http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html#Monitoring_Health_of_NodeManagers

配置 hadoop 守护进程参数

hadoop 的配置文件存在默认参数，很多参数在不必要的时候，可以默认不用设置值。
注意以下，集群名字，会与 hadoop 集群名，hdfs 集群名混用，大抵意思相同

core-site.xml

- fs.defaultFS
namenode 的 uri, 格式为 hdfs://host:port/
对于单机分布式, 伪分布式, 或者单个 namenode 情况下, 可以直接使用 hdfs://ip:9000/ 的形式, 或者干脆 hdfs://localhost:9000/ (会存在页面跳转的问题)。在完全分布式的情况下格式为 hdfs://clustername/, 此时直接写集群的名字, 地址交由 zookeeper 来选举决定, 也不能输入端口, 端口是在 namenode 中指定的。
- io.file.buffer.size
默认值 131072, 为读写 sequenceFiles 的缓冲区大小。可使用默认值
- hadoop.tmp.dir
指定 hadoop 的数据临时存储目录
- hadoop.proxyuser.hduser.hosts
指定可以访问 hadoop 集群的 ip, 默认为*, 即所有用户
- hadoop.proxyuser.hduser.groups
指定可以访问 hadoop 集群的用户, 默认为*, 即所有用户
- ha.zookeeper.quorum
指定 zookeeper 的 ip 地址, 用于从多个 namenode 中选举主 namenode 和备用 namenode
完全分布式必须使用

hdfs-site.xml

注意 1, 由于设置项中会使用到集群名, 这里直接使用 mycluster 取代, 实际使用时注意将该值改为 dfs.nameservices 项指定的值。

注意 2, 假定 mycluster 中存在 nn1, nn2 两个 namenode 结点

注意 3, 假定 mycluster 中存在 node1, node2, node3 三个 datanode

- dfs.hosts.exclude
结点黑名单列表文件, 可以用于下线 hadoop 结点
dfs.hosts 表示允许哪些可以连接 hdfs
- dfs.blocksize
指定 hdfs 的数据块(block)大小, 默认 64M
- dfs.nameservices
指定 hdfs 的集群名字, 需要与 core-site.xml 中一致
- dfs.namenode.handler.count
namenode 与 datanode 的 rpc 通讯线程数目, 默认为 100

- `dfs.ha.namenodes.mycluster`
指定 hadoop 集群中的 namenode 结点
- `dfs.namenode.rpc-address.mycluster.nn1`
指定 namenode 的 rpc 通信地址和端口，如 `nn1:9000`，nn1 必须提前做好 hosts
所有 namenode 都必须设置该项
完全分布式必须使用
- `dfs.namenode.http-address.mycluster.nn1`
指定 namenode 的 http 通讯地址和端口，如 `nn1:50070`，nn1 必须提前做好 hosts
所有 namenode 都必须设置该项
完全分布式必须使用
- `dfs.namenode.shared.edits.dir`
指定 namenode 的元数据，在哪些主机上存放，如
`qjournal://node1:8485;node2:8485;node3:8485/mycluster`
- `dfs.journalnode.edits.dir`
指定 namenode 的元数据在 journalnode 上的存放位置
- `dfs.client.failover.proxy.provider.mycluster`
配置失败的自动启动方式
- `dfs.ha.fencing.methods`
配置隔离方式
- `dfs.ha.fencing.ssh.private-key-files`
配置隔离方式时需要使用的 ssh 免密码登录
- `dfs.datanode.data.dir`
指定 datanode 数据存放地址
- `dfs.replication`
指定数据冗余份数，默认值为 3
- `dfs.webhdfs.enabled`
指定可以通过 web 访问 hdfs 目录，值为 `true`
- `dfs.ha.automatic-failover.enabled`
支持 HA 的 namenode 自动切换功能，值为 `true`
HA 必须配置项
- `dfs.journalnode.http-address`
- `dfs.journalnode.rpc-address`

mapred-site.xml

- `mapreduce.framework.name`
配置 mapreduce 运行于 yarn 中
- `mapreduce.jobhistory.address`
配置 mapreduce jobhistory server 地址，默认端口 10020
- `mapreduce.jobhistory.webapp.address`
配置 mapreduce jobhistory server web ui 地址，默认端口 19888

yarn-site.xml

以下做配置介绍和说明时，假定配置了逻辑名 rm1 和 rm2 的 resource manager

- yarn.log-aggregation-enable
开启 yarn 的日志聚合功能
- yarn.log-aggregation.retain-seconds
在 HDFS 上聚合的日志最长保留多少秒
- yarn.resourcemanager.connect.retry-interval.ms
resource manager 失联后重新链接的时间
- yarn.resourcemanager.ha.enabled
开启 resource manager 的 HA 功能，默认为 False
- yarn.resourcemanager.ha.rm-ids
配置 resource manager 的逻辑名，这里假定值为 rm1 和 rm2
- ha.zookeeper.quorum
配置 zookeeper 集群结点
- yarn.resourcemanager.ha.automatic-failover.enabled
开启 resourcemanager 故障自动切换
- yarn.resourcemanager.hostname.rm1
配置逻辑名为 rm1 的 resourcemanager 对应的真实机器名，选择 node0
- yarn.resourcemanager.hostname.rm2
配置逻辑名为 rm2 的 resourcemanager 对应的真实机器名,选择 node2
- yarn.resourcemanager.ha.id
配置本机的 resourcemanager 逻辑名，如果是 node0 机器，写 rm1，如果是 node2 机器，写 rm2
- yarn.resourcemanager.recovery.enabled
开启 resourcemanager 自恢复功能
- yarn.resourcemanager.zk-state-store.address
指定 resourcemanager 的状态信息存储在 zookeeper 中，默认存储在 FileSystem 中
- yarn.resourcemanager.store.class
指定 resourcemanager 的状态信息存储在 zookeeper 中使用的类接口
- yarn.resourcemanager.zk-address
配置 resourcemanager 与 zookeeper 的连接地址
- yarn.resourcemanager.cluster-id
配置 hdfs 集群名字，这个与 hdfs-site 还有 core-site 相关连
- yarn.app.mapreduce.am.scheduler.connection.wait.interval-ms
scheduler 失联等待连接时间

yarn.resourcemanager.address.rm1

yarn.resourcemanager.scheduler.address.rm1

yarn.resourcemanager.webapp.address.rm1

yarn.resourcemanager.resource-tracker.address.rm1

yarn.resourcemanager.admin.address.rm1

yarn.resourcemanager.ha.admin.address.rm1

yarn.resourcemanager.address.rm2

yarn.resourcemanager.scheduler.address.rm2

yarn.resourcemanager.webapp.address.rm2

yarn.resourcemanager.resource-tracker.address.rm2

yarn.resourcemanager.admin.address.rm2

yarn.resourcemanager.ha.admin.address.rm2

- yarn.nodemanager.aux-services
配置成 mapreduce_shuffle, 表示在 nodemanager 上运行 mapreduce 程序
- yarn.nodemanager.aux-services.mapreduce.shuffle.class
mapreduce 运行的入口类
- yarn.nodemanager.local-dirs
中间结果存放地址
- yarn.nodemanager.log-dirs
日志存放地址
- mapreduce.shuffle.port
nodemanager 上的 mapreduce 程序互相通讯接口

hbase-site.xml

参考

<https://www.cnblogs.com/JamesXiao/p/6202372.html>

配置步骤

以下逐步配置 hadoop。先配置 hdfs, 然后是 yarn, 最后 mapreducehistory。
配置 mapreduce, 必须先配置好 mapreducehistiry

HDFS 集群安装启动

以下四大步，是一个从零开始一直到校验步骤，每一步都很重要，不可缺省。第一次安装时需要执行以下的详细步骤，倘若是格式化后重新安装，最好是先将之前生成的临时文件夹，日志文件夹，本地的存储文件夹等全部删除，这样更为方便。

注意，全部使用用户 `hadoop` 启动

ZOOKEEPER 集群安装

zookeeper 是整个安装中最先启动得进程，用于选举 namenode。

第一个 zookeeper 安装:

- 在 192.168.6.210 上安装 zookeeper
- 将 zookeeper 源码复制到集群搭建目录下
- 在 zookeeper 源码目录 `conf` 下新建 `zoo.cfg` 文件，文件内容如下所示：

```
tickTime=2000          #心跳时间间隔，单位
dataDir=/var/lib/zookeeper #临时文件目录，也可以指定其它位置
clientPort=2181         #
initLimit=5
syncLimit=2
server.0=node0:2888:3888 # server.x = 结点名:port1:port2
server.1=node1:2888:3888
server.2=node2:2888:3888
```
- 创建 `dataDir` 文件夹，并将其权限授予 `hadoop` 用户

```
sudo mkdir -p /var/lib/zookeeper
sudo chown -R hadoop.hadoop /var/lib/zookeeper
```
- 创建 `myid` 文件

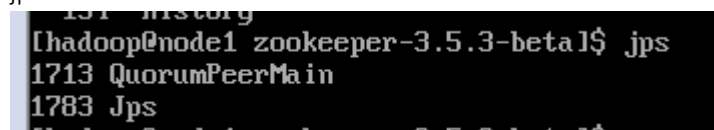
```
sudo vi /var/lib/zookeeper/myid
```

根据 zookeeper 的配置规则，每个 zookeeper 结点都必须在 `dataDir` 下新建一个 `myid` 文件，并将 `zoo.conf` 对应的 `x` 值写入 `myid` 中。如 `node1` 机器里写 1，`node2` 机器里写 2，`node0` 机器里写 0，仅仅一行。

这里写 0 即可
- 在源码目录下启动 zookeeper

```
bin/zkServer.sh start
```
- 确认 zookeeper 已经启动

```
jps
```



```
1713 QuorumPeerMain
1783 Jps
```

其余 zookeeper 安装

Journalnode 集群启动

确保 hdfs 指定的所有 journalnode 都已经启动。journalnode 将会用于各个 namenode 之间的数据同步。另外，请不要将 datanode 启动，在 HA 中 datanode 是在最后启动的。

Journalnode 相关指令：

在 namenode 中启动所有 journalnode：

```
sbin/hadoop-daemons.sh start journalnode
```

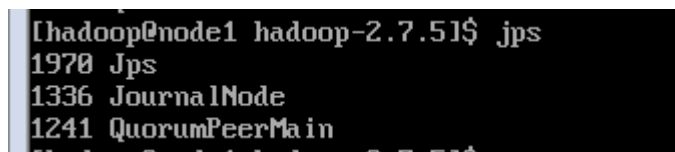
在 journalnode 中单独启动 journalnode

```
sbin/hadoop-daemon.sh start journalnode
```

这里推荐在某一个 namenode 下执行以下指令，启动所有 journalnode：

```
sbin/hadoop-daemons.sh start journalnode
```

使用 jps 指令查看 journalnode 是否启动：



```
[hadoop@node1 hadoop-2.7.5]$ jps
1970 Jps
1336 JournalNode
1241 QuorumPeerMain
```

上图表明 zookeeper 和 journalnode 已经启动

edits 队列同步：

开始同步所有 namenode 之间得信息

- 选择一台 namenode 格式化 Namenode 数据，并将数据同步到其它机器（一直选择 Y 即可）

```
bin/hdfs namenode -format
```

```
sbin/hadoop-daemon.sh start namenode
```

- 选择另一台 namenode:

先同步 edits

```
bin/hdfs namenode -bootstrapStandby
```

此处若失败，一查看配置文件是否正确，二确认 hosts 文件配置，三查看机器防火墙配置

若依旧失败，可以重新执行上一步，将所有的日志全部删除前，将 HA 集群中的所有点下/tmp/hadoop-hadoop 文件夹下的内容全部删除（即 dfs.namenode.name.dir 配置的路径清空）

然后开启 namenode

```
sbin/hadoop-daemon.sh start namenode
```

- 在其中一个 namenode 上初始化 zkfc:
bin/hdfs zkfc -formatZK

HDFS(HA)集群启动

- 启动 HA 集群
- 先停止上面节点:
sbin/stop-dfs.sh
- 全面启动
sbin/start-dfs.sh

附 HDFS(HA)的配置文件参考：

除了以下说到的配置文件以下，其余的配置直接从第一台机器复制即可

- zookeeper 的 myid 在不同机器不一样
- yarn 的 resourcemanager id

配置 java 运行环境

```
vi etc/hadoop/hadoop-env.sh
export JAVA_HOME=/lovelsl/java/jdk
```

配置 hadoop 核心参数

```
vi etc/hadoop/core-site.sh
<configuration>
```

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://kvdata</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/lovelsl/hadoop/tmp</value>
</property>

<property>
  <name>ha.zookeeper.quorum</name>
  <value>node1:2181,node2:2181,node0:2181</value>
```

```
</property>
```

```
</configuration>
```

配置 hdfs

```
vi  etc/hadoop/hdfs-site.sh
```

```
<configuration>
```

```
  <property>
```

```
    <name>dfs.replication</name>
```

```
    <value>1</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>dfs.nameservices</name>
```

```
    <value>kvdata</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>dfs.ha.namenodes.kvdata</name>
```

```
    <value>node0,node2</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>dfs.namenode.rpc-address.kvdata.node0</name>
```

```
    <value>node0:8020</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>dfs.namenode.rpc-address.kvdata.node2</name>
```

```
    <value>node2:8020</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>dfs.namenode.http-address.kvdata.node0</name>
```

```
    <value>node0:50070</value>
```

```
  </property>
```

```
  <property>
```

```
    <name>dfs.namenode.http-address.kvdata.node2</name>
```

```
    <value>node2:50070</value>
```

```
  </property>
```

```

<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>qjournal://node1:8485;node2:8485;node0:8485/kvdata</value>
</property>
<property>
  <name>dfs.client.failover.proxy.provider.kvdata</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProv
ider</value>
</property>
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/hadoop/.ssh/id_rsa</value>
</property>
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/lovelsl/hadoop/journal/data</value>
</property>
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>

</configuration>

```

配置 datanode

```

vi etc/hadoop/slaves
node0
node1
node2

```

YARN 集群配置

- 配置 yarn 需要同时配置 mapreduce，具体参考 mapreduce-history。
- 配置 yarn-env.sh，添加 java 运行环境


```

vi  etc/hadoop/yarn-env.sh
export  JAVA_HOME=/lovelsl/java/jdk

```
- 配置 yarn

参考 yarn-site.xml

- 启动 yarn

在 namenode→node1 上执行：

sbin/start-yarn.sh

注意：以下配置文件是 node0 的配置文件。由于配置时，令 node0, node2 为 namenode，因此 yarn-site.xml 文件，复制到 node1 时不用更改，复制到 node2 时需要更改 yarn 逻辑名为 rm2

yarn-site.xml 配置文件

<configuration>

```
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>
<property>
  <name>yarn.log-aggregation.retain-seconds</name>
  <value>259200</value>
</property>
<property>
  <name>yarn.resourcemanager.connect.retry-interval.ms</name>
  <value>2000</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.rm-ids</name>
  <value>rm1,rm2</value>
</property>
<property>
  <name>ha.zookeeper.quorum</name>
  <value>node0:2181,node1:2181,node2:2181</value>
</property>
<property>
  <name>yarn.resourcemanager.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname.rm1</name>
  <value>node0</value>
</property>
```

```

<property>
  <name>yarn.resourcemanager.hostname.rm2</name>
  <value>node2</value>
</property>

<property>
  <name>yarn.resourcemanager.ha.id</name>
  <value>rm1</value>
</property>

<property>
  <name>yarn.resourcemanager.recovery.enabled</name>
  <value>true</value>
</property>
<property>
  <name>yarn.resourcemanager.zk-state-store.address</name>
  <value>node0:2181,node1:2181,node2:2181</value>
</property>

<property>
  <name>yarn.resourcemanager.store.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>
</property>

<property>
  <name>yarn.resourcemanager.zk-address</name>
  <value>node0:2181,node1:2181,node2:2181</value>
</property>

<property>
  <name>yarn.resourcemanager.cluster-id</name>
  <value>kvdata</value>
</property>

<property>
  <name>yarn.app.mapreduce.am.scheduler.connection.wait.interval-ms</name>
  <value>5000</value>
</property>

<property>
  <name>yarn.resourcemanager.address.rm1</name>
  <value>node0:8132</value>

```

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.rm1</name>

<value>node0:8130</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address.rm1</name>

<value>node0:8088</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm1</name>

<value>node0:8131</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address.rm1</name>

<value>node0:8033</value>

</property>

<property>

<name>yarn.resourcemanager.ha.admin.address.rm1</name>

<value>node0:23142</value>

</property>

<property>

<name>yarn.resourcemanager.address.rm2</name>

<value>node2:8132</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.rm2</name>

<value>node2:8130</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address.rm2</name>

<value>node2:8088</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm2</name>


```
    <value>node2:8131</value>
  </property>

  <property>
    <name>yarn.resourcemanager.admin.address.rm2</name>
    <value>node2:8033</value>
  </property>

  <property>
    <name>yarn.resourcemanager.ha.admin.address.rm2</name>
    <value>node2:23142</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>

  <property>
    <name>yarn.nodemanager.local-dirs</name>
    <value>/lovelsl/hadoop/storage/yarn/local</value>
  </property>

  <property>
    <name>yarn.nodemanager.log-dirs</name>
    <value>/lovelsl/hadoop/storage/yarn/logs</value>
  </property>

  <property>
    <name>mapreduce.shuffle.port</name>
    <value>23080</value>
  </property>

</configuration>
```

mapreduce-history 集群配置

无论如何配置 mapreduce 必须先配置 maprecue-history

- 配置 java 运行环境
vi etc/Hadoop/mapred-env.sh
export JAVA_HOME=/lovelsl/java/jdk
- 配置 mapreduce 运行参数
配置 mapreduce-history, 指定 mapreduce-history 主机为 node2, 并将下面的配置文件复制到 node0,node1,node2

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>node2:10020</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>node2:19888</value>
  </property>
</configuration>
```

- 在 namenode 上启动
sbin/mr-jobhistory-daemon.sh start historyserver

HADOOP 启动检测

说明无论是 hdfs, 还是 yarn 所有功能的端口都是可以根据情况配置的, 因此如果遇到校验问题, 需要参考配置文件中的设置, 再细致分析。

HDFS 启动检测

HDFS 启动进程验证

- node0:
[root@node0 ~]# jps
18243 Jps
1620 QuorumPeerMain #zookeeper
1892 DataNode
1782 NameNode
2090 JournalNode #edits 日志队列
2282 DFSZKFailoverController #namenode 选举

- node1:

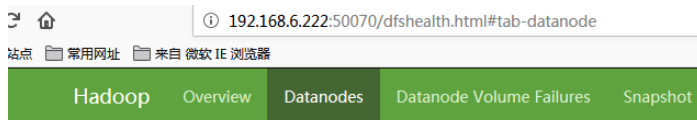
```
[root@node1 ~]# jps
2272 QuorumPeerMain
2361 DataNode
2459 JournalNode
3503 Jps
```

- node2

```
2608 DFSZKFailoverController
2371 DataNode
8004 Jps
2214 QuorumPeerMain
2300 NameNode
2478 JournalNode
```

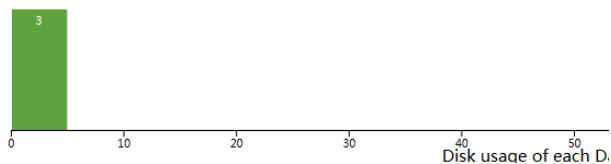
HDFS 页面访问验证

在网页中输入 namenode 的 ip 地址，找到状态为 active 的 namenode，确认所有 datanode 可以连接到 namenode



Datanode Information

Datanode usage histogram



In operation

Node	Last contact	Admin State	Capacity	Used
node1:50010 (192.168.6.211:50010)	1	In Service	49.98 GB	8 KB
node2:50010 (192.168.6.222:50010)	1	In Service	49.98 GB	8 KB
node0:50010 (192.168.6.210:50010)	2	In Service	49.98 GB	16 KB

Hadoop 命令行接口

hadoop 支持的命令种类繁多，这里主要讲述 dfs 相关的指令。

Hadoop 支持类 unix shell 的命令行操作，形如：bin/hadoop fs <args>
在目前版本中以上指令还可以如此操作:bin/hdfs dfs <args>

测试指令

查看存储跟目录下的文件夹结构：

```
hadoop fs -ls /
```

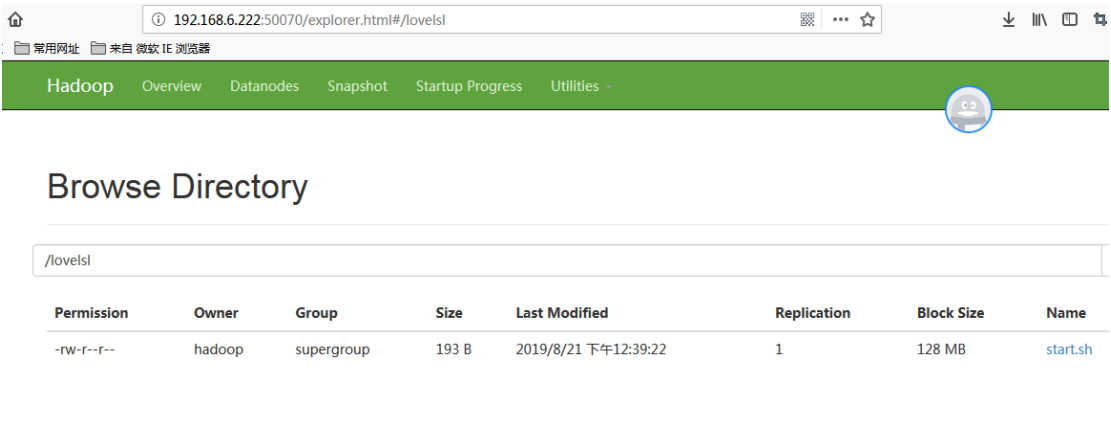
在 hadoop 存储中创建一个文件夹

```
hadoop fs -mkdir /lovelsl
```

向 hadoop 中指定文件夹上传一个文件

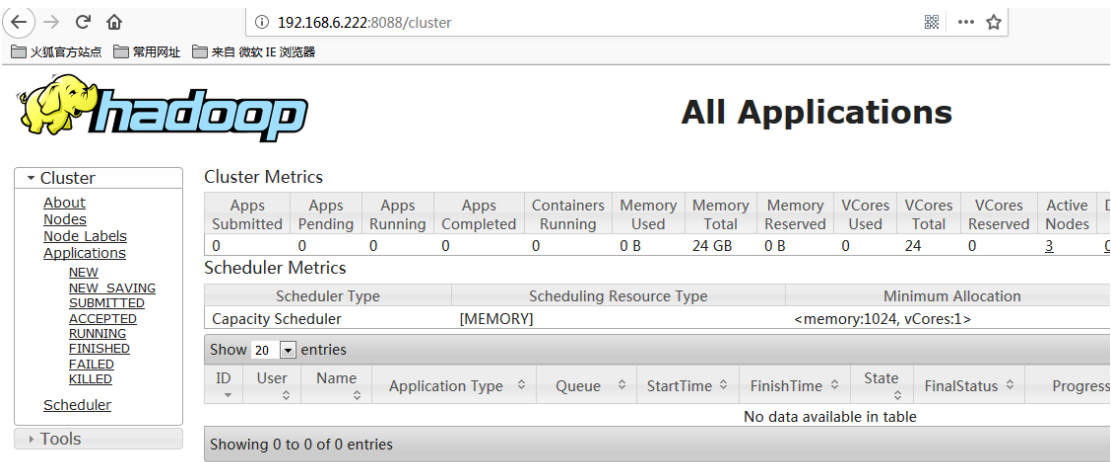
```
hadoop fs -put start.sh /lovelsl
```

指令执行完毕后，从 web 上访问 192.168.6.222:50070，以下是效果图



YARN 启动检测

YARN 页面访问验证



YARN 启动进程验证

以下是重启后的记录，进程号与上面不一致，可以忽略，主要查看各进程名

Node0

```
[root@node0 ~]# jps
```

2944 Jps

2115 JournalNode

1914 DataNode

2314 DFSZKFailoverController

2570 NodeManager

1643 QuorumPeerMain

2462 ResourceManager

1807 NameNode

Node1

```
[root@node1 ~]# jps
```

2384 DataNode

2752 Jps

2482 JournalNode

2291 QuorumPeerMain

2598 NodeManager

Node2

```
[root@node2 ~]# jps
```

2768 NodeManager

2323 NameNode

2501 JournalNode

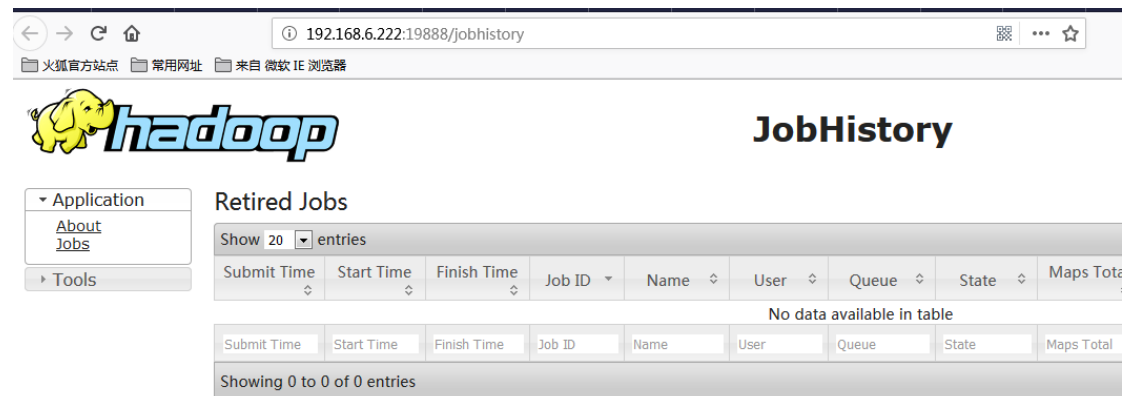
2232 QuorumPeerMain

2937 Jps

2397 DataNode

2639 DFSZKFailoverController

mapreduce-history 启动检测



HADOOP 重启

正常情况下 HDFS 的开机和关机

在 namenode 上执行：

遇上突然接到停电通知时，可以按照下面的指令执行

- 先停止上面节点：
sbin/stop-dfs.sh
此时每台机器，jps 查看只剩下 QuorumPeerMain
- 全面启动
确保每台机器的 zookeeper 已经启动
sbin/start-dfs.sh

正常情况下 YARAN 的开机和关机

在 namenode 上执行->node1：

- 启动 yarn 结点
sbin/start-yarn.sh
- 停止 yarn 结点
sbin/stop-yarn.sh

正常情况下 mapreduce-history 的开机和关机

mapreduce-history 主机为 node2，在 Node2 执行

- 停止 mapreduce-history
sbin/mr-jobhistory-daemon.sh stop historyserver
- 启动 mapreduce-history
sbin/mr-jobhistory-daemon.sh start historyserver

HADOOP 其余指令

某些特殊情况下，部分进程无法正常关闭或者打开，这时需要手动单独将它打开。

- resourcemanager 关闭
sbin/yarn-daemon.sh stop resourcemanager

HBASE 完全分布式

Hbase+ hadoop 账户配置

ip	192.168.6.210	192.168.6.211	192.168.6.222
hostname	node0	node1	node2
user	hadoop	hadoop	hadoop
ssh	√	√	√
ntp	Server	Client	Client
zookeeper—myid	0	1	2
hmaster	√(备)	√(主)	
regionserver	√	√	√

HBASE 完全分布式配置

第一个结点的配置：

先配置好一台 HBASE 的机器，然后将这些参数复制到其它机器上。这里选择了 node1

- 为 node1 配置无密码访问 node1, node2, node0
- 配置环境 JAVA 环境变量和 HBASE 环境变量

- 配置 hbase/conf/hbase-env.sh
指定 JAVA 路径
export JAVA_HOME=/lovelsl/java/jdk
将错误的 java 参数注释
Configure PermSize. Only needed in JDK7. You can safely remove it for JDK8+
#export HBASE_MASTER_OPTS="\$HBASE_MASTER_OPTS -XX:PermSize=128m
-XX:MaxPermSize=128m -XX:ReservedCodeCacheSize=256m"
#export HBASE_REGIONSERVER_OPTS="\$HBASE_REGIONSERVER_OPTS
-XX:PermSize=128m -XX:MaxPermSize=128m -XX:ReservedCodeCacheSize=256m"
指定 zookeeper 的模式为由 zookeeper 结点负责
export HBASE_MANAGES_ZK=false
- 配置 hbase-site.xml
<configuration>
 <property>
 指定 hbase 使用的 hadoop 位置，注意 mycluster 时集群名，配置 hadoop 时
指定的，如果遗忘去 hadoop 的 xml 配置文件中寻找，端口是 namenode 的访问端
口
 <name>hbase.rootdir</name>
 <value>hdfs://kvdata:8020/hbase</value>
 </property>
 <property>
 #指定采用分布式搭建 hbase
 <name>hbase.cluster.distributed</name>
 <value>true</value>
 </property>
 <property>
 指定 zookeeper 的结点
 <name>hbase.zookeeper.quorum</name>
 <value>node1,node2,node3</value>
 </property>
</configuration>
- 配置 regionserver 的结点
node1
node2
node0

其余结点的配置：

在 hbase 中，regionserver 和 hmaster 的配置文件等完全一致，只是启动方式不一样

- 配置 backup-masters 和 regionserver
将 hbase-env.sh 和 hbase-site.xml，以及 regionserver 中的内容复制到 node0,node2
上。

HBASE 与 HADOOP 协调

Hbase 的内容存储在 hadoop 中，因此两者需要协调配置。

以下操作在所有 regionserver 和 hmaster 均要执行

- 复制 hadoop 的配置文件到 hbase 的 conf 下
`cp /lovelsl/hadoop/hadoop/etc/hadoop/hdfs-site.xml .`

HBASE 完全分布式启动

Hbase 启动时，首先需要在主结点上，将 hbase 的所有 regionserver 和主 hmaster 启动，然后单独在备用结点启动备用 hmaster。

- 在主节点 node1：
`bin/start-hbase.sh`
- 在备用结点 node0:
`bin/hbase-daemon.sh start master`

HBASE 启动检测

HBASE 页面访问检测

访问 HBASE 的主结点页面

←

→

↺

🏠

192.168.6.211:16010/master-status

🔍

...

☆

📁 火狐官方网站📁 常用网址📁 来自 微软 IE 浏览器

A P A C H E

HBASE

Home

Table Details

Procedures

Local Logs

Log Level

Debug Dump

Metrics Dump

HBase Configuration

Master node1

Region Servers

Base StatsMemoryRequestsStorefilesCompactions

ServerName	Start time	Version	Requests Per Second
node0,16020,1566557908096	Fri Aug 23 18:58:28 CST 2019	1.3.2	0
node1,16020,1566557910560	Fri Aug 23 18:58:30 CST 2019	1.3.2	0
node2,16020,1566557906867	Fri Aug 23 18:58:26 CST 2019	1.3.2	0
Total: 3			0

Backup Masters

ServerName	Port	Start Time
node0	16000	Fri Aug 23 19:03:29 CST 2019
Total: 1		

访问 HBASE 备用结点页面

←

→

↺

🏠

192.168.6.210:16010/master-status

🔍

...

☆

📁 火狐官方网站📁 常用网址📁 来自 微软 IE 浏览器

A P A C H E

HBASE

Home

Table Details

Procedures

Local Logs

Log Level

Debug Dump

Metrics Dump

HBase Configuration

Backup Master node0

Current Active Master: node1

Tasks

Show All Monitored TasksShow non-RPC TasksShow All RPC Handler TasksShow Active RPC CallsShow Client OperationsView as JSC

Start Time	Description	State	Status
Fri Aug 23 19:03:33 CST 2019	Master startup	RUNNING (since 25mins, 54sec ago)	Another master is the active master, node1,16000,15665579073 (since 25mins, 54sec ago)

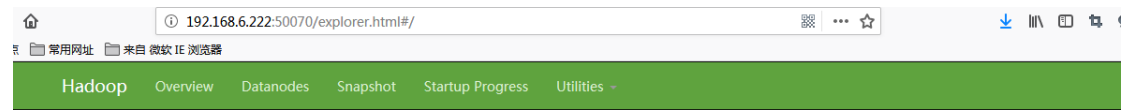
Software Attributes

Attribute Name	Value	Description
HBase Version	1.3.2, revision=1bedb5bfb5a99067e7bc54718c3124f632b6e17	HBase version and revision
HBase Compiled	Mon Mar 19 18:47:19 UTC 2018, root	When HBase version was compiled
HBase Source Checksum	e7f36f244609783299ccbc41a8ee0dd4	HBase source MD5 checksum
Hadoop Version	2.5.1, revision=2e18d179e4a8065b6a9f29cf2de9451891265cce	Hadoop version and revision
Hadoop Compiled	2014-09-05T23:05Z, kasha	When Hadoop version was compiled

在 hadoop 中验证存在 habse 的存储路径

以下/hbase 是在配置文件中定义的 hbase 在 hadoop 中的存储位置

/tmp 文件夹是 hbase 的临时文件在 hdfs 中的位置



Browse Directory

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	2019/8/23 下午6:58:42	0	0 B	hbase
drwxr-xr-x	hadoop	supergroup	0 B	2019/8/21 下午12:39:22	0	0 B	loveisl
drwxrwx---	hadoop	supergroup	0 B	2019/8/23 上午9:51:53	0	0 B	tmp

HBASE 启动进程校验

注意：由于上面的 hadoop yarn 启动选举的 resourcemanager 与本次不一样，所以 resource manager 所在的进程机器不一样。

node1：HBASE 主

```
[root@node1 ~]# jps
2272 QuorumPeerMain      #zookeeper
4020 NodeManager          #yarn
4920 HRegionServer       # HBASE
2361 DataNode            # hdfs
4793 HMaster             # HBASE 主
2459 JournalNode         # HDFS 日志
5407 Jps
```

node0：HBASE 备

```
[root@node0 ~]# jps
1620 QuorumPeerMain      #zookeeper
1892 DataNode            #yarn
1782 NameNode            # HBASE
27512 HRegionServer      # HBASE
28201 Jps
2090 JournalNode         # HDFS 日志
2282 DFSZKFailoverController #HDFS
20029 NodeManager        #yarn
27775 HMaster            # HBASE 备用
```

Node2

```
[root@node2 ~]# jps
```

2608 DFSZKFailoverController
9425 JobHistoryServer
2371 DataNode
12387 Jps
2214 QuorumPeerMain
9048 NodeManager
8618 ResourceManager
2300 NameNode
12045 HRegionServer
2478 JournalNode

HBASE 重启

单执行 HBASE 重启，不会丢失 HBASE 数据。将 HBASE 存储在 HDFS 中的数据删除，才会丢失数据。格式化 HBASE 最号将临时文件夹 hdfs 中的数据一并删除。

- 在主节点 node1：
bin/stop-hbase.sh
- 在备用结点 node0:
bin/hbase-daemon.sh stop master
- 在主节点 node1：
bin/start-hbase.sh
- 在备用结点 node0:
bin/hbase-daemon.sh start master

HBASE 其余指令

- 异常 regionserver 关闭
在异常运行的结点上运行：
bin/hbase-daemon.sh stop regionserver

Hive 远程模式（多用户模式）

远程模式下的角色定位

ip	192.168.6.210	192.168.6.211	192.168.6.222
hostname	node0	node1	node2

user	hadoop	hadoop	hadoop
ssh	√	√	√
ntp	Server	Client	Client
zookeeper—myid	0	1	2
mariadb	√		
Metaserver			√
metaclient		√	

Hive 远程模式（多用户模式）配置

不同于 hadoop 和 hbase 下大多数配置文件一致，hive 中配置需要根据实际需求配置。如下所示的配置中，node0 是 mysql 主机，因为不做为 hive 中的 metaserver 或者 metaclient，因此在配置完 mysql 后，不做任何设置。而 metaserver 和 metaclient 因为功能不一样，因此做的配置也完全不一样。

但是，假如有其它主机想要连接 hive 使用命令行客户端，则必须复制 metaclient 的配置。

Mysql 主机配置：

Node0 配置：Mysql 用于存储 hive 的元数据，安装时直接使用 yum install mysql 即可，这里默认安装的是 mariadb，无论是那个都可以。

创建一个允许远程访问的 mysql 账号

```
mysql> GRANT ALL PRIVILEGES ON *.* To 'ls1'@'%' IDENTIFIED BY 'ls1';
Query OK, 0 rows affected (0.00 sec)
```

Meta server 主机配置：

Node2 配置：

Metaserver 配置

Hive 的客户端是通过 meta server 与 mysql 和 hadoop 做数据处理和记录。

- 配置文件 hive-site.xml 解析

```
<configuration>
```

```
#配置 hive 元数据存储位置，该处设置需要与 meta client 中一致
```

```
<property>
```

```
<name>hive.metastore.warehouse.dir</name>
```

```

        <value>/user/hive/warehouse</value>
    </property>
</property>
    <name>hive.metastore.schema.verification</name>
    <value>>false</value>
</property>

#配置 hive 连接 mysql
<property>
    <name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://192.168.6.210:3306/hive?createDatabaseIfNotExist=true&character
Encoding=UTF-8&serverTimezone=UTC</value>
</property>
#配置 hive 连接 mysql 时，程序应该导入的类名
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.cj.jdbc.Driver</value>
    <description>Driver class name for a JDBC metastore</description>
</property>
#配置 hive 连接 mysql 的账户
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hsl</value>
    <description>Username to use against metastore database</description>
</property>
#配置 hive 连接 mysql 的密码
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hsl</value>
    <description>password to use against metastore database</description>
</property>
</configuration>

```

为 metasever 配置 mysql 的 jar 驱动

注意，这里虽然使用的是 mariadb，但实际使用的还是 mysql 的驱动包。因为 mariadb 没有适合的 java 驱动。

这里使用的是 mysql-connector-java-8.0.16.jar
 将该驱动包复制到 hive 的 lib 目录下

Meta client 主机配置：

Node1 配置：

Meta client 配置

Hive 的客户端是通过 meta server 与 mysql 和 hadoop 做数据处理和记录。

- 配置文件 hive-site.xml 解析
#配置 hive 元数据存储位置，该处设置需要与 meta server 中一致

```
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/user/hive/warehouse</value>
</property>
<property>
  <name>hive.metastore.schema.verification</name>
  <value>>false</value>
</property>

#meta server 位置
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://node2:9083</value>
</property>
```

为 meta client 配置 jline

一般 hive 安装包 lib 文件夹下的 jline 文件版本与 hadoop 安装包下的 share/hadoop/yarn 文件夹下的 jline 文件版本不一致，在版本不一致时需要将版本统一处理，统一使用高版本即可，即将 hive 中的 jline 替换 hadoop 中的对应文件。

Hive 远程模式（多用户模式）启动

服务端操作

Node2

- 初始化元数据
最好先确认没有 hive 数据库
schematool -initSchema -dbType mysql

- 启动服务端
推荐运行指令
hive --service metastore

```
jps  
runjar
```

客户端操作

```
Node1  
hive
```

Hive 启动检测

Hive 元数据库查看

正如 `javax.jdo.option.ConnectionURL` 设置一样，node0 上创建了元数据库 hive

无法访问 hdfs 的/tmp 目录时执行：

```
hadoop dfs -chmod -R 755 /tmp  
hadoop dfs -chmod -R 777 /tmp
```



```

[root@node0 ~]# mysql -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 91
Server version: 5.5.60-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| hive |
| mysql |
| performance_schema |
| test |
+-----+
5 rows in set (0.00 sec)

MariaDB [(none)]> 

```

进程启动检测

```

[root@node2 ~]# jps
2768 NodeManager
3569 RunJar          #hive
3714 Jps
2323 NameNode
2501 JournalNode
2997 JobHistoryServer
2232 QuorumPeerMain
3112 HRegionServer
2397 DataNode
2639 DFSZKFailoverController

```

Hive 重启

Hive 的数据分为存储在 hdfs 中的块数据和存储在 mysql 中的元数据，重启不会丢失数据。格式化数据需要删除 mysql 中的元数据数据库，和 hdfs 中的数据，最好将临时文件中的数据一并删除。

注意配置的 node2 为服务端，因此只能在 node2 启动服务端，对应的客户端为 node0，所以也只能在 node0 上使用命令行客户端连接测试。

- 关闭服务端
 - ps aux | grep hive
 - kill -9 pid

- 启动服务端
推荐运行指令
`hive --service metastore`

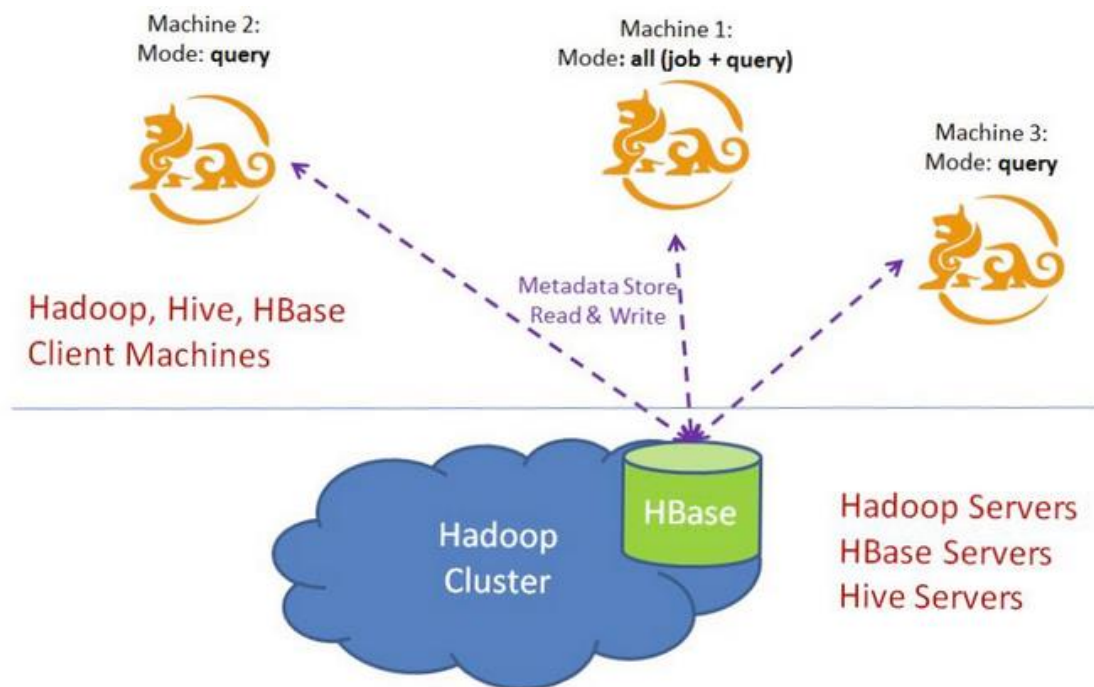
APACHE KYLIN 安装

Apache kylin 是一个 OLAP 的大数据分析框架，支持集群和单机方式安装。

APACHE KYLIN 与 HADOOP 集群关系：

Kylin 实例是无状态的服务，运行时的状态信息存储在 HBase metastore 中。出于负载均衡的考虑，您可以启用多个共享一个 metastore 的 Kylin 实例，使得各个节点分担查询压力且互为备份，从而提高服务的可用性。

从下图的集群部署图可以得知，在整个集群中，kylin 会扮演 hadoop，hive，hbase 的客户端角色。因此需要与 hadoop，hive，hbase 做交叉配置。另外 kylin 可以自行配置成一个单独的 Kylin 集群。



Apache kylin 配置参考解析

<http://kylin.apache.org/cn/docs/install/configuration.html>

APACHE KYLIN 配置

此处因为系统资源原因, 选择 node0 作为 apache kylin 的服务端和 web 端, 因为 apache kylin 在实际运行中会作为 hive 的客户端, 而在上一节中, 我们只在 node1 上配置 hive 客户端, 所以需要配置 node0 上的 hive 客户端

配置 node0 的 hive 访问客户端

Node0 配置 :

Meta client 配置

Hive 的客户端是通过 meta server 与 mysql 和 hadoop 做数据处理和记录。

- 配置文件 hive-site.xml 解析
#配置 hive 元数据存储位置, 该处设置需要与 meta server 中一致

```
<property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
</property>
<property>
    <name>hive.metastore.schema.validation</name>
    <value>>false</value>
</property>

#meta server 位置
<property>
    <value>thrift://node2:9083</value>
</property>
```

为 meta client 配置 jline

一般 hive 安装包 lib 文件夹下的 jline 文件版本与 hadoop 安装包下的 share/hadoop/yarn 文件夹下的 jline 文件版本不一致, 在版本不一致时需要将版本统一处理, 统一使用高版本即可, 即将 hive 中的 jline 替换 hadoop 中的对应文件。

配置 apache kylin 的系统参数

Vi kylin.properties

- Kylin 在 hdfs 中的存储根路径
kylin.env.hdfs-working-dir=/kylin
- ## DEV|QA|PROD. DEV will turn on some dev features, QA and PROD has no difference in terms of functions.

kylin.env=QA

- Kylin 的元数据存储 在 hbase 中的位置
kylin.metadata.url=kylin_metadata@hbase
- Kylin 在 zookeeper 中的存储根路径
kylin.env.zookeeper-base-path=/kylin
- Kylin 连接的 zookeeper
kylin.env.zookeeper-connect-string=node0:2181,node1:2181,node2:2181
- ## Hadoop conf folder, will export this as "HADOOP_CONF_DIR" to run spark-submit

This must contain site xmls of core, yarn, hive, and hbase in one folder

kylin.env.hadoop-conf-dir=/lovelsl/hadoop/hadoop/etc/hadoop

kylin.server.mode=all

List of web servers in use, this enables one web server instance to sync up with other servers.

- Kylin 的 web 服务器，可以有多个使用 nginx 负载均衡即可
kylin.server.cluster-servers=192.168.6.210:7070
- Kylin 的 hbase 客户端超时时间
- Kylin 的 hbase
- 指令 kylin 使用的 hbase 目录
改值，必须与 hbase 的 hbase.rootdir 配置值一样，否则 kylin 无法存储和读取数据

配置 apache kylin 的无 SSL 访问

默认开启，会导致不能访问。vi tomcat/conf/server.xml

注释掉以下内容，大概在 84-87 行

```
<!-- <Connector port="7443" protocol="org.apache.coyote.http11.Http1Protocol"
    maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
    keystoreFile="conf/.keystore" keystorePass="changeit"
    clientAuth="false" sslProtocol="TLS" /> -->
```

配置前端代理访问 apache kylin 的 WEB UI

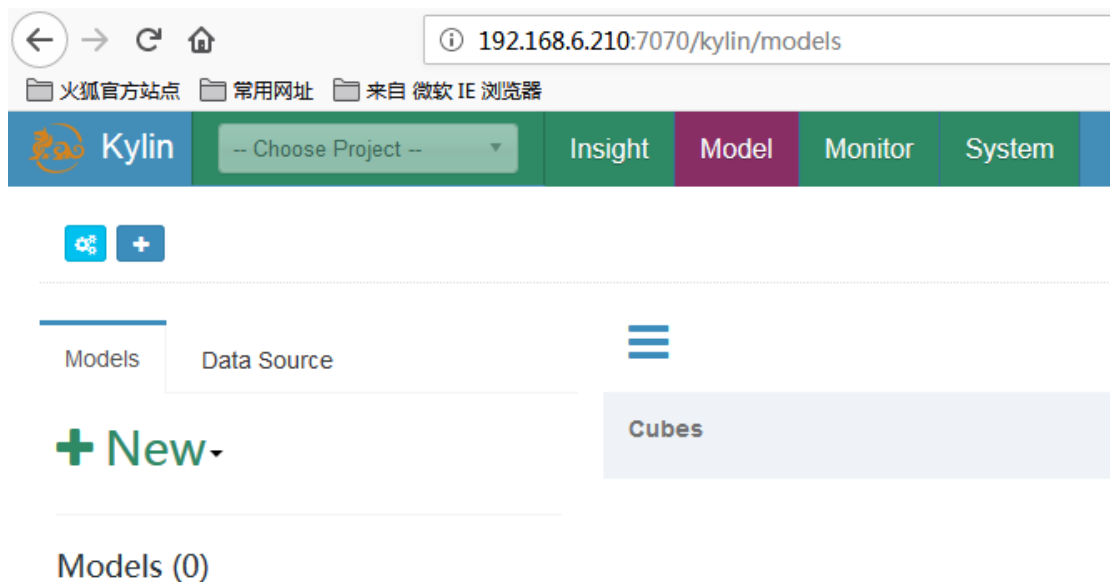
默认情况下只指定一个 web ui，但在实际应用中，可以使用 nginx 做为负载均衡，在后端配置多个 web ui。

APACHE KYLIN 启动：

- Kylin 的启动检测
bin/check-env.sh
- Kylin 启动
bin/kylin.sh start
- Kylin 停止
bin/kylin.sh stop

APACHE KYLIN 启动检测

Apache kylin 启动后，登录 UI 时，会用到账号 ADMIN 密码 KYLIN 登录账户。
访问 url 为 192.168.6.210:7070/kylin，以下是登录后的效果。



以上是运行初步检测，获取到 Web ui 后，可以先执行 bin/kylin.sh stop 暂停 web ui，然后在运行 bin/sample.sh 导入 kylin 官方测试数据后，重新启动 bin/kyin.sh start。