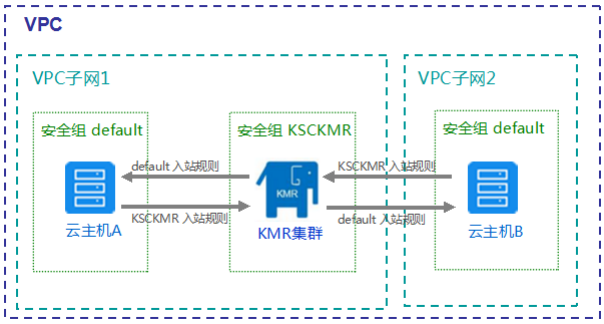
目录

VPC网络配置

KMR网络架构

KMR采用云主机（Kingsoft Cloud Elastic Compute，简称KEC）构建集群，二者服务都依托于VPC，创建于用户指定的VPC网络和VPC子网中，VPC具有安全组功能，将VPC中的产品划分不同的安全域，为每个安全域定义不同的访问控制规则。KMR集群默认安全组为“KSCKMR”，集群内部各节点在该安全组内可相互访问。每个云主机在创建时也会绑定唯一的安全组，默认为“default”，在同一VPC下，处于相同或不同子网的KMR集群和云主机可通过配置安全组规则进行互通。



通过KMR集群访问非KMR集群云主机

需要配置要访问的云主机对应VPC安全组的入站规则，允许KMR集群访问该云主机。

1. 从金山云控制台进入**云服务器**，获取该云主机的VPC和安全组名称。

2. 从金山云控制台进入**虚拟私有网络**，进入刚刚获取的VPC。

3. 选中该云主机对应的安全组，创建安全组规则。

4. 用户可根据需求创建不同方式的安全组规则，填写正确的KMR集群所在的网段即可。

通过非KMR集群云主机访问KMR集群

需要配置KMR集群对应VPC安全组的入站规则，允许其他云主机访问KMR集群。

首先查看KMR集群的VPC网络，然后从金山云控制台进入**虚拟私有网络**，进入该VPC，KMR集群默认的安全组为“KSCKMR”，进入该安全组并配置安全组入站规则，填写正确的非KMR集群云主机的网段即可配置互通。

> 注意：
>
> 以上配置KMR集群和其他云主机互通的前提是在同一VPC下，不同的VPC下主机互通需要先对VPC进行配置使VPC互通，再配置安全组规则。
>
> VPC安全组规则默认出站规则全部放行，因此，可不配置出站规则，如有特殊需求，可自行更改。

开通非master1节点的外网访问权限

默认情况下，KMR集群中只有Master1有外网访问权限，通过外网EIP联结外网。

如果其他节点也需要访问外网，则需要在对应VPC或子网中建议一个外网的NAT，如下图所示。

注意：一个VPC只能绑定一个NAT，因此，请删除，默认创建的NAT，再创建新的NAT。

NAT相关的问题可以联系负责网络的同事。

Spark实践指南

Apache Spark 是开源的集群框架和编程模型，与 Hadoop 类似，也是一款常用于大数据处理的分布式系统。Spark 与 Hadoop 的不同之处在于 Spark 拥有经过优化的有向无环图 (DAG) 执行引擎并会积极地在内存中缓存数据，这可提高性能，尤其是对于某些算法和交互式查询。

作为金山云 KMR 的组件之一，我们对其进行了高可用处理， 而且 Spark 可以与 KMR 的其他应用程序一同安装，同时，还可以通过 OpenAPI 或者 SDK 直接对 KS3 中的数据进行操作。除此之外 KMR 也将 Hive 与 Spark 做了集成，您可以通过 HiveContext 对象运行使用 Spark 的 Hive 相关的操作。

本文主要对一些 Spark 的基础操作进行了讲解，希望能够对您在 KMR 上使用 Spark 有所帮助。

全文会以场景的形式对 Spark 的一些功能和使用方式进行介绍，通过模拟各种场景，可以使您更快的熟悉 KMR 上的 Spark 组件的使用方法。

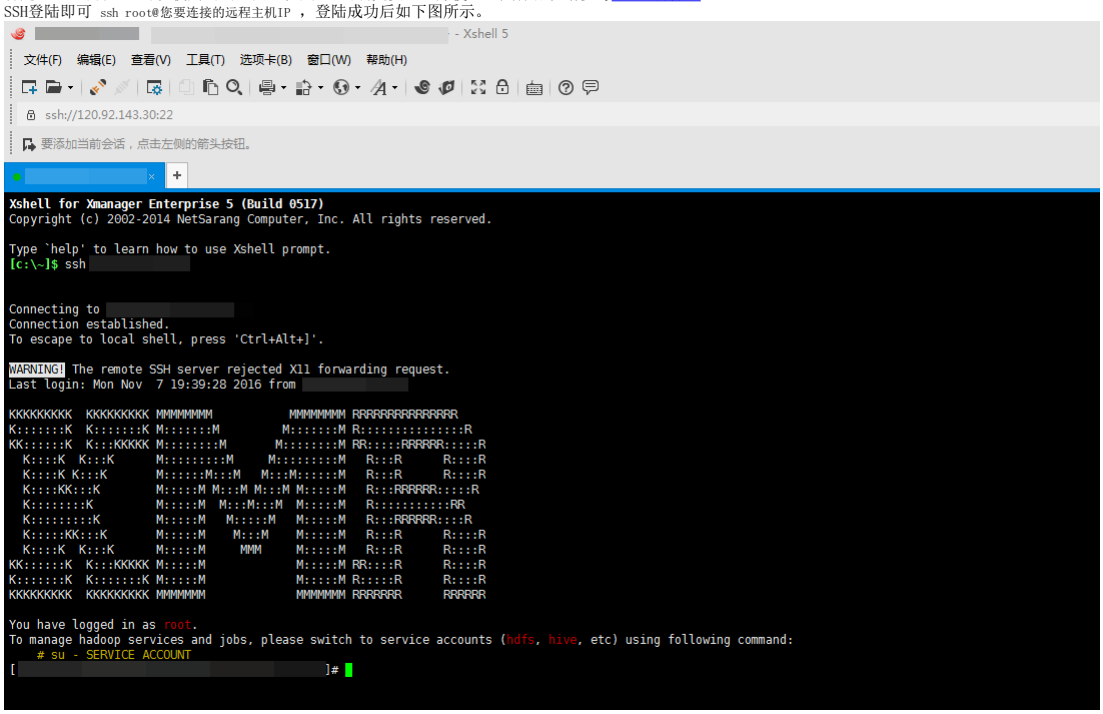
在正式了解相关内容前，您需要进行一些准备工作。

准备

1. 创建 KMR 集群并勾选 Spark 服务。 如下图所示，详情请参考创建 KMR 集群。

2. 通过SSH连接到集群。
   - 获取公网的云主机 MASTER 节点 的 IP，您可在KMR控制台，选择**集群管理**，点击您需要查看的集群，在**集群详情**页面中查看，如下图所示。

   - 绑定 SSH 密钥（也可以使用密码登陆，但是SSH密钥更安全、方便），具体流程请参考SSH密钥管理。
   - SSH登陆即可 ssh root@您要连接的远程主机IP ，登陆成功后如下图所示。



3. KMR 集群各项服务可以与KS3联用，关于如何使用KS3服务请参考数据导入。

到此已经完成所有准备工作。

Spark 离线任务处理

所谓离线任务，就是用户已经将待处理的数据存储在 KS3 上或者是 HDFS 中，这些数据是“过去”产生的，并不是实时产生的。用户在提交任务（job）以后，由集群自动完成计算并得出结果。

一般情况下，在进行数据清洗时，大多采用离线处理的方式。具体到场景来说，比如日志分析、用户行为分析（日志分析的一种）等。这些数据大都具有**一次写入，多次读取**的特点。一般要处理具有这种特点的数据大都采用离线处理的方式。

以下会有几个例子，用来熟悉两个经常遇到的场景的处理方法。

- 以下内容不做特殊说明的情况下，**凡是执行 SPARK 相关的命令时，请确保您的系统用户已经由 root用户 切换到 spark 用户；**
  - 切换当前工作目录到 /home/spark 。

```
#切换系统用户为 spark
su spark
#切换工作目录到当前用户的 home 目录
cd ~
```

入门操作

当前场景下，用户将待处理数据存储在本地计算机，需要将其上传至 HDFS 再进行计算处理。

- 程序准备
  这里将会以计算 pi（圆周率）为例来演示离线任务处理。
  用户可以使用spark-submit命令来提交 SPARK 任务。spark-submit 具体使用可以通过 spark-submit --help 查看。 *注意：此包由官方提供，因此无需上传 jar 包*

- 程序提交
  在命令行中执行如下命令

```
yarn-client
#spark spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode client --driver-memory 4g --num-executors 2 --executor-memory 2g --executor-cores 2 /usr/hdp/2.4.0.0-169/spark/lib/spark-examples*.jar 10 2>/dev/null

yarn-cluster
#spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster --driver-memory 4g --num-executors 2 --executor-memory 2g --executor-cores 2 /usr/hdp/2.4.0.0-169/spark/lib/spark-examples*.jar 10 2>/dev/null
```

注意：因为是 yarn cluster 模式，所以在终端看不到任何输出信息（这里仅指有效输出信息，debug 的标准输出已被屏蔽），需要在 yarn 的控制台 查看打印的日志 log，可通过：**AMBARI > YARN > Quick Links > ResourceManager UI**来查看。

- 相关参数说明

参数	参考值	说明
class	org.apache.spark.examples.SparkPi	作业的主类，程序的入口
master	yarn	KMR使用Yarn的模式（也支持Standalone模式，后接集群master的URL，如local或者spark:\/\/host:port，生产环境下推荐使用YARN）
yarn-client	简写，等效--masteryarn--deploy-modeclient，即--masteryarn-client	
yarn-cluster	简写，等效--masteryarn--deploy-modecluster，即--masteryarn-cluster	
deploy-mode	client	部署方式（client\/cluster），client模式表示作业的ApplicationMaster会放在Master节点上运行。注意，该参数需要和--masteryarn连用_
cluster	cluster模式表示作业的ApplicationMaster会随机的在core节点中的任意一台上启动运行。注意，该参数需要和--masteryarn连用	
driver-memory	4G	driver使用的内存
num-executors	2	创建executor的数量
executor-memory	2G	每个executor使用的最大内存，不能超过单机的最大可使用内存
executor-cores	2	各个executor使用的并发线程数目，也即每个executor最大可并发执行的Task数目

补充：上面仅是 spark-submit 一部分参数，详情请点击此处。 spark-submit 参数填写完毕，后跟 jar 包所在路径，最后 2>/dev/null 是为了屏蔽程序 debug 信息的标准输出。

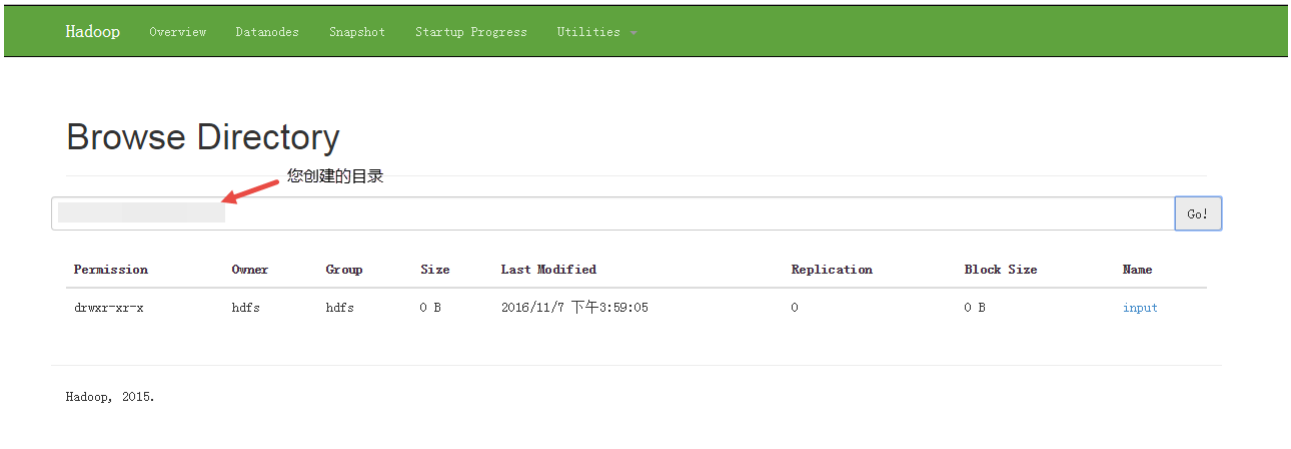
场景一：数据储存在本地

- 程序准备
  - 任务源码，点击这里下载。
  - 准备输入文件
    格式要求
    1. 任意字符串，彼此用空格作为分隔符。
    2. 若干行，也可以直接用 spark 包下面的README.md文件。
  - **上传所需 jar 包以及要处理的输入文件**
    1. 将生成的 jar 包和待处理的文件 in.file ，通过 **xftp** 上传至 MASTER 主机，假设目录为 **/home/spark** 。
    2. 将 in.file 文件上传至 HDFS （或者 KS3 ），命令如下：

```
# sudo -u hdfs hdfs dfs -mkdir -p /user/YOURNAME/testdata/input
# sudo -u hdfs hdfs dfs -put /home/spark/in.file /user/YOURNAME/testdata/input
```

*注意：第一条命令为创建文件输入目录，**请勿自行创建文件输出目录**；另外，在执行第二条命令前请检查当前所在目录。*

通过 **Ambari > HDFS > Quick Links > 任意一activity集群 > NameNode UI > Utilities > Browse the file system** 可以访问 HDFS 的 WEB 界面查看文件是否上传成功以及输出的文件，如下图所示。



也可通过以下命令查看

```
# sudo -u hdfs dfs -ls /user/YOURNAME/testdata/input
```

- 任务提交

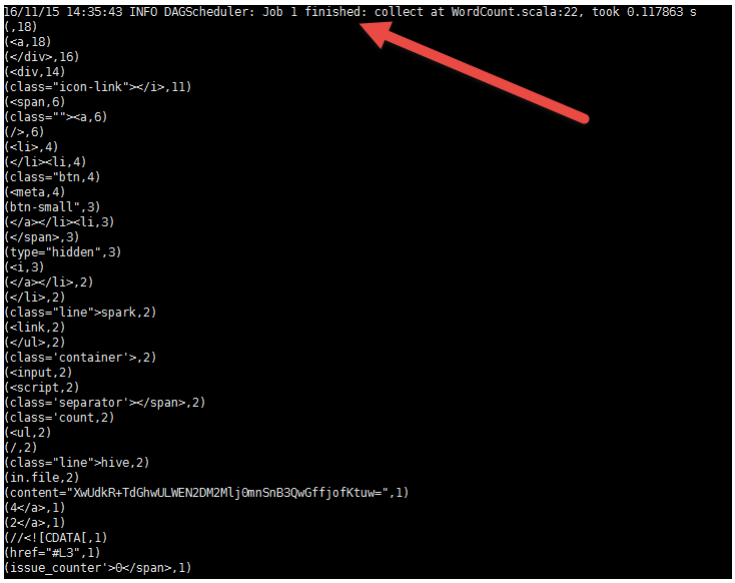
  执行下方指令即可

```
local(file on hdfs)
# sudo -u spark spark-submit --class com.kmr.demo.WordCount --master local[2] wordcount-1.0-SNAPSHOT.jar "/user/YOURNAME/testdata/input" "/user/YOURNAME/testdata/output"

yarn-client(inputfile on local, outputfile on hdfs)
#sudo -u spark spark-submit --class com.kmr.demo.WordCount --master yarn-client /home/spark/wordcount-1.0-SNAPSHOT.jar file:///usr/hdp/2.4.0.0-169/spark/README.md  hdfs:///user/YOURNAME/output

yarn-cluster (file on hdfs)
# sudo -u spark spark-submit --class com.kmr.demo.WordCount --master yarn --deploy-mode cluster /home/spark/wordcount-1.0-SNAPSHOT.jar  "/user/YOURNAME/testdata/input" "/user/YOURNAME/testdata/output"
```

- 查询结果

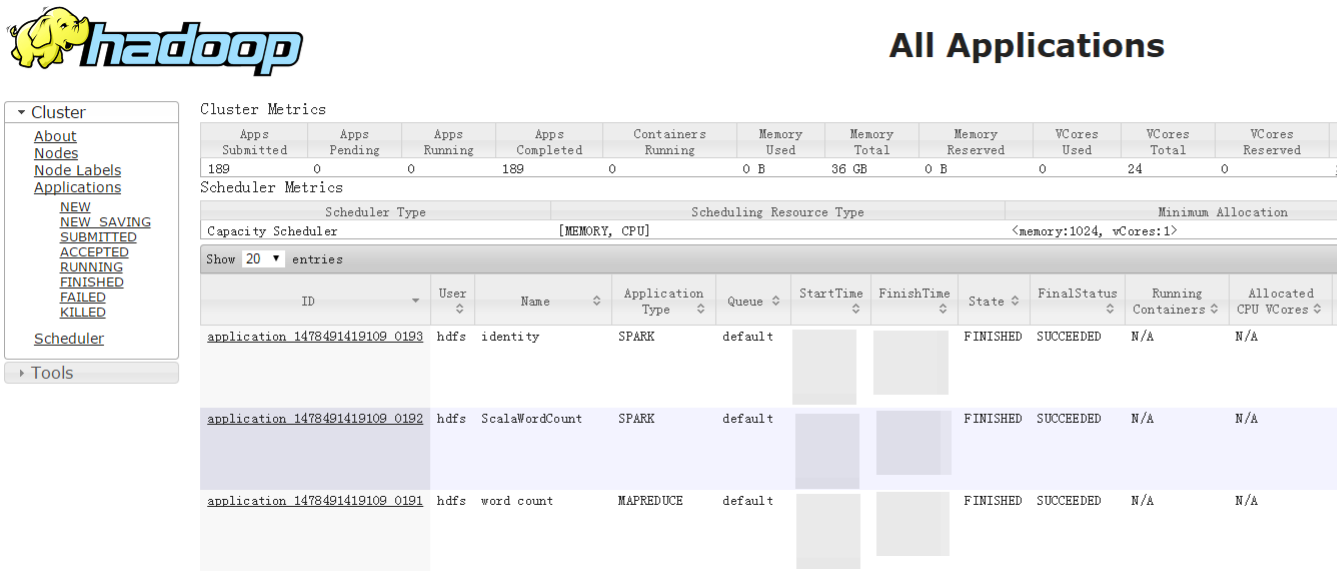
- 在命令行中查询

  在执行完命令后，在屏幕的打印信息中可以找到相应内容，如下图所示



- 在Ambari控制台查询

  任务完成后以在集群详情页 -> Ambari控制台 -> YARN -> Quicklinks -> 集群ID -> ResourceManager 查询，结果如下图所示



场景二：数据储存在KS3上

与数据存储在本地处理过程基本相同，只需在 Spark-submit 时增加一个参数--driver-class-path /usr/hdp/2.4.0.0-169/hadoop/lib/hadoop-ks3-0.1.jar。例如

```
yarn-client (file on ks3)
# sudo -u spark spark-submit --class com.kmr.demo.WordCount --driver-class-path /usr/hdp/2.4.0.0-169/hadoop/lib/hadoop-ks3-0.1.jar  --master yarn --deploy-mode client  /home/spark/wordcount-1.0-SNAPSHOT.jar  "ks3://YOURNAME/spark/input" "ks3://YOURNAME/spark/output"
```

第二章 SparkSQL 的使用及操作

在 Spark 中的一栈式解决方案中，最常用的组件之一就是 Spark SQL，它是 Spark 的一个结构化数据处理模块，其最大优势在于性能非常高，而且还使用了基于成本的优化器、列储存、代码生成等技术。此外 Spark SQL也可以扩展到上千个计算节点以及数小时的计算能力，并且支持自动容错恢复。

使用 Spark SQL 有两种方式：一种是作为分布式的 SQL 引擎，只需写 SQL 就可以进行计算，无需复杂编码；另一种是在 Spark 程序中，通过 API 的形式来操作数据。以下会有一些例子（场景）用于介绍如何在 KMR 中如何更高效的使用 Spark SQL。

以下例子中所有的源码可以点击这里下载。

数据基本操作

- **数据准备**

  数据输入源：spark 包自带 **peopele.txt** 和 **people.json**
  数据输入源路径：/usr/hdp/2.4.0.0-169/spark/examples/src/main/resources
  people.**json**

  ```
  {"name":"Michael"}
  {"name":"Andy", "age":30}
  {"name":"Justin", "age":19}
  ```

  people.**txt**

  ```
  Michael, 29
  Andy, 30
  Justin, 19
  ```

- **通过 spark-shell 创建 DataFrame 操作**

  ```
  #切换 linux 系统用户为 spark (默认登录 KMR 集群的用户为 root )
  #su spark
  #进入 spark-shell 模式，资源管理使用时 yarn ，部署方式是 client
  ```

```
$ spark-shell --master yarn-client
#创建 DataFrame
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
scala> val df = sqlContext.read.json("file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/people.json")
scala> df.show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

- **RDD 转成 DataFrame** （使 people.txt **非结构化的数据** 也可以运行 **SQL** 操作）
  - 方式一:反射机制推断 RDD 模式，得到 DataFrame(在源码中找到DDToDataFrame项目即可)
  - 方式二:以编程方式定义 RDD 模式，得到 DataFrame(在源码中找到DDToDataFrame项目即可)

  执行

```
#第一步：将代码打包
#第二步：上传代码到 MASTER 的主机 spark 目录下，即 /home/spark/
#第三步：提交 job ，启动 DRIVER 程序
#注 意：最后一个参数为整数类型，其中 1 为方式一将普通 RDD 转成 DataFrame ；其他为方式二
#sudo -u spark spark-submit --class com.kmr.rddToDF.Demo --master yarn-client /home/spark/RDDToDataFrame-1.0-SNAPSHOT.jar "file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/people.txt" "hdfs:///user/spark/out/person" 1
```

- **数据加载**

  DataFrame 提供统一接口加载以下数据 **load()** 方式 ， format() 指定具体数据源格式

  - 结构化数据
  - Parquet
  - JSON
  - Hive 表
  - JDBC 连接外部数据

  shell 下的代码如下：

```
#首先，切入 shell 模式，部署方式为 yarn-client
#cd /home/spark && sudo -u spark spark-shell --master yarn-client
#通过 spark-shell 为例， format() 指定加载格式，默认是 Parquet ，可以通过  spark.sql.sources.default 参数修改
scala> val df = sqlContext.read.format("json").load("file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/people.json")
#通过 save() 进行存盘
df.select("name","age").write.format("parquet").save("/usr/qjjia/namesAndAges.parquet")
*注意：执行#cd /home/spark && sudo -u spark spark-shell --master yarn-client这条命令前请检查当前所在目录*
```

场景一：结构化数据处理

这里分别以 JSON 文件和 Hive 表为例

- JSON文件操作（ people.json --> RDD --> DataFrame --> 注册临时表 --> SQL查询）

  项目名：JSONFile

  项目核心代码（具体代码,您可以点这里，自行下载）：

```
import org.apache.spark.sql.SQLContext
import org.apache.spark.{SparkConf, SparkContext}

object Demo {
    def main(args: Array[String]) {
      val dirIn = args(0) //数据输入PATH

      val conf = new SparkConf().setAppName("JSON")
      // SparkContext 是程序和集群的唯一通道
      val sc = new SparkContext(conf)
      // 通过SparkContext 创建SQLContext
      val sqlContext = new SQLContext(sc)

      val people = sqlContext.read.json(dirIn)

      people.printSchema()

      //注册DataFrame作为一个临时表
      people.registerTempTable("jsonTable")

      //使用SQL 语句操作
    val teenagers = sqlContext.sql("select name from jsonTable where age >=13 and age <= 19")
    teenagers.map(t => "Name:" + t(0)).collect.foreach(println)

    // val anotherRDD = sc.parallelize("""{"name":"spark","address":{"city":"USA","avenue":"SEVEN"}}""" :: Nil)
    // val anotherPeople = sqlContext.read.json(anotherRDD)
    sc.stop()
  }
}
```

  提交 job，同上 **RDDToDataFrame** submit 过程

```
# sudo -u spark spark-submit --class com.kmr.JSON.Demo --master yarn-client /home/spark/JsonFile-1.0-SNAPSHOT.jar "file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/people.json"
```

- Hive 表操作

  项目名：SparkSQL_Hive

  INPUT:/usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/kv1.txt

  格式为：string + 空格 + string （一行）

  核心代码（具体代码,您可以点这里，自行下载）：

```
object Demo  {
    def main(args: Array[String]) {
        val dirIn = args(0)
        val conf = new SparkConf().setAppName("RDDToDF")
        val sc = new SparkContext(conf)

        //通过sc创建HiveContext的实例hiveContext
        val hiveContext = new HiveContext(sc)
       //通过HiveContext的sql命令创建表
       hiveContext.sql("create table if not exists src (key int,value string)")
       //加载数据
       hiveContext.sql("load data local inpath '"+ dirIn +"' into table src")
       //HiveQL 的查询表达
       hiveContext.sql("from src select key,value").collect.foreach(println)
       sc.stop()
    }
}
```

  提交 job ，命令如下：

```
# sudo -u spark spark-submit --class com.kmr.sparkHive.Demo --master yarn-client /home/spark/sparkSQL_Hive-1.0-SNAPSHOT.jar "file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/kv1.txt"
```

场景二：非结构化数据处理

处理非结构化数据的逻辑是先将其转换为结构化数据再进行处理

处理流程为：people.txt --> RDD --> DataFrame -- saveAs Parquet;load Parquet --> 重构 DataFrame --> 注册临时表 --> SQL 查询

这里以上文中提到的people.txt为例

项目名：**ParquetFile**

- 项目核心代码（具体代码,您可以点这里，自行下载）：

```
val people = sqlContext.read.parquet(dirIn)
peopleDF.write.parquet(dirOut)
```

- 提交 job，同上 **RDDToDataFrame** submit 过程

```
# sudo -u spark spark-submit --class com.kmr.parquet.Demo --master yarn-client /home/spark/ParquetFile-1.0-SNAPSHOT.jar "file:///usr/hdp/2.4.0.0-169/spark/examples/src/main/resources/people.txt" "hdfs://
/user/spark/out/person2.parquet"
```

场景三：多源数据整合处理

有时我们就会遇到多种数据源同时存在的情况，此时，通过不同的数据源，构造基于相同的 Schema 的 DataFrame ，进行汇总操作，可用合并操作，这里有一个例子来帮助您理解。

项目名：comprehensive

核心代码（具体代码,您可以点这里，自行下载）：

```
// 使用case 定义类 Log
    case class Log(id:String,info:String)

    object Demo  {
        def main(args: Array[String]) {

            val dirOut1 = args(1)
            val dirOut2 = args(2)
            val conf = new SparkConf().setAppName("RDDToDF")
            val sc = new SparkContext(conf)
            val sqlContext = new SQLContext(sc)
        mix
            import sqlContext.implicits._
            val df1 = sc.parallelize(Array(("id1","info1"),("id2","info2"))).map(l => Log(l._1,l._2)).toDF

            //查看 Schema 架构
            df1.printSchema()

            //df1 文件保存成 Parquet文件
            df1.write.parquet(dirOut1)

            val df2 = sc.parallelize(Array(("id3","info3"),("id4","info4"),("id5","info5"))).map(l =>Log(l._1,l._2)).toDF
            df2.write.parquet(dirOut2)

            //数据源1 进行加载
            val data1 = sqlContext.read.parquet(dirOut1)
            //数据源2 进行进行加载
            val data2 = sqlContext.read.parquet(dirOut2)
            //数据源进行整合
            val data3 = data1.unionAll(data2)
            //注册临时表
            data3.registerTempTable("logs")
            //查询执行
            sqlContext.sql("select * from logs").collect.foreach(println)

            sc.stop()
        }
    }
```

- 提交 job，命令如下：

```
# sudo -u spark spark-submit --class com.kmr.comprehensive.Demo --master yarn-client /home/spark/comprehensive-1.0-SNAPSHOT.jar "hdfs:///user/spark/out/log1.parqute" "hdfs:///user/spark/out/log2.parqute"
```

Spark SQL 命令行操作

本地模式运行 Hive Metastore 服务的一个有效工具，通过命令行接收查询输入

执行代码： `#cd /home/spark && sudo -u spark spark-sql --master yarn-client`

复杂使用场景

Spark SQL 可以与已经存在的关系型数据库相连，实现 Spark 对关系型数据的操作。

以下以一个案例用来讲解 KRDS 、 Spark 、MySQL 如何一同使用

- KRDS 的实例创建，请戳这里
- shell 连接：

```
# sudo -u spark spark-shell --driver-class-path /usr/share/java/mysql-connector-java-5.1.17.jar
```

1. 项目名： **SparkRDS**
2. 场景： 连接传统关系数据库，实现 SPARK 对关系型数据的操作
3. 引入 MySQL 的驱动包：/usr/share/java/mysql-connector-java-5.1.17.jar
4. 已经购买 KRDS 服务，并成功创建 MySQL 实例，并为其配置正确的**白名单**（安全组 IP ）
5. 远程连接 RDS 并创建自己的数据库：
6. mysql 创建表 student：

```
mysql> create table `student`( `id` decimal(50,0) default null,`gender` varchar(20) default null,`name` varchar(20) default null,`age` int default null) engine=InnoDB default charset=utf8;
Query OK, 0 rows affected (0.01 sec)

mysql> describe student;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| id     | decimal(50,0) | YES  |     | NULL    |       |
| gender | varchar(20)   | YES  |     | NULL    |       |
| name   | varchar(20)   | YES  |     | NULL    |       |
| age    | int(11)       | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

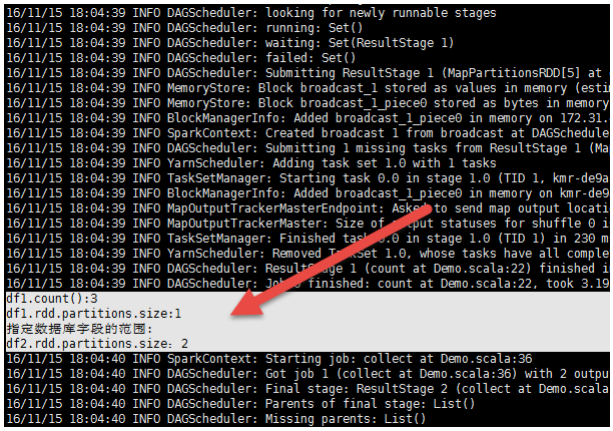
7. 插入一些数据，如下

```
mysql> select * from student;
+------+--------+------+------+
| id   | gender | name | age  |
+------+--------+------+------+
|    0 | male   | Mike |   12 |
|    1 | male   | Ben  |   13 |
|    2 | female | Lily |   12 |
+------+--------+------+------+
3 rows in set (0.00 sec)
```

8. 项目代码点这里下载。
9. 提交job

```
# sudo -u spark spark-submit --driver-class-path /usr/share/java/mysql-connector-java-5.1.17.jar --class com.kmr.sparkRDS.Demo --master yarn-client /home/spark/SparkRDS-1.0-SNAPSHOT.jar
```

10. 控制台查询结果



第三章 Spark 在流式处理中的应用

日常处理数据的过程中，除了离线处理，也有数据实时产生实时处理的情况。为满足实时处理数据的需求，就需要整合数据源（数据生产者）、处理组件以及结果输出这三部分，以达到流式处理（实时处理）的目的。

以下会有一个例子用 Kafka 来实现流式处理，更多的使用方式请参考 SparkStreaming 官方文档

在做流式处理以前，需要引入一个概念 Dstreams（Discretized Streams），这是 Spark 实现流式处理所必需的一种高度抽象的数据结构。详细信息请参考 Apache Spark Streamming。

Kafka 和 Spark 来实现流式处理

- 项目名：**KafkaWordCount**
- 输入源：伪造的数据（KafkaWordCountProducer）
- 模拟 Kafka 过程：
  1. Xshell 远程登录 MASTER 主机
  2. 用 ssh 切换到任意一台核心节的主机上（ CORE 节点），具体步骤如下

```
#查看 CORE 的主机名
#cat /etc/hosts
172.31.*.* kmr-core-1-001.ksc.com kmr-core-1-001
172.31.*.* kmr-core-1-002.ksc.com kmr-core-1-002
172.31.*.* kmr-core-1-003.ksc.com kmr-core-1-003
172.31.*.* kmr-master-1-001.ksc.com kmr-master-1-001
172.31.*.* kmr-master-2-001.ksc.com kmr-master-2-001
# ssh 进入任意 CORE 节点
#ssh kmr-core-1-001.ksc.com
#查看 kafka 所在目录
#ps aux | grep kafka
```

  3. （producer）发送消息，具体执行过程如下：

```
#切入 kafka 工作主目录
# cd /usr/hdp/2.4.0.0-169/kafka/bin
#创建 topic
#./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1  --topic test
#检验 topic 创建是否成功（如果正常返回 test）
#./kafka-topics.sh --list --zookeeper localhost:2181
#打开 producer，发送消息
#./kafka-console-producer.sh --broker-list kmr-core-1-001.ksc.com:6667 --topic test
#####启动成功后，输入以下内容测试
hdfs hdfs spark spark spark storm streaming DF DStream
```

  4. （consumer）接受消息，具体执行如下过程：

```
#切入kafka工作主目录
# cd /usr/hdp/2.4.0.0-169/kafka/bin
#保持 producer 端不动，另起，一个 shell 窗口登入当前 CORE 节点（kmr-core-1-001.ksc.com）
#./kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning
#####启动成功后，如果一切正常将会显示 producer 端输入的内容
```

- 源码路径：
  /usr/hdp/2.4.0.0-169/spark/examples/src/main/scala/org/apache/spark/examples/streaming/KafkaWordCount.scala
- 提交 job ：
  1. 停止运行刚才的 kafka-console-producer 和 kafka-console-consumer
  2. 运行 KafkaWordCountProducer

```
#切换到 MASTER 主机， spark 工作目录下
#cd /usr/hdp/2.4.0.0-169/spark/bin/
# ./run-example org.apache.spark.examples.streaming.KafkaWordCountProducer kmr-core-1-001.ksc.com:6667 test 3 5
```

  3. 运行 KafkaWordCount

```
#保持上一步骤的producer端口，另起，一个 shell 窗口连接 MASTER
#切换到spark工作目录下
#cd /usr/hdp/2.4.0.0-169/spark/bin/
# sudo -u spark  ./run-example org.apache.spark.examples.streaming.KafkaWordCount kmr-core-1-001.ksc.com:2181 test-consumer-group test 1
```

  4. 参数解释：
     1. KafkaWordCountProducer
        1. **KafkaWordCountProducer kmr-core-1-001.ksc.com:6667** 表示 producer 的地址和端口
        2. **test** 表示 topic
        3. **3** 表示每秒发多少条消息
        4. **5** 表示每条消息中有几个单词
     2. KafkaWordCount
        1. **kmr-core-1-001.ksc.com:2181** 表示 zookeeper 的监听地址
        2. **test-consumer-group** 表示 consumer-group 的名称，必须和 $KAFKA_HOME/config/consumer.properties 中的 group.id 的配置内容一致
        3. **test** 表示 topic
        4. **1** 表示线程数。

到此您已经基本掌握了在KMR上使用Spark。

Windows下的SSH密钥管理工具

Windows下的SSH密钥管理工具大致分为两类：

1. 单独的SSH客户端，包括PuTTY、SecureCRT和Xshell；
2. Linux的shell环境，包括Babun、Cmder、Git Bash。

单独的SSH客户端

PuTTY

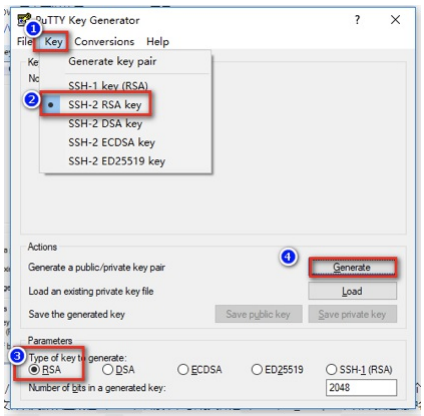
软件下载

下载地址：http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html，下载putty.exe 和puttygen.exe。

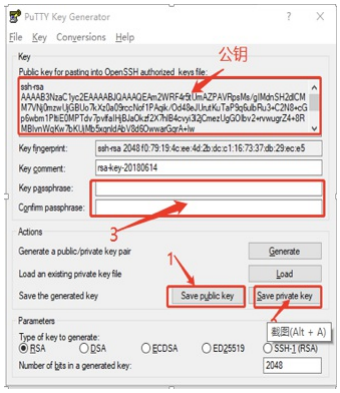
密钥生成

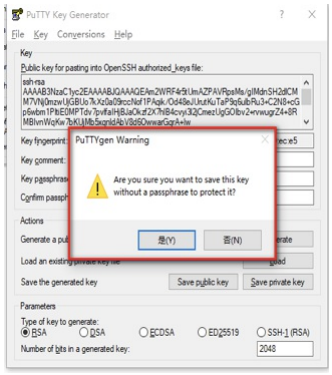
打开PuTTYgen.exe，生成密钥。

1. 注意选择菜单栏中**key**下**SSH-2 RSA key**，在**type of key to generate** 中选择**RSA**，单机**Generate**，进行密钥生成（注意：点击**generate**后，需要鼠标在puttygen.exe中来回移动，才可生成密钥）；



2. 生成密钥完成后，下图上方红框中为公钥，点击下方1，2两处将公钥和私钥进行保存。在图中3处输入密码，进行私钥保护，若不输入密码，会弹出提示框，单击是（Y）继续或者单击否（N）可返回进行密码设置。

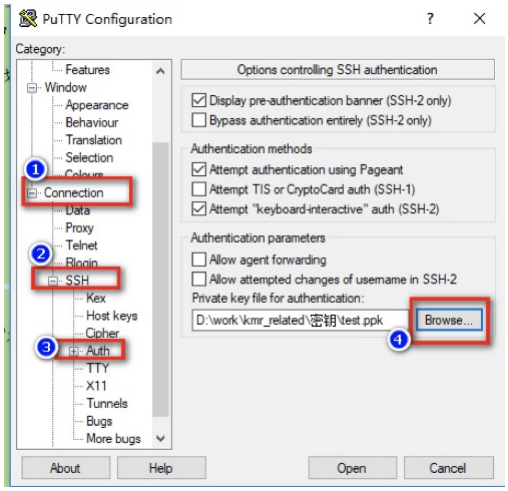




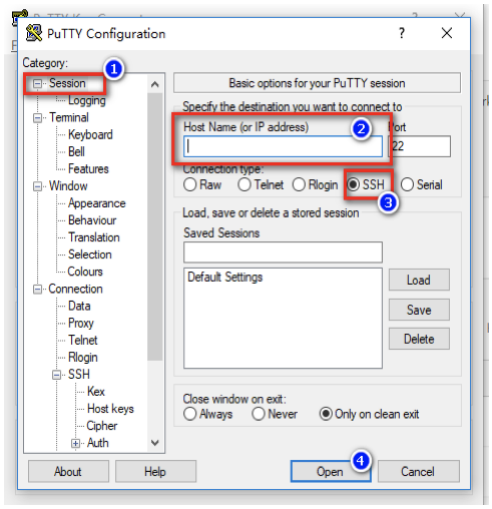
3. 在KMR中添加密钥，打开KMR控制台，选择**SSH密钥管理**，点击**添加密钥**按钮，进入添加密钥界面。

使用SSH访问KMR集群

1. 选择创建密钥时创建好的密钥，点击**加载到集群**按钮，把密钥加载到集群；
2. 在putty.exe中导入创建密钥时产生的私钥。
   - 打开在http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html下载好的putty.exe.
   - 分别点击**Connection**>**SSH**>**Auth**，然后点击右侧的**Browse**选择刚刚所生产的密匙。



   - 单击左侧的**Session**，在右侧**Host Name(or IP address)** 下输入公网IP，注意**Connection type**中选择**SSH**，单机**OPEN**，登陆到Putty控制台；



   - 登陆用户名为root，若为私钥设置了密码保护，输入root用户名后需要输入所设置的密码，注意密码并不会在界面上有占位提示，输入完成按下Enter键即可。

SecureCRT

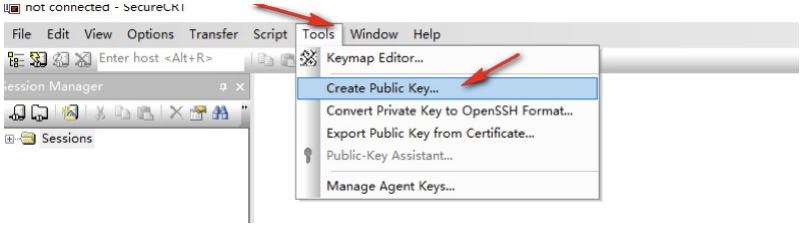
软件下载

下载地址：https://www.vandyke.com/download/securecrt/download.html（需注册）。

密钥生成

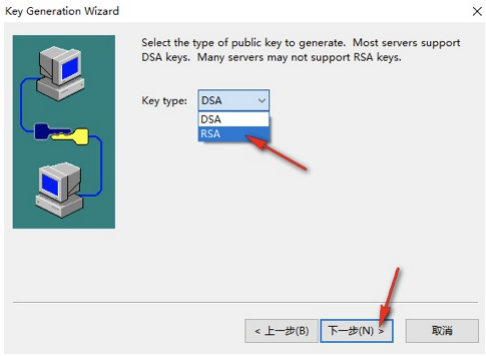
打开Secure CRT，生成密钥。

1. 点击**Tools**>**create public key**。

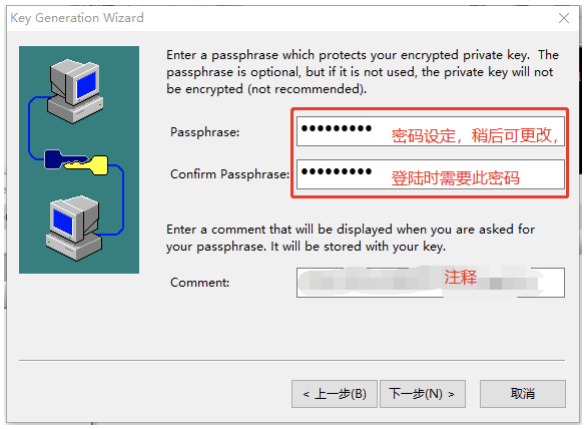


2. 点击**下一步**。

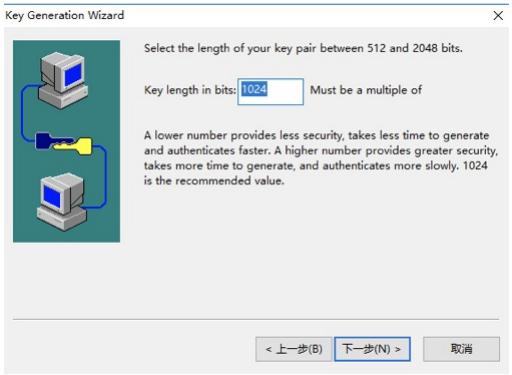
3. Key类型选择**RSA**，点击**下一步**。



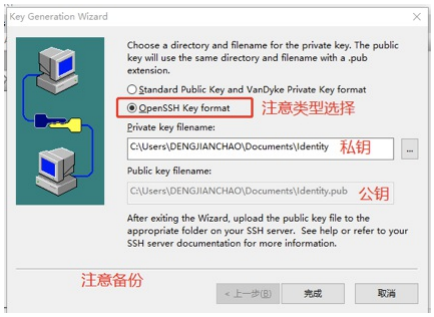
4. 设置好密码和注释，点击**下一步**。



5. 密匙长度，1024，保持默认即可，点击**下一步**后会生成密匙。



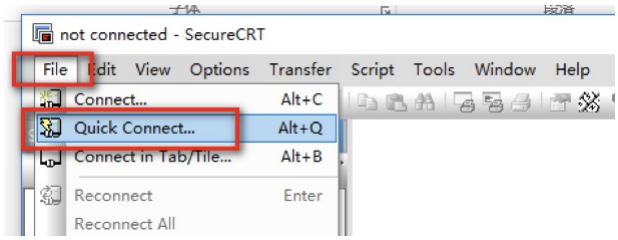
6. 生成密钥后，在点击**下一步**，得到以下页面，注意选择好密钥类型，注意密钥的备份，图中上边一块是私钥，下边是公钥，注意相应备份。



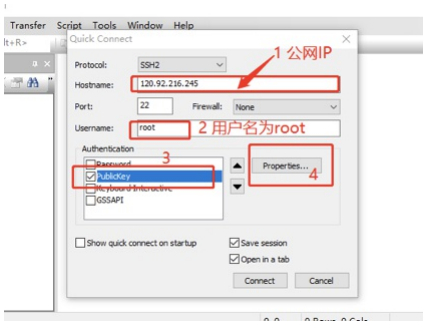
7. 在KMR中添加密钥，打开KMR控制台，选择**SSH密钥管理**，点击**添加密钥**按钮，进入添加密钥界面。

使用SSH访问KMR集群

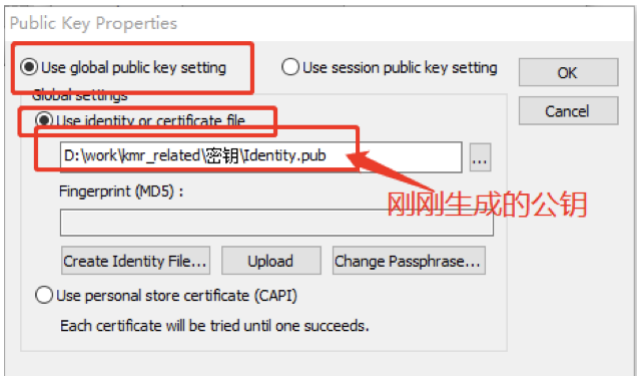
1. 选择创建密钥时创建好的密钥，点击**加载到集群**按钮，把密钥加载到集群；
2. 在SSH工具中导入创建密钥时产生的私钥，下面以Secure CRT为例进行说明。
   - 打开Secure CRT，**File>Quick Connect**。



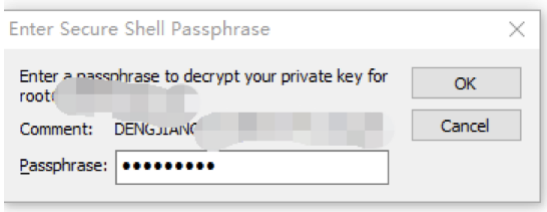
   - 在对话框中各处分别输入公网IP，**Username**处输入root，勾选**PublicKey**，单击**properties**。



   - 将生成的公钥加载进去，在点击**OK**，然后点击**Connect**。



   - 点击**Connect**后，需要输入刚刚设置的密码。



您也可以在Secure CRT中进行颜色设置，设置方法参考该链接内容 https://blog.csdn.net/u010031673/article/details/51130889。

Xshell

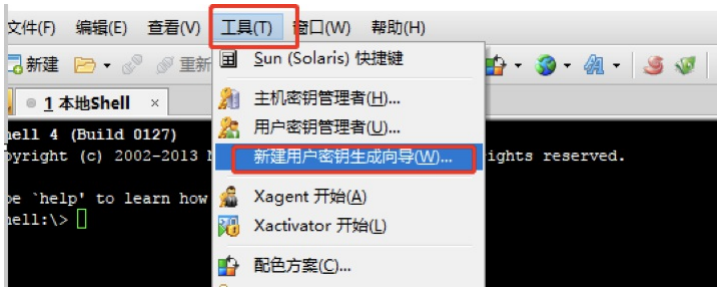
软件下载

下载地址：https://www.netsarang.com/products/xsh_overview.html

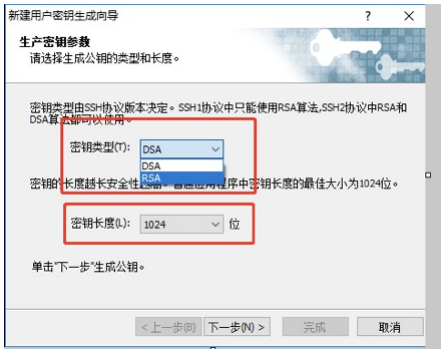
密钥生成

打开Xshell，生成密钥

1. 点击**工具>新建用户新建密钥生成向导（W）**。



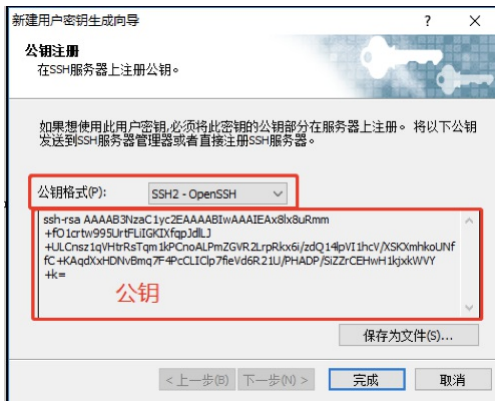
2. 密钥类型选择RSA，长度1024位，可保持不变，单击**下一步**。



3. 等待生成密钥，生成完成后，出现以下界面，单击**下一步**。

4. 输入**密钥名称**和**密码**，单击**下一步**。

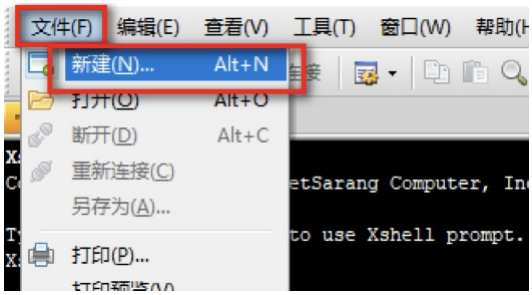
5. 公钥格式选择**SSH2-OpenSSH**，将所生成的公钥进行保存备份。



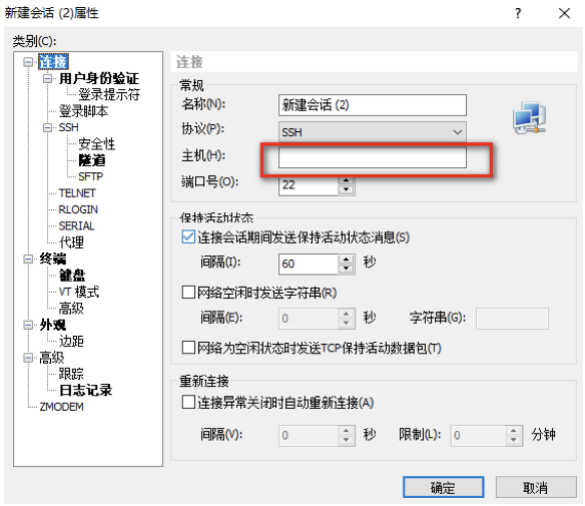
6. 在KMR中添加密钥，打开KMR控制台，选择**SSH密钥管理**，点击**添加密钥**按钮，进入添加密钥界面。

使用SSH访问KMR集群

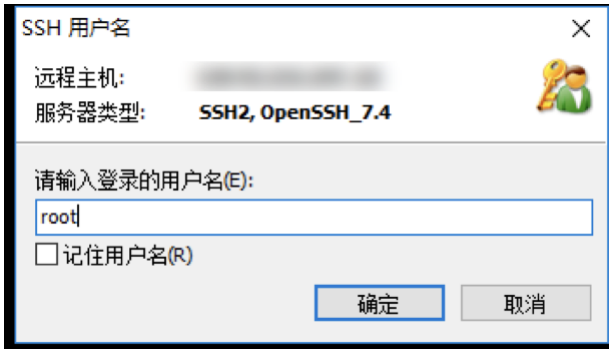
1. 选择创建密钥时创建好的密钥，点击**加载到集群**按钮，把密钥加载到集群。
2. 在SSH工具中导入创建密钥时产生的私钥。
   - 打开Xshell，**文件>新建**



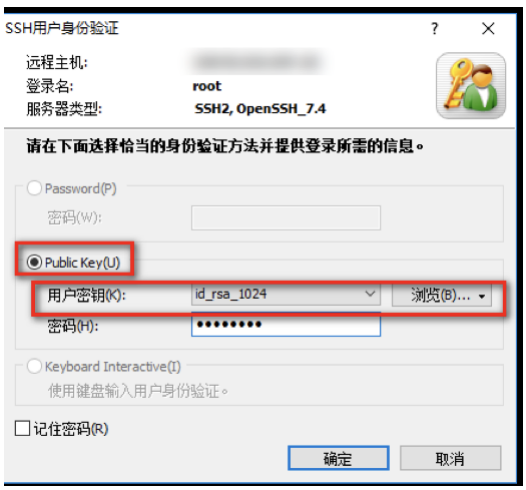
   - 在主机中输入公网IP，设置好**名称**及**端口号**，协议选择**SSH**，单击**确定**。



   - 输入登录用户名root，单击**确定**。



   - 选择PublicKey，加载刚刚生成的公钥，输入**密码**，点击**确定**。



Linux的shell环境

Babun

项目地址：http://babun.github.io/

安装：下载之后，点击解压出来install.bat。

启动：进入cmd.exe中输入babun，回车即可进入Linux环境。

Cmder

项目地址：http://cmder.net/

只需要下载mini版本即可，Full版本包含git for windows。

解压后直接运行Cmder.exe即可进入Linux环境。

Git Bash

项目地址：https://gitforwindows.org/

安装：下载git for windows客户端。

启动：从开始菜单中git目录中启动git bash即可进入Linux环境。

创建密匙

以上三工具安装好，可模拟Linux环境后，后续操作均相同。

公私钥创建

命令：ssh-keygen -t rsa

回车后第一个提示框为指定文件名，第二个和第三个提示框为密码和密码确认。

如果不指定文件名则默认生成在\~/.ssh/目录下，公钥文件是~/.ssh/id_rsa.pub，如果指定用户名则在当前目录生成公钥和私钥。用户使用ssh-keygen时也可以使用-f自己指定公钥目录。完整的参数及含义请参考man ssh-keygen。

生成密钥后，可以使用cat ~/.ssh/id_rsa.pub将公钥输出到terminal中，再复制，也可以使用pbcopy < ~/.ssh/id_rsa.pub命令将公钥字符串直接输出到剪贴板中。 如提示公钥格式不正确，请确认复制的字符串后是否有空行。

注意：将生成的私钥权限改为400！ chmod 400 ~/.ssh/id_rsa

控制台密匙添加

在KMR中添加密钥，打开KMR控制台，选择**SSH密钥管理**，点击**添加密钥**按钮，进入添加密钥界面。

可在命令框中展现公钥内容，再将其粘贴至添加密钥界面的公钥框中。 cat ~/.ssh/id_rsa.pub。

登陆KMR

命令：ssh root@公网IP

hive添加第三方jar包

在**Ambari>hive>config**中的

hive-interactive-env template和hive-env template中搜索HIVE_AUX_JARS_PATH

HIVE_AUX_JARS_PATH=/usr/hdp/current/hive-server2-hive2/lib/hive-hcatalog-core.jar:/home/hive/json-serde-1.3.8-jar-with-dependencies.cdh5.jar

将对应的jar包地址添加到目录中，多个jar包地址用:分隔，jar需要放在hive所在主机对应的本地文件中。

通过内部负载均衡为集群服务提供统一的服务端口

大数据场景下，通常是以集群的形式向外提供服务。在程序代码中通常需要填入集群的所有hosts。

例如，Kakfa的服务列表为：

```
kmr-dd626f75-gn-6de00bfb-core-1-001.ksc.com
kmr-dd626f75-gn-6de00bfb-core-1-002.ksc.com
kmr-dd626f75-gn-6de00bfb-master-1-001.ksc.com
```

ES的服务列表为：

```
kmr-1f56d223-gn-ef4a457b-master-1-001.ksc.com
kmr-1f56d223-gn-ef4a457b-master-2-001.ksc.com
kmr-1f56d223-gn-ef4a457b-master-2-002.ksc.com
```

Zookeeper的服务列表为：

```
kmr-dd626f75-gn-6de00bfb-core-1-002.ksc.com:2181
kmr-dd626f75-gn-6de00bfb-core-1-001.ksc.com:2181
kmr-dd626f75-gn-6de00bfb-master-1-001.ksc.com:2181
```

如果将服务列表硬编码在程序代码中，后续服务集群规模有调整时，都需要对应进行调整，针对这种情况，可以使用内部负载均衡，将实际提供服务的host和端口屏蔽调，只为程序代码暴露负载均衡地址，具体的集群规模调整都在负载均衡中进行。

具体流程如下：

1. 在控制台中创建负载均衡
2. 创建监听器

- 选择正确的VPC；
- 填写要监听的端口，例如zookeeper填写2181，Kafka则为6667；
- 选择后端的真实实例；
- 添加真实服务器响应的端口；

这里列出了两个监听器，一个是监听Zookeeper的请求，一个是监听Kafka的请求。

后续集群规模的调整，只需要在负载均衡的真实服务器添加或删除就可以。在程序中填入的集群服务host不再需要填写集群的全部host而是填写负载均衡的IP地址。

Flume写数据入HDFS Demo

环境要求：

1.Kafka0.11.0版本。 2.Flume1.7版本。3.KMR4.0.1版本。4.KSCKMR安全组开放8020与50010端口。

操作步骤参考：

1.配置flume配置文件，例如，路径为/root/apache-flume-1.7.0-bin/conf/flume-hdfs.conf。

```
# ------------------- define data source -----------------------
# source alias
agent.sources = source_from_kafka
# channels alias
agent.channels = mem_channel
# sink alias
agent.sinks = s3_sink

# define kafka source
agent.sources.source_from_kafka.type = org.apache.flume.source.kafka.KafkaSource
agent.sources.source_from_kafka.batchSize = 10
# set kafka broker address
## 此处注意kafka所在安全组是否对flume主机放行，端口为6667
agent.sources.source_from_kafka.kafka.bootstrap.servers = 10.0.0.139:6667 # 替换为对应的托管Kafka ，并将kafka服务起的/etc/hosts文件 加入flume机器
# set kafka topic
agent.sources.source_from_kafka.kafka.topics = jacktest
# set kafka groupid
agent.sources.source_from_kafka.kafka.consumer.group.id = flumeTest1

# defind hdfs sink
agent.sinks.s3_sink.type = hdfs
# set store hdfs path
## 此处注意KMR所在安全组是否对flume主机放行，端口为8020
agent.sinks.s3_sink.hdfs.path = hdfs://10.0.88.30:8020/tmp  #KMR master1地址，端口为8020

# set file size to trigger roll
agent.sinks.s3_sink.hdfs.rollSize = 0
agent.sinks.s3_sink.hdfs.rollCount = 0
agent.sinks.s3_sink.hdfs.rollInterval = 5
#agent.sinks.s3_sink.hdfs.threadsPoolSize = 30
agent.sinks.s3_sink.hdfs.fileType = DataStream
agent.sinks.s3_sink.hdfs.writeFormat = Text

# define channel from kafka source to hdfs sink
agent.channels.mem_channel.type = memory
# channel store size
agent.channels.mem_channel.capacity = 1000
# transaction size
agent.channels.mem_channel.transactionCapacity = 1000
agent.channels.mem_channel.byteCapacity = 800000
agent.echannels.mem_channel.byteCapacityBufferPercentage = 20
agent.echannels.mem_channel.keep-alive = 60

# specify the channel the sink should use
agent.sources.source_from_kafka.channels = mem_channel
agent.sinks.s3_sink.channel = mem_channel
```

2.将hadoop-auth-2.7.3.2.6.1.0-129.jar、 commons-configuration-1.6.jar、 hadoop-common-2.7.3.2.6.1.0-129.jar、 hadoop-hdfs-2.7.3.2.6.1.0-129.jar、 htrace-core-3.1.0-incubating.jar和 commonsio-2.4.jar拷贝至目录 /root/apache-flume-1.7.0-bin/lib或配置环境变量${HADOOP_HOME}。

```
export HADOOP_HOME=/hadoop/hadoop-2.7.3.2.6.1.0-129
```

3.启动Flume。

```
bin/flume-ng agent --conf conf/ --name agent --conf-file conf/flume-hdfs.conf -Dflume.root.logger=WARN,console
```