BCID开发手册

1. 概述

1.1 关于本手册

开发人员手册提供了样例代码和相关指令的范例，以帮助开发人员、快速熟悉BCID认证并且能够利用BCID认证方式基于CMH大数据平台开发应用。

BCID是一种Hadoop安全认证方法，使用AccessId和SecurityKey认证，类似于AWS和ODPS的用户认证方案。

1.2 概念和术语

本文档包含HDFS、MapReduce、Spark、Hive、Storm和HBase等六个接口部分

组件名称	描述
HDFS	针对大数据的分布式文件系统，提供文件的读写访问
MapReduce	大规模数据集的并行计算，将数据处理任务分布到集群的各个节点
HBase	一个高可靠性、高性能、面向列、可伸缩的分布式存储系统
Hive	hive是基于Hadoop的一个数据仓库工具
Storm	是一个分布式高容错的实时流计算系统。
Spark	Spark是一种类似于MapReduce的通用并行框架

1.3 系统架构

BCID认证原理图如下：

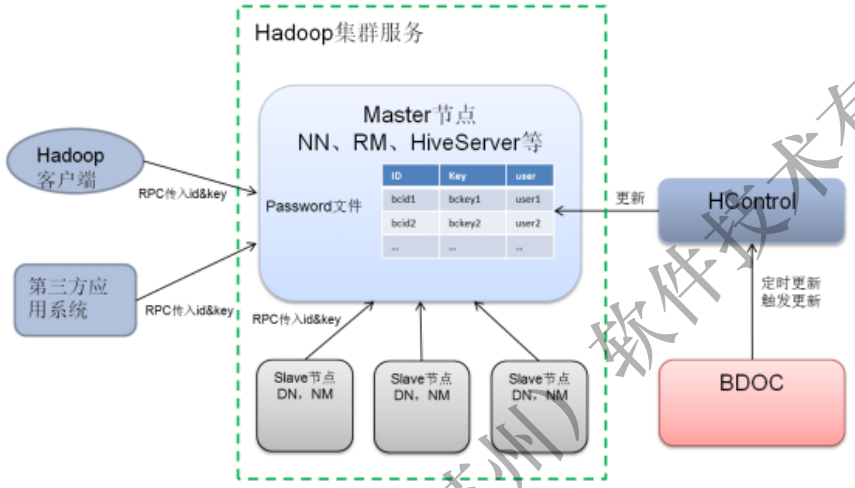

图1 BCID认证原理图

用户的idkey信息在BDOC（大数据运营管理平台）中统一维护，用户及管理员可以通过bdoc前台web页面对用户认证信息进行增删改查的操作。

2. 开发流程

应用开发人员基于CMH+bcid认证开发应用主要有以下步骤：

2.1 配置maven库依赖

开发人员需要使用与bdoc平台内版本一致的苏研套件jar包和客户端依赖。

以套件套件版本bc1.3.2为例，使用的各服务版本信息如下表。

服务名称	版本号
Hadoop	2.6.0-bc1.3.2
Hive	1.1.0-bc1.3.2
Spark	1.5.2-bc1.3.2

Storm	1.0.0-bc1.3.2
HBase	1.1.3-bc1.3.2

2.1.1使用maven构建项目

1、 需要在pom.xml中新增项目的依赖库地址：

```
<repositories>
    <repository>
      <id>archiva.internal</id>
      <name>Internal Release Repository</name>
      <url> http://223.105.0.153:8081/repository/internal/ </url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  <repositories>
```

2、 开发者需要在本地的~/.m2中，创建settings.xml文件，文件内容如下：

```
<settings>
  <servers>
    <server>
      <id>archiva.internal</id>
      <username>xxx</username>
      <password>xxx</password>
    </server>
  </serv.ers>
  <mirrors>
    <mirror>
      <id>archiva.internal</id>
      <name>The CMSS maven internal repo</name>
      <url>http://223.105.0.153:8081/repository/internal</url>
      <mirrorOf>*</mirrorOf>
    </mirror>
  </mirrors>
</settings>
```

其中，mirror中的<url>用来下载项目需要的jar包依赖。

<id>archiva.internal</id>处需要填写开发者maven账号，需要申请，由苏研提供。

2.1.2非maven项目

需要用苏研提供的依赖包添加到项目的构建路径中。

苏研提供各个服务的rpm安装包，执行rpm -ivh
解压后默认路径为/cmss/bch/bc1.3.2/${service_name}（各个服务的安装包内部目录层次参考社区版本），里面会包含各个组件的依赖包，其中：

- Hadoop.相关jar包在$HADOOP_HOME/share中。
- Hbase 相关jar包在$HBASE_HOME/lib中。
- Hive 相关jar包在$HIVE_HOME/lib和$HIVE_HOME/hcatalog/share/中。
- Spark java依赖包在$SPARK_HOME/lib中,python依赖包在$SPARK_HOME/python中。

2.2 设置bcid认证

具体介绍见第三章。

2.3 应用业务逻辑

使用原生接口即可。

2.4 客户端配置

如果需要用客户端连接集群，需要用苏研提供的客户端tar包或者rpm包，解压后放置在本地虚机，再将集群的配置文件拷贝覆盖到本地客户端即可。

例如：hadoop-2.6.0-bc1.3.2_1.x86_64.rpm包

1、执行rpm -ivh hadoop-2.6.0-bc1.3.2_1.x86_64.rpm

会默认安装客户端到/cmss/bch/bc1.3.2/hadoop目录下。

2、配置环境变量

$HADOOP_HOME=/cmss/bch/bc1.3.2/hadoop，

添加$HADOOP_HOME/bin、$HADOOP_HOME/sbin到PATH

3、拷贝集群hadoop的配置文件（core-site.xml.hdfs-site.xml,mapred-site.xml,yarn-site.xml等）到本地/cmss/bch/bc1.3.2/hadoop/etc/hadoop下。

4、配置本地/cmss/bch/bc1.3.2/hadoop/etc/hadoop/hadoop-env.sh 中的$HADOOP_HOME，$JAVA_HOME项

5、本地/etc/hosts中需要添加集群RM,NN所在节点的ip/hostname映射。

这样，本地就可以通过hadoop shell操作集群了（未考虑认证和鉴权限制）。

bcid认证需要将id和key配置在客户端mapred-site.xml中

```
<property>
<name>hadoop.security.bdoc.access.id</name>
<value>***</value>
</property>
<property>
<name>hadoop.security.bdoc.access.key</name>
<value>***</value>
</property>
```


3. 接口

3.1 HDFS接口

HDFS是Apache Hadoop Core项目的一部分。HDFS有着高容错性（fault-tolerant）的特点，并且设计用来部署在低廉的（low-cost）硬件上。而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。

HDFS接口使用方式是：原生的Hadoop HDFS接口 + BCID认证方式。有三种方式使用ID、key的认证：

3.1.1客户端Core.xml中增加ID、key的认证

```
<property>
<name>hadoop.security.bdoc.access.id</name>
<value>***</value>
</property>
<property>
<name>hadoop.security.bdoc.access.key</name>
<value>***</value>
</property>
```

3.1.2命令行参数使用-D

```
bin/hadoop[bdocTokenOptions] command [genericOptions] [commandOptions]
bdocTokenOption are:
-Dhadoop.security.bdoc.access.id=xxxx
-Dhadoop.security.bdoc.access.key=yyyy
```

3.1.3代码中使用

```
{
  //通过Configuration对象把ID、Key传入
Configuration conf = new Configuration ();
conf.set(hadoop.security.bdoc.access.id, accessId);
conf.set(hadoop.security.bdoc.access.key, accessKey);
}
```

3.1.4示例

```
命令行：
bin/hadoop fs -Dhadoop.security.bdoc.access.id=xxxx -Dhadoop.security.bdoc.access.key=yyyy -put
<localsrc> <dst>

代码块：
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration ();
conf.set(hadoop.security.bdoc.access.id, accessId);
conf.set(hadoop.security.bdoc.access.key, accessKey);
FileSystem fs = FileSystem.get(conf);
fs.mkdir(path);
…
}
```

3.1.5实际应用

在集群外应用HDFS例子：

1. 创建一个用户(假设为client用户)，用来执行hadoopclient操作。
2. 解压hadoop-2.6.0-bc1.3.2.tar.gz，得到hadoop-2.6.0-bc1.3.2目录。
   tarzxvf hadoop-2.6.0-bc1.3.2.tar.gz
3. 配置用户变量~/.bashrc，加入Hadoop Home和Java Home的环境变量：

```
export HADOOP_HOME=/home/client/hadoop-2.6.0-bc1.3.2
export HADOOP_CONF_DIR=/home/client/hadoop-2.6.0-bc1.3.2/etc/hadoop
export JAVA_HOME=/home/client/jdk1.7.0_65
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
source ~/.bashrc
```

4. 将集群中任一节点上的/etc/hadoop/conf目录下的core-site.xml, hdfs-site.xml, yarn-site.xml, mapred-site.xml配置文件拷贝到client端的/home/client/hadoop-2.6.0-bc1.3.2/etc/hadoop目录下。
5. 如果client在集群外部，client所在节点的/etc/hosts里面需要配置server端集群的节点的hostname与IP的映射。
6. 如果配置了环境变量，直接就可以使用client命令了，比如要put一个文件到远程的bch集群，可以使用如下命令：

```
hdfs dfs -Dhadoop.security.bdoc.access.id=*** -Dhadoop.security.bdoc.access.key=*** -put srcdist
```

3.2 MapReduce接口

用户开发的MapReduce应用程序要依赖CMH-hadoop的jar包（程序的开发可以选用Intellij、Eclipse等多种IDE），jar包的依赖有如下两种方式，详见如下章节。

3.2.1开发环境

1）下载jar包，直接导入IDE

依赖的hadoop相关jar包，可以从Maven库中下载，链接为http://223.105.0.153:8081/repository/internal/。

2）创建Maven项目，指定Maven库，在pom.xml中新增下面的配置

```
<repositories>
<repository>
<id>archiva.internal</id>
<name>Internal Release Repository</name>
<url> http://223.105.0.153:8081/repository/internal/ </url>
<releases>
<enabled>true</enabled>
</releases>
<snapshots>
<enabled>false</enabled>
</snapshots>
</repository>
<repositories>
```

在pom中配置所依赖的jar包，程序开发后，使用Maven命令编译即可，以样例中的WordCount为例，所依赖的jar包如下：

${CMH-hadoop.version}=2.6.0-bc1.3.2

```
<dependencies>
<dependency>
<groupId>commons-cli</groupId>
<artifactId>commons-cli</artifactId>
<version>1.1</version>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-hdfs</artifactId>
<version>${CMH-hadoop.version}</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-common</artifactId>
<version>${CMH-hadoop.version}</version>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>org.apache.hadoop</groupId>
<artifactId>hadoop-mapreduce-client-app</artifactId>
<version>${CMH-hadoop.version}</version>
<scope>provided</scope>
</dependency>
</dependencies>
```


3.2.2认证参数

使用BDOC管理的Hadoop运行MR程序必须使用如下四个参数

参数	说明
hadoop.security.bdoc.access.id	bdoc用户认证的id, id和key结对使用（必选）
hadoop.security.bdoc.access.key	bdoc用户认证的key, id和key结对使用（必选）
mapreduce.job.queuename	作业提交的bdoc队列（必选）

3.2.3BDOC命令行参数

```
hadoop jar <jar>[mainclass] [bdocOptions] [mainclassOptions]
```

[mainclass]: MR 程序的主类，可选

[bdocOptions]: 必选

-Dhadoop.security.bdoc.access.id　//用户的Access Key ID

-Dhadoop.security.bdoc.access.key　//用户的Secret Access Key

-Dmapreduce.job.queuename //分配给用户的队列名

[mainclassOptions] ：用户自己MR程序的运行参数, 可选

3.2.4命令行示例

```
hadoop jar hadoop-mapreduce-examples.jar pi -Dhadoop.security.bdoc.access.id=***
-Dhadoop.security.bdoc.access.key=*** -Dmapreduce.job.queuename=queuename  2 5
```

3.2.5示例代码

```
public static void main(String [] args) {
//读classpath里面的hadoop configuration
Configuration conf = new Configuration();
// 如果设置上面的-D的参数，会覆盖掉conf中的四个参数（如果有）
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
//后面是正常的mr code
Job job = new Job(conf, "mr app");
Job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
for (inti = 0; i<otherArgs.length - 1; ++i) {
FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
}

FileOutputFormat.setOutputPath(job,
new Path(otherArgs[otherArgs.length - 1]));
system.exit(job.waitForCompletion(true) ? 0 : 1);
}
```


3.3 Spark接口

CMH-spark采用yarn作为cluster manager，spark应用被提交至yarn执行，执行模式为yarn-cluster。

3.3.1认证参数

参数	说明
spark.hadoop.hadoop.security.bdoc.access.id	bdoc用户认证的id, id和key结对使用（必选）
spark.hadoop.hadoop.security.bdoc.access.key	bdoc用户认证的key, id和key结对使用（必选）
queue	作业提交的bdoc队列（必选）

3.3.2 spark-submit

Spark应用使用spark-submit脚本提交，脚本位于$SPARK_HOME/bin目录中，执行时需要传入的参数说明如下（部分）：

参数名称含义

```
--class CLASS_NAME                          主类名称，含包名
--master  yarn-cluster
--conf spark.hadoop.hadoop.security.bdoc.access.id=ID      bdoc用户认证的id
--conf spark.hadoop.hadoop.security.bdoc.access.key=KEY   bdoc用户认证的key
--queue QUEUENAME 作业提交的bdoc队列
--num-executors NUM                          启动的executor数量，默认是2个,仅限于Spark on Yarn模式
--driver-memory MEM                          Driver程序使用内存大小
--executor-memory MEM                        executor内存大小，默认1G
--executor-cores NUM                         每个executor使用的内核数，默认为1，仅限于Spark on Yarn模式
    APPLICATION                              应用jar包
```

以SparkPi为例：

```
export HADOOP_CONF_DIR=xx
spark-submit
--class org.apache.spark.examples.SparkPi
--conf spark.hadoop.hadoop.security.bdoc.access.id=xxx
--conf spark.hadoop.hadoop.security.bdoc.access.key=xxx
--queue=xxx
--master yarn
--num-executors 1
--driver-memory 1g
--executor-memory 1g
--executor-cores 1
${SPARK_HOME}/lib/spark-examples-1.3.0-bc1.3.2-hadoop2.6.0-bc1.3.2.jar 4
```

3.3.3代码调用

scala创建一个SparkContext对象（使用Java语言时创建JavaSparkContext）。

```
var conf = new SparkConf()
.set(“spark.hadoop.hadoop.security.bdoc.access.id”, “xxx”)
.set(“spark.hadoop.hadoop.security.bdoc.access.key”, “xxx”)
.set(“spark.yarn.queue”, “xxx”)
var sc = new SparkContext(conf);
```

3.3.4 配置文件

spark的配置文件、hadoop的配置文件core-site都可以配置id key生效。

Spark默认配置在${SPARK_HOME}/conf/spark-default.conf。

3.3.5 Spark sql

spark sql命令行：

/cmss/bch/bc1.3.4/spark/bin/spark-sql

--conf spark.hadoop.hadoop.security.bdoc.access.id=accessid1

--conf spark.hadoop.hadoop.security.bdoc.access.key=accesskey1

--master yarn

--deploy-mode client

--queue root.test

需要在启动的时候加上相关的bcid参数，并且指定queue

脚本任务：

/cmss/bch/bc1.3.4/spark/bin/spark-sql

--conf spark.hadoop.hadoop.security.bdoc.access.id=accessid1

--conf spark.hadoop.hadoop.security.bdoc.access.key=accesskey1

--master yarn

--deploy-mode client

--queue root.test

-i a.sql

其中，将需要执行的sql语句放入a.sql文件中，可以直接执行。

3.3.6 Spark streaming

由于Spark streaming任务都是通过spark submit提交，因此验证spark-submit即可。

具体见3.3.2

3.4 Hive接口

hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

3.4.1命令行(CLI和beeline)

Hive客户端

```
//登录
bin/hive -hiveconf hadoop.security.bdoc.access.id=accessid1 -hiveconf
hadoop.security.bdoc.access.key=accesskey1  -hiveconf mapreduce.job.queuename=xxx
//使用hive 命令
hive>hive command;

//或者登陆hive shell 后，通过
set adoop.security.bdoc.access.id=accessid1  set hadoop.security.bdoc.access.key=key
```

beeline

```
>beeline -u
"jdbc:hive2://ht-lyf4:10000/default?hadoop.security.bdoc.access.id=2e241e7dcabcc6529af1;hadoop.security.
bdoc.access.key=d75d04abf7c995286118fa6dd77f03259a22d386"
```

3.4.2Hive JDBC 接口

通过JDBC 链接，在做链接的时候需要传入如下参数：

```
jdbc:hive2://localhost:10000/default ? hadoop.security.bdoc.access.key=accesskey1;hadoop.security.bdoc.ac
cess.id=accessid1;mapreduce.job.queuename=root.q_test1
```

示例代码：

```
public static void main(String[] args) throws SQLException {
try {
Connection conn = null;
Class.forName("org.apache.hive.jdbc.HiveDriver");
try {
String url =
"jdbc:hive2://localhost:10000/default ? hadoop.security.bdoc.access.key=accesskey1;hadoop.security.bdoc.ac
cess.id=accessid1;mapreduce.job.queuename=root.q_test1;
conn = DriverManager.getConnection(url, "user", "password");
// other code
} catch (SQLException e) {
e.printStackTrace();
} finally {
if (conn != null) {
conn.close();
}
}
} catch (ClassNotFoundException e) {
e.printStackTrace();
}
```


3.4.3 使用thirft连接hive示例代码

示例代码：

```
public static void main(String[] args) {
   TTransport socket = new TSocket("bdi9.cmss.com", 10000);

   try {
    TTransport transport = PlainSaslHelper.getPlainTransport("hive", "hive", socket);
    TProtocol protocol = new  TBinaryProtocol(transport);
    TCLIService.Client client = new TCLIService.Client(protocol);
    transport.open();
    TOpenSessionReq req = new TOpenSessionReq();
    //bcid认证配置
    Map<String,String> confs = new HashMap<>();
    confs.put("set:hiveconf:hadoop.security.bdoc.access.id", "84f53c1482c22ce34adf");
    confs.put("set:hiveconf:hadoop.security.bdoc.access.key", "66bc5f1e599105592927effe78fd927175d37438");
    confs.put("set:hiveconf:mapreduce.job.queuename", "Testrenter18");
   //要访问的数据库
    confs.put("use:database", "default");
    req.setConfiguration(confs);
    TOpenSessionResp resp = client.OpenSession(req);
    TSessionHandle handle = resp.getSessionHandle();
    TExecuteStatementReq stmtReq = new TExecuteStatementReq(handle, "select * from test");
    TExecuteStatementResp stmtResp = client.ExecuteStatement(stmtReq);
    TOperationHandle opHandler = stmtResp.getOperationHandle();
    TFetchResultsReq fetchReq = new TFetchResultsReq(opHandler, TFetchOrientation.FETCH_NEXT, 100);
    TFetchResultsResp fetchResp = client.FetchResults(fetchReq);
    TRowSet rowSet = fetchResp.getResults();
    List<TColumn> columns = rowSet.getColumns();
    TI32Column values = (TI32Column) columns.get(0).getFieldValue();
    for(int i=0; i < values.getValuesSize();i++) {
      System.out.println(values.getValues().get(i));
    }
    TCloseOperationReq closeReq = new TCloseOperationReq();
    closeReq.setOperationHandle(opHandler);
    client.CloseOperation(closeReq);
    TCloseSessionReq closeConnectionReq = new TCloseSessionReq(handle);
    client.CloseSession(closeConnectionReq);

    transport.close();

   } catch (TException | SaslException e) {
    e.printStackTrace();
   }
  }
```


3.4.4Hive代理用户bcid认证

bcid认证模式下，支持客户端通过proxyUser方式访问集群，示例代码如下：

```
public class TestProxy {
private static Configuration getBdocConf(String id, String key)
throws IOException {
Configuration bdocConf = new Configuration();

bdocConf.addResource(new FileInputStream("/home/luodi/proxy_conf/core-site.xml"));
bdocConf.addResource(new FileInputStream("/home/luodi/proxy_conf/hdfs-site.xml"));
bdocConf.addResource(new FileInputStream("/home/luodi/proxy_conf/yarn-site.xml"));
bdocConf.addResource(new FileInputStream("/home/luodi/proxy_conf/mapred-site.xml"));

bdocConf.set("hadoop.security.authentication", "simple");
bdocConf.set("hadoop.security.bdoc.access.id", id);
bdocConf.set("hadoop.security.bdoc.access.key", key);

return bdocConf;
}

public static void main(String[] args) throws IOException,
InterruptedException {
try {
final Configuration conf = getBdocConf("accessid1", "accesskey1");
UserGroupInformation superUser = UserGroupInformation
.getCurrentUser(conf);
UserGroupInformation proxyUgi = UserGroupInformation.createProxyUser(
"proxyUser", superUser);
// set job queue
conf.set("mapreduce.job.queuename", "root.queue1");
proxyUgi.doAs(new PrivilegedExceptionAction<Void>() {
@Override
public Void run() throws Exception {
// access HDFS with proxyUgi
FileSystem fs = FileSystem.get(conf);
fs.mkdirs(new Path("/proxyUserDir"));

// submit mr job with proxyUgi
Job job = new Job(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path("/wc_input"));
FileOutputFormat.setOutputPath(job, new Path("/wc_output"));
System.exit(job.waitForCompletion(true) ? 0 : 1);
return null;
}
});
} catch (Exception e) {
throw new RuntimeException(e);
}
}
}
```

注意要点：
1，superUser需要有代理proxyUser访问的权限，该权限在服务端配置。

2，客户端需要依赖苏研版本的hadoop相关jar包
3，客户端conf中需要添加入superUser的id&key
4，客户端conf中配置的认证方式（hadoop.security.authentication）需要为simple（默认值）

3.5 Storm接口

Storm是一个免费开源、分布式、高容错的实时计算系统。Storm令持续不断的流计算变得容易，弥补了Hadoop批处理所不能满足的实时要求。Storm经常用于在实时分析、在线机器学习、持续计算、分布式远程调用和ETL等领域。Storm的部署管理非常简单，而且，在同类的流式计算工具，Storm的性能也是非常出众的。

3.5.1 开发环境

Maven设置与2.2.1中相似，依赖的为bdoc的storm：

```
<dependency>
   <groupId>org.apache.storm</groupId>
   <artifactId>storm-core</artifactId>
   <version>1.0.0-bc1.3.2</version>
</dependency>
```

3.5.2 代码中使用

首先，开发Spout：

1）定义输出字段

```
 @Override
public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer)
{
 outputFieldsDeclarer.declare(new Fields("score","key"));
}
```

2）发送数据

```
@Override
public void nextTuple(){
   collector.emit(new Values("1.0","for"));
   Utils.sleep(2000);
}
```

然后，开发处理组件bolt：在上述例子中，只是发送了固定的数据，可根据实际情况获取发送数据。

```
@Override
public void execute(Tuple tuple){
   try{
   //Serialize the activity object to a JSON string
   logger.info("Printing Tuple ("+ ++counter + "): ");
   logger.info(objectMapper.writeValueAsString(tuple));
   }catch(JsonProcessingException ex){
   }
}
```

然后，使用TopologyBuilder定义拓扑：上述bolt只是将流中的数据打印，可根据实际情况替换逻辑。

```
TopologyBuilder builder=new TopologyBuilder();
builder.setSpout("mySpout",new TestSpout(),1);
builder.setBolt("myBlot",new TestBolt,1).shuffleGrouping("mySpout");
Topology topology=builder.createTopology();
```

其中，“accessid1”和“accesskey1”可根据实际bcid鉴权信息修改。最后，使用StormSubmitter提交拓扑：

```
Config conf=new Config();
conf.setTopologyWorkerMaxHeapSize(500);
conf.setDebug(true);
SubmitOptions submitOptions=new SubmitOptions();
Map<String,String> creds=new HashMap<String,String>();
creds.put("accessid1","accesskey1");
submitOptions.set_creds(new Credentials(creds));
submitOptions.set_initial_status(TopologyInitialStaus.ACTIVE);
StormSubmitter.submitTopology("topo3",conf,topology,submitOptions);
```

编译后即可通过storm命令行来提交拓扑。

3.6 HBase接口

HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可在廉价PC Server上搭建起大规模结构化存储集群。

3.6.1 HBase Shell

进入到HBase 目录下面,执行如下命令，进入HBase Shell

开启BCID，需要在hbase-site.xml加入如下属性：

```
<property>
    <name>hadoop.security.bdoc.access.id</name>
    <value>bdoc-user-access-id</value>
</property>
<property>
    <name>hadoop.security.bdoc.access.key</name>
    <value>bdoc-user-secret-access-key</value>
</property>
```

然后客户端执行

```
./bin/hbase shell
```

3.6.2 Java开发环境

```
<dependencies>
    <dependency>
      <groupId>org.apache.hbase</groupId>
      <artifactId>hbase-client</artifactId>
      <version>1.1.3-bc1.3.2</version>
    </dependency>
  </dependencies>
  <repositories>
    <repository>
      <id>cmss repo</id>
      <name>CMSS Maven Repository</name>
      <url>http://10.254.2.95:8083/repository/internal/</url>
    </repository>
  </repositories>
</dependencies>
```

3.6.3 HBase代码示例

1) 初始化配置

这一步需要配置bcid认证信息

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;

public Configuration initConfiguration() {
    Configuration conf = HBaseConfiguration.create();
   //加入bcid认证信息
    conf.set("hadoop.security.bdoc.access.id", "<bdoc-user-access-id>");
    conf.set("hadoop.security.bdoc.access.key", "<bdoc-user-secret-access-key>");
    return conf;
}
```

2) 初始化连接

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;

public Connection initConnection() {
    Configuration conf = initConfiguration();
    Connection conn = ConnectionFactory.createConnection(conf);
    return conn;
}
```

然后就可以通过bcid认证对hbase表操作，比如获取Table：

```
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.Table;

public Table getTable(String myTableName) {
    Connection conn = initConnection();
    TableName tableName = TableName.valueOf(myTableName);
    Table table = conn.getTable(tableName);
    return table;
}
```

3.6.4 Phoenix连Hbase

Phoenix不显式支持bcid传参。但实际情况中，取决于Phoenix加载的集群core-site.xml，phoenix从core-site读到的id key，就是它作为hbase client发起rpc时用到的id key。这种情况下是支持bcid认证的。

3.6.5 Hbase Thrift

前提：客户端需要安装thrift环境，如果用python则需要需要安装Python lib for hbase。

CMH-1.3.5版本以上。

Python Thrift示例

```
#!/usr/bin/python
import sys
import time
import os

from thrift.transport import TTransport
from thrift.transport import TSocket
from thrift.protocol import TBinaryProtocol
//注意开启BCID，客户端使用的是THBaseService4CMH 而不是THBaseService
from hbase import THBaseService4CMH
from hbase.ttypes import *

transport = TSocket.TSocket('localhost', 9090);
transport = TTransport.TBufferedTransport(transport)
protocol = TBinaryProtocol.TBinaryProtocol(transport);
client = THBaseService4CMH.Client(protocol)
transport.open()

print 'first,put a row to example table'

tableName = 'example'
rowKey = 'row1'
coulumnValue1 = TColumnValue('info','name','seven')
coulumnValue2 = TColumnValue('info','dept','bigdata')
coulumnValues = [coulumnValue1,coulumnValue2]
tPut = TPut(rowKey,coulumnValues)
client.put(tableName,tPut,'accessid1','accesskey1')

print 'second,get the row from the example table '
get = TGet()
get.row=rowKey
result = client.exists(tableName, get,'accessid1','accesskey1')
print 'result is ', result
```


Java Thrift2 示例

*注意开启BCID，客户端使用的是THBaseService4CMH 而不是THBaseService

```
import org.apache.hadoop.hbase.thrift2.generated.TColumnValue;
import org.apache.hadoop.hbase.thrift2.generated.THBaseService;
import org.apache.hadoop.hbase.thrift2.generated.THBaseService4CMH;
import org.apache.hadoop.hbase.thrift2.generated.TPut;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.protocol.TProtocol;
import org.apache.thrift.transport.TSocket;
import java.nio.ByteBuffer;

public class HBaseThrift2Demo {

  /**
   *  首先建表：create 'example',{NAME => 'info', VERSIONS => 1}
   *
   * 注意开启BCID，客户端使用的是THBaseService4CMH 而不是THBaseService
   */
  public void test() throws TException {
    TSocket tSocket = new TSocket("thriftServer2Host", 9090);
    TProtocol protocol = new TBinaryProtocol(tSocket);

    THBaseService4CMH.Iface bcidClient = new THBaseService4CMH.Client(protocol);

    tSocket.open();
    String tableName = "example";
    TPut put = new TPut();
    put.setRow(Bytes.toBytes("row1"));
    TColumnValue columnValue1 = new TColumnValue();
    columnValue1.setFamily(Bytes.toBytes("info"));
    columnValue1.setQualifier(Bytes.toBytes("partment"));
    columnValue1.setValue(Bytes.toBytes("bigdata"));

    TColumnValue columnValue2 = new TColumnValue();
    columnValue2.setFamily(Bytes.toBytes("info"));
    columnValue2.setQualifier(Bytes.toBytes("name"));
    columnValue2.setValue(Bytes.toBytes("seven"));
    put.addToColumnValues(columnValue1);
    put.addToColumnValues(columnValue2);

    bcidClient.put(ByteBuffer.wrap(Bytes.toBytes(tableName)), put,
ByteBuffer.wrap(Bytes.toBytes("accessid1")),
            ByteBuffer.wrap(Bytes.toBytes("'accesskey1")));
  }
}
```


3.7 Sqoop2

在运行sqoop任务时，sqoop server
充当hadoop的一个client作用，所以hadoop的相关jar包和配置文件（core-site.xml、mapred-site.xml..）必须可用。

1、在core-site.xml中配置bcid的id、key属性：

```
<property>
<name>hadoop.security.bdoc.access.id</name>
<value>***</value>
</property>
<property>
<name>hadoop.security.bdoc.access.key</name>
<value>***</value>
</property>

<property>
  <name>hadoop.proxyuser.sqoop2.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.sqoop2.groups</name>
  <value>*</value>
</property>
```

2、修改conf/sqoop.properties

添加如下：

```
hadoop.security.bdoc.access.id=***
hadoop.security.bdoc.access.key=***
org.apache.sqoop.submission.engine.mapreduce.configuration.directory=/path/to/hadoop/conf
```

然后就可以使用sqoop通过bcid认证。

4. 联系我们

姓名	邮箱

5. 附件

无