# Hortonworks Data Platform

## User Guides

(Oct 28, 2014)

docs.hortonworks.com

# Hortonworks Data Platform : User Guides



The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, Zookeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

# Table of Contents

# List of Tables

# 1. HBase Import Tools

HBase includes several methods of loading data into tables. Various methods exist for loading data from relational format into non-relational format.

The most straightforward method is to either use the `TableOutputFormat` class from a MapReduce job, or use the normal client APIs; however, these are not always the most efficient methods because these APIs cannot handle bulk loading.

Bulk Importing bypasses the HBase API and writes contents, which are properly formatted as HBase data files – HFiles, directly to the file system. Analyzing HBase data with MapReduce requires custom coding.

Using bulk load will use less CPU and network resources than simply using the HBase API. `ImportTsv` is a custom MapReduce application that will load data in Tab Separated Value TSV format into HBase.

The following discusses typical use cases for bulk loading data into HBase:

- HBase can act as ETL data sink
- HBase can be used as data source

Bulk load workflows generate HFiles offline and have two distinct stages:

- Use either `ImportTsv` or `import` utilities or write a custom application to generate HFiles from Hive/Pig.
- Use `completebulkload` to load the HFiles to HDFS

### Note

By default, the bulk loader class `ImportTsv` in HBase imports a tab separated files.

## 1.1. Using Pig to Bulk Load Data Into HBase

Use the following instructions to bulk load data into HBase using Pig:

1. Prepare the input file.

   For example, consider the sample `data.tsv` file as shown below:

```
row1 c1 c2
row2 c1 c2
row3 c1 c2
row4 c1 c2
row5 c1 c2
row6 c1 c2
row7 c1 c2
row8 c1 c2
row9 c1 c2
row10    c1 c2
```

2. Make the data available on the cluster. Execute the following command on your HBase Server machine:

```
hadoop fs -put $filename /tmp/
```

Using the previous example:

```
hadoop fs -put data.tsv /tmp/
```

3. Create or register the HBase table in HCatalog. Execute the following command on your HBase Server machine:

```
hcat -f $HBase_Table_Name
```

For example, for a sample `simple.ddl` table as shown below:

```
CREATE TABLE
simple_hcat_load_table (id STRING, c1 STRING, c2 STRING)
STORED BY 'org.apache.hcatalog.hbase.HBaseHCatStorageHandler'
TBLPROPERTIES (
  'hbase.table.name' = 'simple_hcat_load_table',
  'hbase.columns.mapping' = 'd:c1,d:c2',
  'hcat.hbase.output.bulkMode' = 'true'
);
```

Execute the following command:

```
hcat -f simple.ddl
```

4. Create the import file. For example, create a file named `simple.bulkload.pig` with the following contents:

**Note**

This import file uses the `data.tsv` file and `simple.ddl` table created previously. Ensure that you modify the contents of this file according to your environment.

```
A = LOAD 'hdfs:///tmp/data.tsv' USING PigStorage('\t') AS (id:chararray,
 c1:chararray, c2:chararray);
-- DUMP A;
STORE A INTO 'simple_hcat_load_table' USING org.apache.hive.hcatalog.pig.
HCatStorer();
```

5. Use Pig to populate the HBase table via HCatalog bulkload.

Continuing with the previous example, execute the following command on your HBase Server machine:

```
pig -useHCatalog simple.bulkload.pig
```

# 2. Using Apache Storm

The exponential increase in realtime data from sources such as machine sensors creates a need for data processing systems that can ingest this data, process it, and respond in real time. A typical use case involves an automated system that might respond to machine sensor data by sending email to support staff or placing an advertisement on a consumer's smart phone. Apache Storm enables such data-driven and automated activity by providing a realtime, scalable, and distributed solution for streaming data. Apache Storm can be used with any programming language and guarantees that data streams are processed without data loss. Storm is datatype-agnostic; it processes data streams of any data type.

A complete introduction to the Storm API is beyond the scope of this documentation. However, the next section, Basic Storm Concepts, provides a brief overview of the most essential concepts and a link to the javadoc API. Experienced Storm developers may want to skip to the following sections, Ingesting Data with the KafkaSpout Storm Connector and Writing Data to HDFS and HBase with Storm Connectors, to learn about the group of connectors provided by Hortonworks that facilitate ingesting and writing streaming data directly to HDFS and HBase. Managing Storm Topologies introduces readers to using the Storm GUI to manage topologies for a cluster. Finally, Running the RollingTopWords Topology shows the source code for a sample application included with the `storm-starter.jar`.

### Tip

See the Storm documentation at the Storm incubator site for a more thorough discussion of Apache Storm concepts.

## 2.1. Basic Storm Concepts

Writing Storm applications requires an understanding of the following basic concepts:

**Table 2.1. Storm Concepts**

| Storm Concept | Description |
|---|---|
| Tuple | A named list of values of any data type. The native data structure used by Storm. |
| Stream | An unbounded sequence of tuples. |
| Spout | Generates a stream from a realtime data source. |
| Bolt | Contains data processing, persistence, and messaging alert logic. Can also emit |

| Storm Concept | Description |
|---|---|
| | tuples for downstream bolts. |
| Stream Grouping | Controls the routing of tuples to bolts for processing. |
| Topology | A group of spouts and bolts wired together into a workflow. A Storm application. |
| Processing Reliability | Storm guarantee about the delivery of tuples in a topology. |
| Worker | A Storm process. A worker may run one or more executors. |
| Executor | A Storm thread launched by a Storm worker. An executor may run one or more tasks. |
| Task | A Storm job from a spout or bolt. |
| Process Controller | Monitors and restarts failed Storm processes. Examples include `supervisord`, `monit`, and `daemontools`. |
| Master/ Nimbus Node | The host in a multi-node Storm cluster that runs a process controller, such as `supervisord`, and the Storm nimbus, ui, and other related daemons. The process controller is responsible for restarting failed process |

| Storm Concept | Description |
|---|---|
| | controller daemons, such as `supervisor`, on slave nodes. The Storm nimbus daemon is responsible for monitoring the Storm cluster and assigning tasks to slave nodes for execution. |
| Slave Node | A host in a multi-node Storm cluster that runs a process controller daemon, such as `supervisor`, as well as the worker processes that run Storm topologies. The process controller daemon is responsible for restarting failed worker processes. |

### Spout

All spouts must implement the `backtype.storm.topology.IRichSpout` interface from the storm core API. `BaseRichSpout` is the most basic implementation, but there are several others, including `ClojureSpout`, `DRPCSpout`, and `FeederSpout`. In addition, Hortonworks provides a Kafka Spout to ingest data from a Kafka cluster.

The following example, `RandomSentenceSpout`, is included with the `storm-starter` connector installed with Storm at `/usr/lib/storm/contrib/storm-starter`.

```
package storm.starter.spout;

import backtype.storm.spout.SpoutOutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseRichSpout;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Values;
import backtype.storm.utils.Utils;

import java.util.Map;
import java.util.Random;
```

```
public class RandomSentenceSpout extends BaseRichSpout {
  SpoutOutputCollector _collector;
  Random _rand;


  @Override
  public void open(Map conf, TopologyContext context, SpoutOutputCollector
 collector) {
    _collector = collector;
    _rand = new Random();
  }

  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{ "the cow jumped over the moon", "an
 apple a day keeps the doctor away",
        "four score and seven years ago", "snow white and the seven dwarfs",
 "i am at two with nature" };
    String sentence = sentences[_rand.nextInt(sentences.length)];
    _collector.emit(new Values(sentence));
  }

  @Override
  public void ack(Object id) {
  }

  @Override
  public void fail(Object id) {
  }

  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }

}
```

**Bolt**

All bolts must implement the `IRichBolt` interface. `BaseRichBolt` is the most basic implementation , but there are several others, including `BatchBoltExecutor`, `ClojureBolt`, and `JoinResult`.

The following example, `TotalRankingsBolt.java`, is included with `storm-starter` and installed with Storm at `/usr/lib/storm/contrib/storm-starter`.

```
package storm.starter.bolt;

import backtype.storm.tuple.Tuple;
import org.apache.log4j.Logger;
import storm.starter.tools.Rankings;

/**
 * This bolt merges incoming {@link Rankings}.
 * <p/>
 * It can be used to merge intermediate rankings generated by {@link
 IntermediateRankingsBolt} into a final,
 * consolidated ranking. To do so, configure this bolt with a globalGrouping
 on {@link IntermediateRankingsBolt}.
```

```
 */
public final class TotalRankingsBolt extends AbstractRankerBolt {

  private static final long serialVersionUID = -8447525895532302198L;
  private static final Logger LOG = Logger.getLogger(TotalRankingsBolt.class);

  public TotalRankingsBolt() {
    super();
  }

  public TotalRankingsBolt(int topN) {
    super(topN);
  }

  public TotalRankingsBolt(int topN, int emitFrequencyInSeconds) {
    super(topN, emitFrequencyInSeconds);
  }

  @Override
  void updateRankingsWithTuple(Tuple tuple) {
    Rankings rankingsToBeMerged = (Rankings) tuple.getValue(0);
    super.getRankings().updateWith(rankingsToBeMerged);
    super.getRankings().pruneZeroCounts();
  }

  @Override
  Logger getLogger() {
    return LOG;
  }

}
```

### Stream Grouping

Stream grouping allows Storm developers to control how tuples are routed to bolts in a workflow. The following table describes the stream groupings available.

#### Table 2.2. Stream Groupings

| Stream Grouping | Description |
|---|---|
| Shuffle | Sends tuples to bolts in random, round robin sequence. Use for atomic operations, such as math. |
| Fields | Sends tuples to a bolt based on one or more fields in the tuple. Use to segment an incoming stream and to count tuples of a specified type. |
| All | Sends a single copy of each |

| Stream Grouping | Description |
|---|---|
| | tuple to all instances of a receiving bolt. Use to send a signal, such as clear cache or refresh state, to all bolts. |
| Custom | Customized processing sequence. Use to get maximum flexibility of topology processing based on factors such as data types, load, and seasonality. |
| Direct | Source decides which bolt receives a tuple. |
| Global | Sends tuples generated by all instances of a source to a single target instance. Use for global counting operations. |

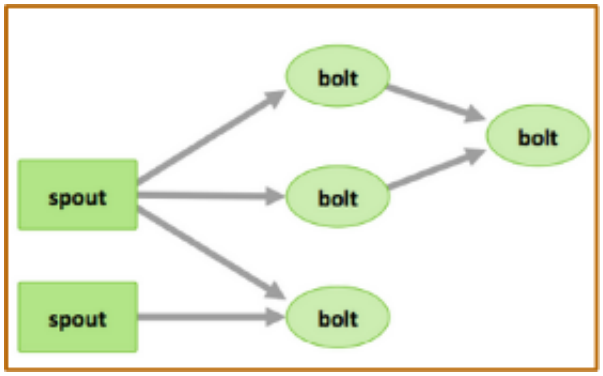
Storm developers specify the field grouping for each bolt using methods on the `TopologyBuilder.BoltGetter` inner class, as shown in the following excerpt from the the `WordCountTopology.java` example included with `storm-starter`.

```
TopologyBuilder builder = new TopologyBuilder();

    builder.setSpout("spout", new RandomSentenceSpout(), 5);

    builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
    builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new
 Fields("word"));
```

The first bolt uses shuffle grouping to split random sentences generated with the `RandomSentenceSpout`. The second bolt uses fields grouping to segment and perform a count of individual words in the sentences.

### Topology

The following image depicts a Storm topology with a simple workflow.




The `TopologyBuilder` class is the starting point for quickly writing Storm topologies with the `storm-core` API. The class contains getter and setter methods for the spouts and bolts that comprise the streaming data workflow, as shown in the following sample code.

```
...
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout1", new BaseRichSpout());
builder.setSpout("spout2", new BaseRichSpout());
builder.setBolt("bolt1", new BaseBasicBolt());
builder.setBolt("bolt2", new BaseBasicBolt());
builder.setBolt("bolt3", new BaseBasicBolt());
...
```

**Processing Guarantees**

Storm provides two types of guarantee about the processing of tuples for a Storm topology.

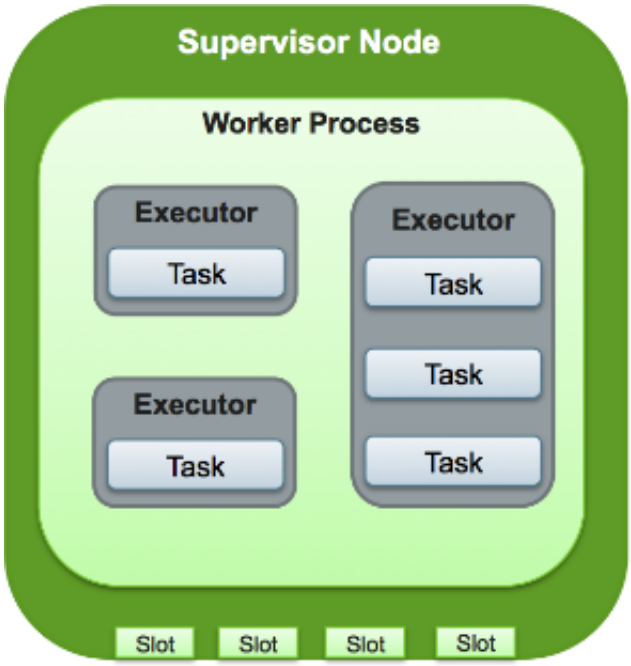
### Table 2.3. Processing Guarantee

| Guarantee | Description |
|---|---|
| At least once | Reliable; Tuples are processed at least once, but may be processed more than once. Use when subsecond latency is required and for unordered idempotent operations. |
| Exactly once | Reliable; Tuples are processed only once. Requires the use of a Trident spout and the Trident API. |

**Workers, Executors, and Tasks**

Apache Storm processes, called *workers*, run on predefined ports on the machine hosting Storm. Each worker process may run one or more *executors*, or threads, where each

executor is a thread spawned by the worker process. Each executor runs one or more *tasks* from the same component, where a component is a spout or bolt from a topology.




See the Storm javadocs at http://storm.incubator.apache.org/apidocs/ for more information.

## 2.2. Ingesting Data with the Apache Kafka Spout

Apache Kafka is a high-throughput distributed messaging system. Hortonworks provides a Kafka spout to facilitate ingesting data from Kafka 0.8x brokers. Storm developers should include downstream bolts in their topologies to process data ingested with the Kafka spout. The `storm-kafka` components include a standard storm spout, as well as fully transactional Trident spout implementations.

The `storm-kafka` spout provides the following key features:

- Supports 'exactly once' tuple processing
- Supports the Trident API
- Supports dynamic discovery of Kafka brokers

```
SpoutConfig spoutConfig = new SpoutConfig(ImmutableList.of("kafkahost1",
"kafkahost2"), //List of Kafka brokers
   8,             // Number of partitions per Kafka host
   "clicks",      // Kafka topic to read from
   "/kafkastorm", // Root path in Zookeeper for the spout to store consumer
 offsets
   "discovery");  // ID for storing consumer offsets in Zookeeper
KafkaSpout kafkaSpout = new KafkaSpout(spoutConfig);
```

The Kafka spout stores its offsets in the same instance of Zookeeper used by Apache Storm. The Kafka spout use these offsets to replay tuples in the event of a downstream failure or timeout.

```
...
spoutConfig.forceStartOffsetTime($TIMESTAMP);
...
```

Kafka chooses the latest offset written around the specified timestamp. A value of `-1` forces the Kafka spout to restart from the latest offset. A value of `-2` forces the spout to restart from the earliest offset.

**Limitations**

The current version of the Kafka spout contains the following limitations:

- Does not support Kafka 0.7x brokers.
- Cannot dynamically discover Kafka partitions, topics, and hosts
- Storm developers must include `${STORM_HOME}/lib/*` in the CLASSPATH environment variable from the command line when running `kafka-topology` in local mode. Otherwise, developers will likely receive a `java.lang.NoClassDefFoundError` exception.

  ```
  java -cp "/usr/lib/storm/contrib/storm-kafka-example-0.9.1.2.1.1.0-320-jar-with-dependencies.jar: /usr/lib/storm/lib/*" org.apache.storm.kafka.TestKafkaTopology <zookeeper_host>
  ```

- Secure Hadoop clusters must comment out the following statement from `${STORM_HOME}/bin/kafka-server-start.sh`:

  ```
  EXTRA_ARGS="-name kafkaServer -loggc"
  ```

**Kafka Configuration**

The `storm-kafka` connector requires some configuration of the Apache Kafka installation. Kafka administrators must add a `zookeeper.connect` property with the hostnames and port numbers of the HDP Zookeeper nodes to Kafka's `server.properties` file.

```
zookeeper.connect=host1:2181,host2:2181,host3:2181
```

# 2.3. Writing Data with Storm

Hortonworks provides the `storm-hdfs` and `storm-hbase` connectors to enable Storm developers to quickly write streaming data to a Hadoop cluster. These connectors are located at `/usr/lib/storm/contrib`, and each contains a .jar file containing the connector's packaged classes and dependencies and another .jar file with javadoc reference documentation. Both `storm-hdfs` and `storm-hbase` support writing to HDFS clusters using Kerberos. In addition, the `storm-hdfs` connector supports HA-enabled clusters.

## 2.3.1. Writing Data with the Storm-Hdfs Connector

The `storm-hdfs` connector provides the following key features:

- Supports HDFS 2.x
- Supports HA-enabled clusters
- Supports both text and sequence files
- Configurable directory and file names
- Customizable synchronization, rotation policies, and rotation actions
- Tuple fails if HDFS update fails
- Supports the Trident API
- Supports writing to kerberized Hadoop cluster

The primary classes of the `storm-hdfs` connector are `HdfsBolt` and `SequenceFileBolt`, both located in the `org.apache.storm.hdfs.bolt` package. Use the `HDFSBolt` class to write text data to HDFS and the `SequenceFileBolt` class to write binary data. Storm developers specify the following information when instantiating the bolt:

### Table 2.4. `HdfsBolt` Methods

| HdfsBolt Method | Description |
|---|---|
| `withFsUrl` | Specifies the target HDFS URL and port number for clusters running without HA. For clusters with HA enabled, this parameter specifies the nameservice ID in the following format: `hdfs://nameserviceID`. No port number is specified when passing a nameservice ID for an HA-enabled cluster. You can find the nameservice ID as the value asigned to the `dfs.nameservices` parameter in the `core-site.xml` configuration file. |
| `withFileNameFormat` | Specifies the delimiter that indicates a |

| HdfsBolt Method | Description |
| --- | --- |
| | boundary between data records. Storm developers can customize by writing their own implementation of the `org.apache.storm.hdfs.format.RecordFormat` interface. Use the provided `org.apache.storm.hdfs.format.DelimitedRecordFormat` class as a convenience class for writing delimited text data with delimiters such as tabs, comma-separated values, and pipes. The `storm-hdfs` bolt uses the `RecordFormat` implementation to convert tuples to byte arrays, so this method can be used with both text and binary data. |
| `withRotationPolicy` | Specifies when to stop writing to a data file and begin writing to another. Storm developers can customize by writing their own implementation of the `org.apache.storm.hdfs.rotation.FileSizeRotationSizePolicy` interface. |
| `withSyncPolicy` | Specifies how frequently to flush buffered data to the HDFS filesystem. This action enables other Hive clients to read the synchronized data, even as the Storm client continues to write |

| HdfsBolt Method | Description |
|---|---|
| | data. Storm developers can customize by writing their own implementation of the `org.apache.storm.hdfs.sync.SyncPolicy` interface. |
| `withFileNameFormat` | Specifies the name of the data file. Storm developers can customize by writing their own interface of the `org.apache.storm.hdfs.format.FileNameFormat` interface. The provided `org.apache.storm.hdfs.format.DefaultFileNameFormat` creates file names with the following naming format: `{prefix}-{componentId}-{taskId}-{rotationNum}-{timestamp}-{extension}`. For example, `MyBolt-5-7-1390579837830.txt`. |

**Example: Cluster Without HA**

The following example specifies an HDFS path of `hdfs://localhost:54310/foo`, pipe-delimited records ('|'), filesystem sychronization every 1,000 tuples, and data file rotation when files reach five MB. The HdfsBolt is instantiated with an HDFS URL and port number.

```
...
// Use pipe as record boundary
RecordFormat format = new DelimitedRecordFormat().withFieldDelimiter("|");

//Synchronize data buffer with the filesystem every 1000 tuples
SyncPolicy syncPolicy = new CountSyncPolicy(1000);

// Rotate data files when they reach five MB
FileRotationPolicy rotationPolicy = new FileSizeRotationPolicy(5.0f, Units.
MB);

// Use default, Storm-generated file names
FileNameFormat fileNameFormat = new DefaultFileNameFormat().withPath("/foo");

// Instantiate the HdfsBolt
HdfsBolt bolt = new HdfsBolt()
         .withFsURL("hdfs://localhost:54310")
         .withFileNameFormat(fileNameFormat)
         .withRecordFormat(format)
         .withRotationPolicy(rotationPolicy)
```

```
        .withSyncPolicy(syncPolicy);
...
```

**Example: HA-Enabled Cluster**

The following example demonstrates how to modify the previous example to run on an HA-enabled cluster. The HdfsBolt is instantiated with with a nameservice ID rather than an HDFS URL and port number.

```
...
HdfsBolt bolt = new HdfsBolt()
          .withFsURL("hdfs://myNameserviceID")
          .withFileNameFormat(fileNameformat)
          .withRecordFormat(format)
          .withRotationPolicy(rotationPolicy)
          .withSynPolicy(syncPolicy);
...
```

**Trident API**

The `storm-hdfs` connector supports the Trident API. Trident. Hortonworks recommends that Storm developers use the `trident` API unless your application requires sub-second latency.

The Trident API implements a `StateFactory` class with an API that resembles the methods from the `storm-code` API as shown in the following code sample:

```
Fields hdfsFields = new Fields("field1", "field2");

 FileNameFormat fileNameFormat = new DefaultFileNameFormat()
             .withPrefix("trident")
             .withExtension(".txt")
             .withPath("/trident");

 RecordFormat recordFormat = new DelimitedRecordFormat()
             .withFields(hdfsFields);

 FileRotationPolicy rotationPolicy = new FileSizeRotationPolicy(5.0f,
 FileSizeRotationPolicy.Units.MB);

 HdfsState.Options options = new HdfsState.HdfsFileOptions()
            .withFileNameFormat(fileNameFormat)
            .withRecordFormat(recordFormat)
            .withRotationPolicy(rotationPolicy)
            .withFsUrl("hdfs://localhost:54310");

 StateFactory factory = new HdfsStateFactory().withOptions(options);

 TridentState state = stream
            .partitionPersist(factory, hdfsFields, new HdfsUpdater(), new
 Fields());
```

See the javadoc for the Trident API, included with the `storm-hdfs` connector, for more information.

**Limitations**

Directory and file names changes are limited to a prepackaged file name format based on a timestamp.

## 2.3.2. Writing Data with the Storm-HBase Connector

The `storm-hbase` connector provides the following key features:

- Supports Apache HBase 0.96 and above
- Supports incrementing counter columns
- Tuples are failed if an update to an HBase table fails
- Ability to group puts in a single batch
- Supports writing to Kerberized HBase clusters

The `storm-hbase` connector enables Storm developers to collect several *PUTS* in a single operation and write to multiple HBase column families and counter columns. A PUT is an HBase operation that inserts data into a single HBase cell. Use the HBase client's write buffer to automatically batch: `hbase.client.write.buffer`. The primary interface in the `storm-hbase` connector is the `org.apache.storm.hbase.bolt.mapper.HBaseMapper` interface. However, the default implementation,`SimpleHBaseMapper`, writes a single column family. Storm developers can implement the `HBaseMapper` interface themselves or extend `SimpleHBaseMapper` if they want to change or override this behavior.

### Table 2.5. SimpleHBaseMapper Methods

| SimpleHBaseMapper Method | Description |
|---|---|
| `withRowKeyField` | Specifies the *row key* for the target HBase row. A row key uniquely identifies a row in HBase. |
| `withColumnFields` | Specifies the target HBase column. |
| `withCounterFields` | Specifies the target HBase counter. |
| `withColumnFamily` | Specifies the target HBase column family. |

**Example**

The following example specifies the 'word' tuple as the row key, adds an HBase column for the tuple 'word' field, adds an HBase counter column for the tuple 'count' field, and writes data to the 'cf' column family.

```
SimpleHBaseMapper mapper = new SimpleHBaseMapper()
         .withRowKeyField("word")
         .withColumnFields(new Fields("word"))
         .withCounterFields(new Fields("count"))
         .withColumnFamily("cf");
```

The `storm-hbase` connector supports the following versions of HBase:

- 0.96
- 0.98

**Limitations**

The current version of the `storm-hbase` connector has the following limitations:

- HBase table must be predefined
- Cannot dynamically add new HBase columns; can write to only one column family at a time
- Assumes that `hbase-site.xml` is in the $CLASSPATH environment variable
- Tuple field names must match HBase column names
- Does not support the Trident API
- Supports writes but not lookups

### 2.3.3. Configuring the Storm-HDFS and Storm-HBase Connectors for a Secure Cluster

Storm developers must provide their own Kerberos keytab and principal name for `storm-hdfs` and `storm-hbase` connectors in topologies that run on secure clusters. The `Config` object must contain the storm keytab file and the principal name, as shown in the following example:

```
Config config = new Config();
...
config.put("storm.keytab.file","$keytab");
config.put("storm.kerberos.principal","$principal");
StormSubmitter.submitTopology("$topologyName",config,builder.
createTopology());
```

## 2.4. Packaging Storm Topologies

Storm developers should verify that the following conditions are met when packaging their topology into a .jar file:

- Use the `maven-shade-plugin`, rather than the `maven-assembly-plugin` to package your Apache Storm topologies. The `maven-shade-plugin` provides the ability to merge JAR manifest entries, which are used by the Hadoop client to resolve URL schemes.
- Include a dependency for the Hadoop version used in the Hadoop cluster.
- Include both the `hdfs-site.xml` and `core-site.xml` configuration files in the .jar file. This is the easiest way to meet the requirement that these two files are in the CLASSPATH of your topology.

### Maven Shade Plugin

Use the following Maven configuration file to package your topology:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>1.4</version>
    <configuration>
        <createDependencyReducedPom>true</createDependencyReducedPom>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>shade</goal>
            </goals>
            <configuration>
                <transformers>
                    <transformer
                            implementation="org.apache.maven.plugins.shade.
resource.ServicesResourceTransformer"/>
                    <transformer
                            implementation="org.apache.maven.plugins.shade.
resource.ManifestResourceTransformer">
                        <mainClass></mainClass>
                    </transformer>
                </transformers>
            </configuration>
        </execution>
    </executions>
</plugin>
```

### Hadoop Dependency

The following example demonstrates how to include a dependency for the Hadoop version:

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.2.0</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

### Troubleshooting

The following table describes common packaging errors.

#### Table 2.6. Topology Packing Errors

| Error | Description |
|---|---|
| `com.google.protobuf.InvalidProtocolBufferException: Protocol` | Hadoop client |

| Error | Description |
| --- | --- |
| `message contained an invalid tag (zero)` | version incompatibility |
| `java.lang.RuntimeException: Error preparing HdfsBolt: No FileSystem for scheme: hdfs` | The jar manifest files are not properly merged in the topology .jar |

# 2.5. Deploying and Managing Apache Storm Topologies

Use the command-line interface to deploy a Storm topology after packaging it in a jar. For example, use the following command to deploy WordCountTopology from the `storm-starter` jar:

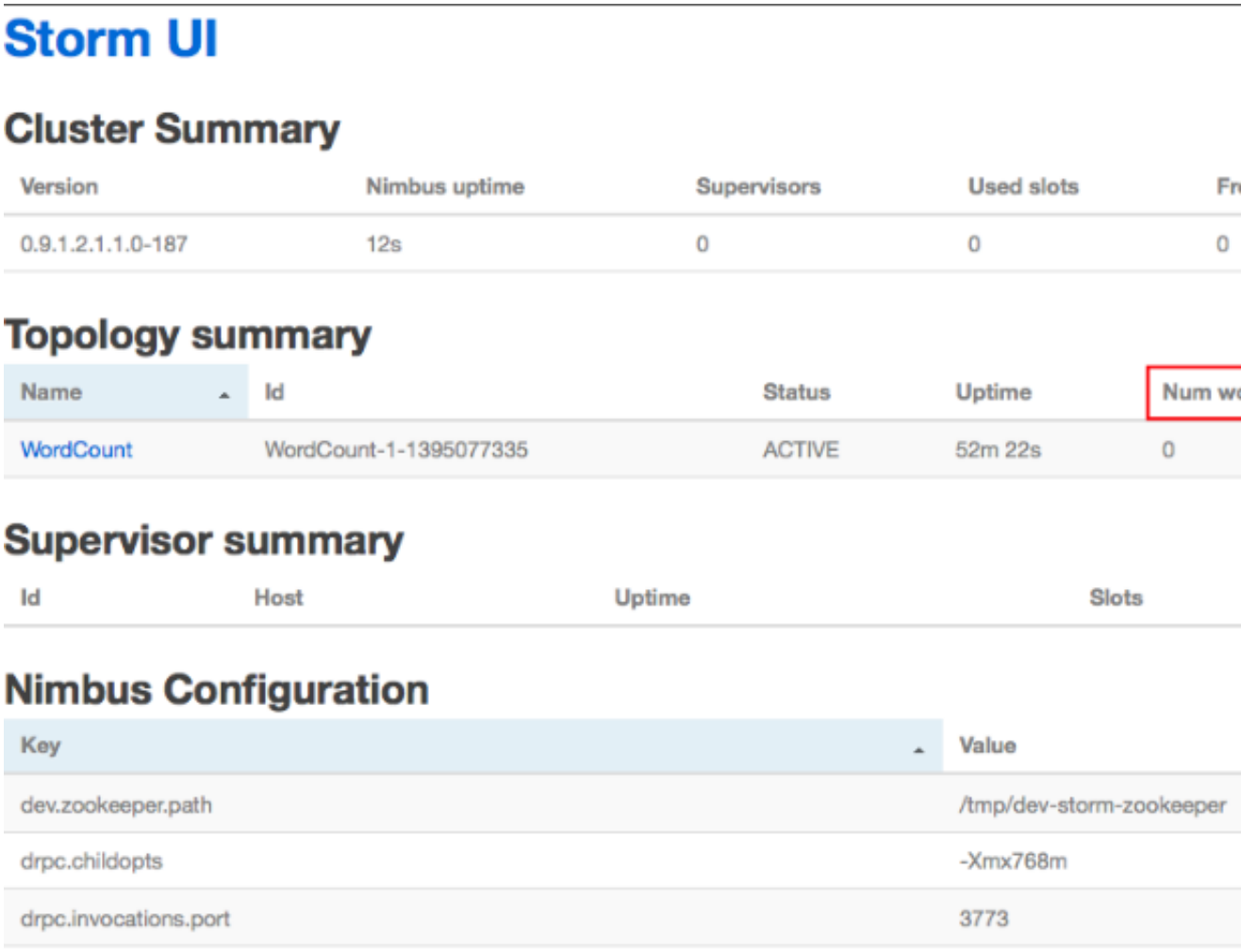
```
storm jar storm-starter-0.0.1-storm-0.9.0.1.jar storm.starter.
WordCountTopology WordCount -c nimbus.host=sandbox.hortonworks.com
```

Point a browser to the following URL to access the Storm UI and to manage deployed topologies.

```
http://<storm-ui-server>:8080
```

## Note

You may need to configure Apache Storm to use a different port if Ambari is also running on the same host with Apache Storm. Both applications use port 8080 by default.



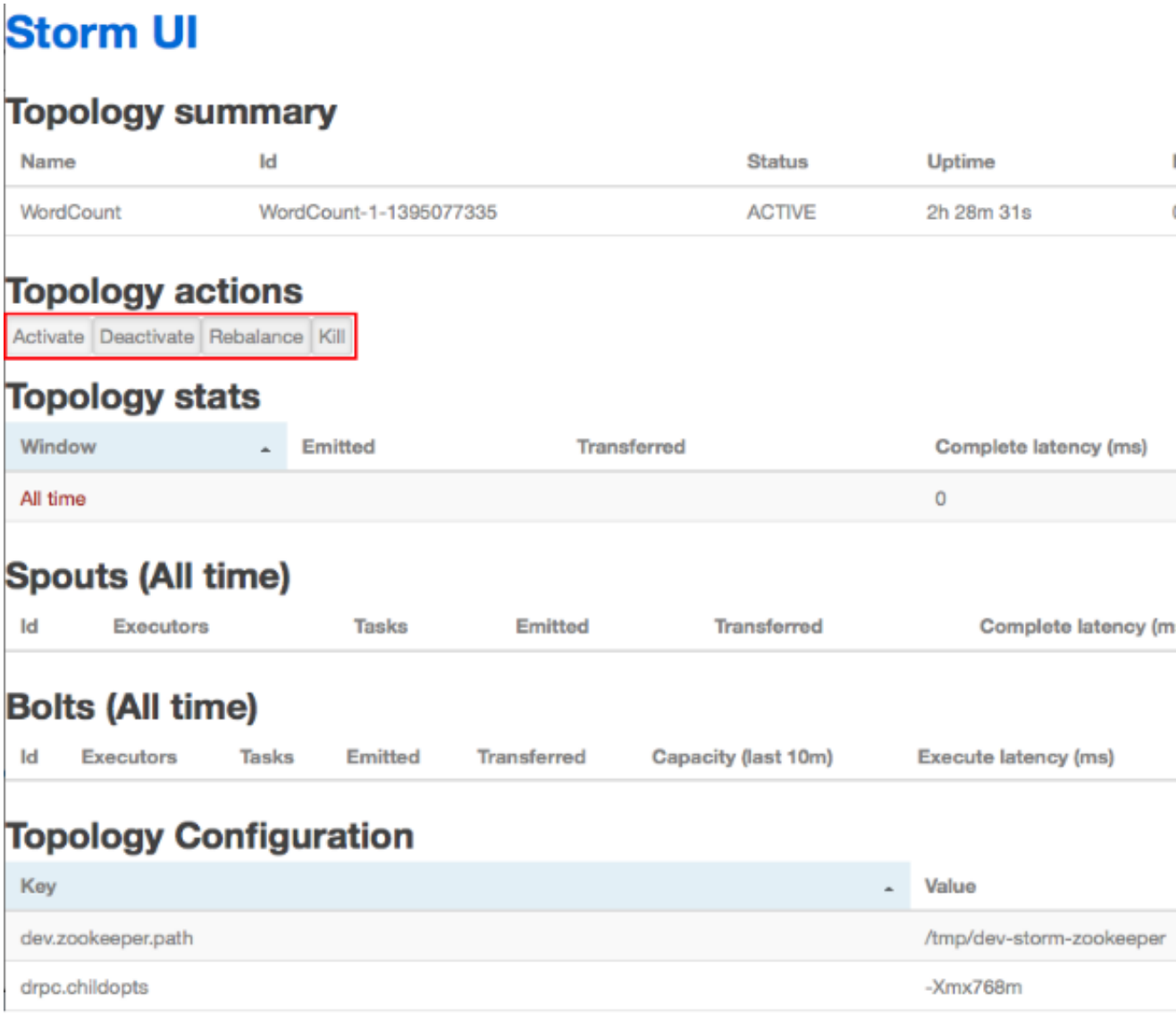

In the image above, no workers, executors, or tasks are running. However, the status of the topology remains active and the uptime continues to increase. Storm topologies, unlike traditional applications, remain active until an administrator deactivates or kills them. Storm administrators use the Storm user interface to perform the following administrative actions:

### Table 2.7. Topology Administrative Actions

| Topology Administrative Action | Description |
|---|---|
| Activate | Returns a topology to active status after it has been deactivated. |
| Deactivate | Sets the status of a topology to inactive. Topology uptime is not affected by deactivation. |

| Topology Administrative Action | Description |
| --- | --- |
| Rebalance | Dynamically increase or decrease the number of worker processes and/or executors. The administrator does not need to restart the cluster or the topology. |
| Kill | Stops the topology and removes it from Apache Storm. The topology no longer appears in the Storm UI, and the administrator must deploy the application again to activate it. |

Click any topology in the Topology Summary section to launch the Topology Summary page. Administrators perform any of the topology actions in the table above by clicking the corresponding button, shown in the following image.




The Executors field in the Spouts and Bolts sections show all running Storm threads, including the host and port. If a bolt is experiencing latency issues, Storm developers should look here to determine which executor has reached capacity. Click the port number to display the log file for the corresponding executor.

## 2.6. Example: RollingTopWords Topology

The `RollingTopWords.java` is included with `storm-starter`.

```
package storm.starter;

import backtype.storm.Config;
import backtype.storm.testing.TestWordSpout;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.tuple.Fields;
import storm.starter.bolt.IntermediateRankingsBolt;
import storm.starter.bolt.RollingCountBolt;
```

```
import storm.starter.bolt.TotalRankingsBolt;
import storm.starter.util.StormRunner;

/**
* This topology does a continuous computation of the top N words that the
 topology has seen in terms of cardinality.
* The top N computation is done in a completely scalable way, and a similar
 approach could be used to compute things
* like trending topics or trending images on Twitter.
*/
public class RollingTopWords {

  private static final int DEFAULT_RUNTIME_IN_SECONDS = 60;
  private static final int TOP_N = 5;

  private final TopologyBuilder builder;
  private final String topologyName;
  private final Config topologyConfig;
  private final int runtimeInSeconds;

  public RollingTopWords() throws InterruptedException {
    builder = new TopologyBuilder();
    topologyName = "slidingWindowCounts";
    topologyConfig = createTopologyConfiguration();
    runtimeInSeconds = DEFAULT_RUNTIME_IN_SECONDS;

    wireTopology();
  }

  private static Config createTopologyConfiguration() {
    Config conf = new Config();
    conf.setDebug(true);
    return conf;
  }

  private void wireTopology() throws InterruptedException {
    String spoutId = "wordGenerator";
    String counterId = "counter";
    String intermediateRankerId = "intermediateRanker";
    String totalRankerId = "finalRanker";
    builder.setSpout(spoutId, new TestWordSpout(), 5);
    builder.setBolt(counterId, new RollingCountBolt(9, 3), 4).
fieldsGrouping(spoutId, new Fields("word"));
    builder.setBolt(intermediateRankerId, new IntermediateRankingsBolt(TOP_N),
 4).fieldsGrouping(counterId, new Fields(
        "obj"));
    builder.setBolt(totalRankerId, new TotalRankingsBolt(TOP_N)).
globalGrouping(intermediateRankerId);
  }

  public void run() throws InterruptedException {
    StormRunner.runTopologyLocally(builder.createTopology(), topologyName,
 topologyConfig, runtimeInSeconds);
  }

  public static void main(String[] args) throws Exception {
    new RollingTopWords().run();
  }
}
```

# 3. HBase Snapshots

HBase Snapshots allow you to take a snapshot of a table without too much impact on Region Servers. Snapshot, Clone and restore operations don't involve data copying. Also, Exporting the snapshot to another cluster doesn't have impact on the Region Servers.

Prior to version 0.94.6, the only way to backup or to clone a table is to use CopyTable/ ExportTable, or to copy all the hfiles in HDFS after disabling the table. The disadvantages of these methods are that you can degrade region server performance (Copy/Export Table) or you need to disable the table, that means no reads or writes; and this is usually unacceptable. In this section:

- Configuration
- Take a Snapshot
- Listing Snapshots
- Deleting Snapshots
- Clone a table from snapshot
- Restore a snapshot
- Snapshots operations and ACLs
- Export to another cluster

## 3.1. Configuration

To turn on the snapshot support just set the `hbase.snapshot.enabled` property to true. (Snapshots are enabled by default in 0.95+ and off by default in 0.94.6+)

```
<property>
  <name>hbase.snapshot.enabled</name>
  <value>true</value>
</property>
```

## 3.2. Take a Snapshot

You can take a snapshot of a table regardless of whether it is enabled or disabled. The snapshot operation doesn't involve any data copying.

```
  $ hbase shell
  hbase> snapshot 'myTable', 'myTableSnapshot-122112'
```

## 3.3. Listing Snapshots

List all snapshots taken (by printing the names and relative information).

```
$ hbase shell
hbase> list_snapshots
```

## 3.4. Deleting Snapshots

You can remove a snapshot, and the files retained for that snapshot will be removed if no longer needed.

```
$ hbase shell
hbase> delete_snapshot 'myTableSnapshot-122112'
```

## 3.5. Clone a table from snapshot

From a snapshot you can create a new table (clone operation) with the same data that you had when the snapshot was taken. The clone operation, doesn't involve data copies, and a change to the cloned table doesn't impact the snapshot or the original table.

```
$ hbase shell
hbase> clone_snapshot 'myTableSnapshot-122112', 'myNewTestTable'
```

## 3.6. Restore a snapshot

The restore operation requires the table to be disabled, and the table will be restored to the state at the time when the snapshot was taken, changing both data and schema if required.

```
$ hbase shell
hbase> disable 'myTable'
hbase> restore_snapshot 'myTableSnapshot-122112'
```

### Note

Since Replication works at log level and snapshots at file-system level, after a restore, the replicas will be in a different state from the master. If you want to use restore, you need to stop replication and redo the bootstrap.

In case of partial data-loss due to misbehaving client, instead of a full restore that requires the table to be disabled, you can clone the table from the snapshot and use a Map-Reduce job to copy the data that you need, from the clone to the main one.

## 3.7. Snapshots operations and ACLs

If you are using security with the AccessController Coprocessor, only a global administrator can take, clone, or restore a snapshot, and these actions do not capture the ACL rights. This

means that restoring a table preserves the ACL rights of the existing table, while cloning a table creates a new table that has no ACL rights until the administrator adds them.

## 3.8. Export to another cluster

The ExportSnapshot tool copies all the data related to a snapshot (hfiles, logs, snapshot metadata) to another cluster. The tool executes a Map-Reduce job, similar to distcp, to copy files between the two clusters, and since it works at file-system level the hbase cluster does not have to be online. The HBase Snapshot Export tool must be run as hbase user. The HBase Snapshot Export tool must have temp directory, specified by "hbase.tmp.dir" (ie / grid/0/var/log/hbase), created on HDFS with hbase user as the owner.

To copy a snapshot called MySnapshot to an HBase cluster srv2 (hdfs://srv2:8020/hbase) using 16 mappers:

```
$ hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot MySnapshot -
copy-to hdfs://yourserver:8020/hbase_root_dir -mappers 16
```

# 4. User Guide - HDFS NFS Gateway

The NFS Gateway for HDFS supports NFSv3 and lets you mount HDFS as part of the client's local file system.

This release of NFS Gateway supports and enables the following usage patterns:

- Browse the HDFS file system through their local file system on NFSv3 client compatible operating systems.
- Download files from the the HDFS file system on to their local file system.
- Upload files from their local file system directly to the HDFS file system.
- Stream data directly to HDFS through the mount point. (File append is supported but random write is not supported.)

## 4.1. Prerequisites

In this section:

- The NFS gateway machine must have everything to run an HDFS client such as a Hadoop core JAR file and a HADOOP_CONF directory.
- The NFS gateway can be either on any DataNode, NameNode, or any HDP client machine. Start the NFS server on that machine.

## 4.2. Instructions

To configure and use the HDFS NFS gateway, complete the following steps:

1. The user running the NFS gateway must be able to proxy all of the users using the NFS mounts. For instance, if user "nfsserver" is running the gateway, and users belonging to the groups "nfs-users1" and "nfs-users2" use the NFS mounts, then in the `core-site.xml` file on the NameNode, the following must be set.

**Note**

Replace "nfsserver" with the user name starting the gateway in your cluster.

```
<property>
  <name>hadoop.proxyuser.nfsserver.groups</name>
  <value>nfs-users1,nfs-users2</value>
  <description>
         The 'nfsserver' user is allowed to proxy all members of the 'nfs-
users1' and
         'nfs-users2' groups. Set this to '*' to allow nfsserver user to
 proxy any group.
  </description>
</property>
```

```
<property>
  <name>hadoop.proxyuser.nfsserver.hosts</name>
  <value>nfs-client-host1.com</value>
  <description>
         This is the host where the nfs gateway is running. Set this to '*'
 to allow
         requests from any hosts to be proxied.
  </description>
</property>
```

The preceding properties are the only required configuration for the NFS gateway in non-secure mode. For Kerberized Hadoop clusters, the following configurations need to be added to `hdfs-site.xml`:

```
  <property>
    <name>dfs.nfsgateway.keytab.file</name>
    <value>/etc/hadoop/conf/nfsserver.keytab</value> <!-- path to the nfs
 gateway keytab -->
  </property>

  <property>
    <name>dfs.nfsgateway.kerberos.principal</name>
    <value>nfsserver/_HOST@YOUR-REALM.COM</value>
  </property>
```

2. Configure settings for HDFS NFS gateway:

   NFS gateway uses the same configurations as used by the NameNode and DataNode. Configure the following properties based on your application's requirements:

   a. Edit the `hdfs-site.xml` file on your NFS gateway machine and modify the following property:

   ```
   <property>
     <name>dfs.namenode.accesstime.precision</name>
     <value>3600000</value>
     <description>The access time for HDFS file is precise up to this value.

                  The default value is 1 hour. Setting a value of 0 disables
                  access times for HDFS.
     </description>
   </property>
   ```

   **Note**

   If the export is mounted with access time update allowed, make sure this property is not disabled in the configuration file. Only NameNode needs to restart after this property is changed. If you have disabled access time update by mounting with "noatime" you do NOT have to change this property nor restart your NameNode.

   b. Add the following property to `hdfs-site.xml` file:

   ```
   <property>
       <name>dfs.nfs3.dump.dir</name>
       <value>/tmp/.hdfs-nfs</value>
   </property>
   ```

**Note**

NFS client often reorders writes. Sequential writes can arrive at the NFS gateway at random order. This directory is used to temporarily save out-of-order writes before writing to HDFS. One needs to make sure the directory has enough space. For example, if the application uploads 10 files with each having 100MB, it is recommended for this directory to have 1GB space in case if a worst-case write reorder happens to every file.

c. Update the following property in the `hdfs-site.xml` file:

```
<property>
    <name>dfs.nfs.exports.allowed.hosts</name>
    <value>* rw</value>
</property>
```

**Note**

By default, the export can be mounted by any client. You must update this property to control access. The value string contains the machine name and access privilege, separated by whitespace characters. The machine name can be in single host, wildcard, or IPv4 network format. The access privilege uses `rw` or `ro` to specify `readwrite` or `readonly` access to exports. If you do not specifiy an access privilege, the default machine access to exports is `readonly`. Separate machine dentries by `;`. For example, `192.168.0.0/22 rw ; host*.example.com ; host1.test.org ro;`.

Restart the NFS gateway after this property is updated.

d. Optional - Customize log settings.

Edit the `log4j.property` file to add the following:

To change trace level, add the following:

```
log4j.logger.org.apache.hadoop.hdfs.nfs=DEBUG
```

To get more details of ONCRPC requests, add the following:

```
log4j.logger.org.apache.hadoop.oncrpc=DEBUG
```

3. Start the NFS gateway service.

Three daemons are required to provide NFS service: `rpcbind` (or `portmap`), `mountd` and `nfsd`. The NFS gateway process has both `nfsd` and `mountd`. It shares the HDFS root "/" as the only export. We recommend using the `portmap` included in NFS gateway package as shown below:

a. Stop `nfs`/`rpcbind`/`portmap` services provided by the platform:

```
service nfs stop
service rpcbind stop
```

b. Start the included `portmap` package (needs root privileges):

```
hadoop portmap
```

OR

```
hadoop-daemon.sh start portmap
```

c. Start `mountd` and `nfsd`.

No root privileges are required for this command. However, verify that the user starting the Hadoop cluster and the user starting the NFS gateway are same.

```
hadoop nfs3
```

OR

```
hadoop-daemon.sh start nfs3
```

**Note**

If the `hadoop-daemon.sh` script starts the NFS gateway, its log file can be found in the hadoop log folder (`/var/log/hadoop`).

For example, if you launched the NFS gateway services as the root user, the log file would be found in a path similar to the following:

```
/var/log/hadoop/root/hadoop-root-nfs3-63ambarihdp21.log
```

d. Stop NFS gateway services.

```
hadoop-daemon.sh stop nfs3
hadoop-daemon.sh stop portmap
```

4. Verify validity of NFS-related services.

a. Execute the following command to verify that all the services are up and running:

```
rpcinfo -p $nfs_server_ip
```

You should see output similar to the following:

```
   program vers proto   port

    100005    1   tcp   4242  mountd

    100005    2   udp   4242  mountd

    100005    2   tcp   4242  mountd

    100000    2   tcp    111  portmapper

    100000    2   udp    111  portmapper

    100005    3   udp   4242  mountd

    100005    1   udp   4242  mountd
```

```
    100003    3   tcp   2049  nfs

    100005    3   tcp   4242  mountd
```

b. Verify that the HDFS namespace is exported and can be mounted.

```
showmount -e $nfs_server_ip
```

You should see output similar to the following:

```
Exports list on $nfs_server_ip :
/ (everyone)
```

5. Mount the export "/".

Currently NFS v3 is supported and uses TCP as the transportation protocol is TCP. The users can mount the HDFS namespace as shown below:

```
mount -t nfs -o vers=3,proto=tcp,nolock $server:/  $mount_point
```

Then the users can access HDFS as part of the local file system except that hard/symbolic link and random write are not supported in this release.

**Note**

Because NLM is not supported, the mount option `nolock` is needed.

**User authentication and mapping:**

NFS gateway in this release uses `AUTH_UNIX` style authentication which means that the the login user on the client is the same user that NFS passes to the HDFS. For example, if the NFS client has current user as `admin`, when the user accesses the mounted directory, NFS gateway will access HDFS as user `admin`. To access HDFS as `hdfs` user, you must first switch the current user to `hdfs` on the client system before accessing the mounted directory.

6. Set up client machine users to interact with HDFS through NFS.

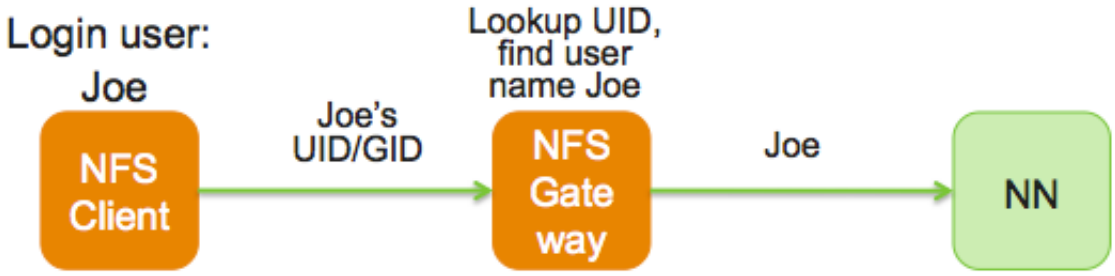
NFS gateway converts the UID to user name and HDFS uses username for checking permissions.

The system administrator must ensure that the user on NFS client machine has the same name and UID as that on the NFS gateway machine. This is usually not a problem if you use the same user management system such as LDAP/NIS to create and deploy users to HDP nodes and to client node.

If the user is created manually, you might need to modify the UID on either the client or NFS gateway host in order to make them the same:

```
usermod -u 123 $myusername
```

The following illustrates how the UID and name are communicated between the NFS client, NFS gateway, and NameNode.


Login user:
Joe
NFS Client
Joe's UID/GID
Lookup UID, find user name Joe
NFS Gate way
Joe
NN

# 5. User Guide - HDFS Snapshots

HDFS Snapshots are read-only point-in-time copies of the file system. Snapshots can be taken on a subtree of the file system or the entire file system. Some common use cases of snapshots are data backup, protection against user errors and disaster recovery.

The implementation of HDFS Snapshots is efficient:

1. Snapshot creation is instantaneous: the cost is **O(1)** excluding the inode lookup time.
2. Additional memory is used only when modifications are made relative to a snapshot: memory usage is **O(M)** where **M** is the number of modified files/directories.
3. Blocks in datanodes are not copied: the snapshot files record the block list and the file size. There is no data copying.
4. Snapshots do not adversely affect regular HDFS operations: modifications are recorded in reverse chronological order so that the current data can be accessed directly. The snapshot data is computed by subtracting the modifications from the current data.

In this document:



## 5.1. Snapshottable Directories

Snapshots can be taken on any directory once the directory has been set as **snapshottable**. A snapshottable directory is able to accommodate 65,536 simultaneous snapshots. There is no limit on the number of snapshottable directories. Administrators may set any directory to be snapshottable. If there are snapshots in a snapshottable directory, the directory can be neither deleted nor renamed before all the snapshots are deleted.

## 5.2. Snapshot Paths

For a snapshottable directory, the path component **".snapshot"** is used for accessing its snapshots. Suppose `/foo` is a snapshottable directory, `/foo/bar` is a file/directory in `/foo`, and `/foo` has a snapshot `s0`. Then, the path `/foo/.snapshot/s0/bar` refers to the snapshot copy of `/foo/bar`. The usual API and CLI can work with the ".snapshot" paths. The following are some examples:

- Listing all the snapshots under a snapshottable directory: `hadoop dfs -ls /foo/.snapshot`
- Listing the files in snapshot `s0`: `hadoop dfs -ls /foo/.snapshot/s0`
- Copying a file from snapshot `s0`: `hadoop dfs -cp /foo/.snapshot/s0/bar /tmp`

The name ".snapshot" is now a reserved file name in HDFS so that users cannot create a file/directory with ".snapshot" as the name. If ".snapshot" is used in a previous version of HDFS, it must be renamed before upgrade; otherwise, upgrade will fail.

# 5.3. Snapshot Operations

Snapshot operations are grouped into the following two categories:

- Administrator Operations
- User Operations

## 5.3.1. Administrator Operations

The operations described in this section require superuser privileges.

- Allow Snapshots: Allowing snapshots of a directory to be created. If the operation completes successfully, the directory becomes snapshottable.
  - Command:

```
hadoop dfsadmin -allowSnapshot $path
```

  - Arguments:

**Table 5.1. Administrator Operations - Allow Snapshots**

| Parameter name | Description |
|---|---|
| path | The path of the snapshottable directory. |

See also the corresponding Java API `void allowSnapshot(Path path)` in `HdfsAdmin`.

- Disallow Snapshots: Disallowing snapshots of a directory to be created. All snapshots of the directory must be deleted before disallowing snapshots.
  - Command:

```
hadoop dfsadmin -disallowSnapshot $path
```

  - Arguments:

**Table 5.2. Administrator Operations - Disallow Snapshots**

| Parameter name | Description |
|---|---|
| path | The path of the snapshottable directory. |

See also the corresponding Java API `void disallowSnapshot(Path path)` in `HdfsAdmin`.

## 5.3.2. User Operations

The section describes user operations. Note that HDFS superuser can perform all the operations without satisfying the permission requirement in the individual operations.

- Create Snapshots: Create a snapshot of a snapshottable directory. This operation requires owner privilege to the snapshottable directory.
  - Command:

```
hadoop dfs -createSnapshot $path $snapshotName
```

  - Arguments:

**Table 5.3. User Operations - Create Snapshots**

| Parameter name | Description |
|---|---|
| path | The path of the snapshottable directory. |
| snapshotName | The snapshot name, which is an optional argument. When it is omitted, a default name is generated using a timestamp with the format "'s'yyyyMMdd-HHmmss.SSS", e.g. "s20130412-151029.033". |

See also the corresponding Java API `Path createSnapshot(Path path)` and `Path createSnapshot(Path path, String snapshotName)` in `FileSystem`. The snapshot path is returned in these methods.

- Delete Snapshots: Delete a snapshot of from a snapshottable directory. This operation requires owner privilege of the snapshottable directory.
  - Command:

```
hadoop dfs -deleteSnapshot $path $snapshotName
```

  - Arguments:

**Table 5.4. User Operations - Delete Snapshots**

| Parameter name | Description |
|---|---|
| path | The path of the snapshottable directory. |
| snapshotName | The snapshot name. |

See also the corresponding Java API `void deleteSnapshot(Path path, String snapshotName)` in `FileSystem`.

- Rename Snapshots: Rename a snapshot. This operation requires owner privilege of the snapshottable directory..
  - Command:

```
hadoop dfs -renameSnapshot $path $oldName $newName
```

  - Arguments:

### Table 5.5. User Operations - Rename Snapshots

| Parameter name | Description |
|---|---|
| path | The path of the snapshottable directory. |
| oldName | The old snapshot name. |
| newName | The new snapshot name. |

See also the corresponding Java API `void renameSnapshot(Path path, String oldName, String newName)` in `FileSystem`.

- Get Snapshottable Directory Listing: Get all the snapshottable directories where the current user has permission to take snapshots.
  - Command:

    ```
    hadoop lsSnapshottableDir
    ```

  - Arguments: N/A

  See also the corresponding Java API `SnapshottableDirectoryStatus[] getSnapshottableDirectoryListing()` in `DistributedFileSystem`.

- Get Snapshots Difference Report: Get the differences between two snapshots. This operation requires read access privilege for all files/directories in both snapshots.
  - Command:

    ```
    hadoop snapshotDiff $path $fromSnapshot $toSnapshot
    ```

  - Arguments:

### Table 5.6. User Operations - Get Snapshots Difference Report

| Parameter name | Description |
|---|---|
| path | The path of the snapshottable directory. |
| fromSnapshot | The name of the starting snapshot. |
| toSnapshot | The name of the ending snapshot. |

# 6. Add HDP Maven Repository to Existing Project

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven projects are defined by their **Project Object Model** or `pom`. This file is located in the base directory of a maven project and is called pom.xml.

Use one of the following options to add HDP Maven repository as a default repository in your existing project:

- **Option I: Add HDP Maven repository to existing Maven project**

  A repository in Maven is used to hold build artifacts and dependencies of varying types. There are strictly only two types of repositories: local and remote.

  The local repository refers to a copy on your own installation that is a cache of the remote downloads, and also contains the temporary build artifacts that you have not yet released. Remote repositories refer to any other type of repository, accessed by a variety of protocols such as `file://` and `http://` These repositories might be a truly remote repository set up by a third party to provide their artifacts for downloading (for example, `repo.maven.apache.org` hosts Maven's central repository). Other "remote" repositories may be internal repositories set up on a file or HTTP server within your company, used to share private artifacts between development teams and for releases.

  To add HDP Maven repository, add the following lines to your Maven project's pom.xml file:

```
<repositories>
    <repository>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
        <checksumPolicy>warn</checksumPolicy>
      </releases>
      <snapshots>
        <enabled>false</enabled>
        <updatePolicy>never</updatePolicy>
        <checksumPolicy>fail</checksumPolicy>
      </snapshots>
      <id>HDPReleases</id>
      <name>HDP Releases</name>
      <url>http://repo.hortonworks.com/content/repositories/releases/</url>
      <layout>default</layout>
    </repository>
  </repositories>
```

- **Option II: Add HDP Maven repository to existing Ant/Ivy project**

  Apache Ivy repositories are configured inside the `<resolvers>` element of an `ivysettings.xml` file. Usually, the resolvers (where to get those required artifacts) are provided through a separate file,`ivysettings.xml` file.

The `ivysettings.xml` file holds a chain of Ivy resolvers used for both resolution and publishing (deployment). Resolvers exist for both regular artifacts and Ivy module files. Apache Ivy uses `chain` to define the preference order for the repositories. Inside the `<chain>` element, you will find a `<url>` element. The `<url>` element is a remote site that contains bundle dependencies.

To add HDP Maven repository to existing Ant/Ivy project, add a new resolver to the existing Ivy chain so that HDP versioned artifacts can be resolved.

- **Option III: Setup Maven proxy**

  It is often the case that users wish to set up a Maven proxy repository inside their corporate firewall and have developer instances resolve artifacts through such a proxy. Proxy repositories provide a single point of remote download for an organization. In addition to control and security concerns, Proxy repositories are primarily important for increased speed across a team. These scenarios can be realized by using internal Maven repositories and a Maven proxy.

  To setup maven proxy pointing to HDP Maven or Nexus repository, use the following URL (http://repo.hortonworks.com/content/repositories/releases/ ) for caching the HDP artifacts to your local or internal Maven, Nexus, or Archiva repositories respectively.

# 7. Apache Flume User Guide

The Apache Flume User Guide contains the following sections: