Chukwa User and Programming Guide

Table of contents

At the core of Chukwa is a flexible system for collecting and processing monitoring data, particularly log files. This document describes how to use the collected data. (For an overview of the Chukwa data model and collection pipeline, see the Design Guide.)

In particular, this document discusses the Chukwa archive file formats, the demux and archiving mapreduce jobs, and the layout of the Chukwa storage directories.

1. Reading data from the sink or the archive

Chukwa gives you several ways of inspecting or processing collected data.

1.1. Dumping some data

It very often happens that you want to retrieve one or more files that have been collected with Chukwa. If the total volume of data to be recovered is not too great, you can use `bin/chukwa dumpArchive`, a command-line tool that does the job. The `dump` tool does an in-memory sort of the data, so you'll be constrained by the Java heap size (typically a few hundred MB).

The `dump` tool takes a search pattern as its first argument, followed by a list of files or file-globs. It will then print the contents of every data stream in those files that matches the pattern. (A data stream is a sequence of chunks with the same host, source, and datatype.) Data is printed in order, with duplicates removed. No metadata is printed. Separate streams are separated by a row of dashes.

For example, the following command will dump all data from every file that matches the glob pattern. Note the use of single quotes to pass glob patterns through to the application, preventing the shell from expanding them.

```
$CHUKWA_HOME/bin/chukwa dumpArchive 'datatype=.*'
'hdfs://host:9000/chukwa/archive/*.arc'
```

The patterns used by `dump` are based on normal regular expressions. They are of the form `field1=regex&field2=regex`. That is, they are a sequence of rules, separated by ampersand signs. Each rule is of the form `metadatafield=regex`, where `metadatafield` is one of the Chukwa metadata fields, and `regex` is a regular expression. The valid metadata field names are: `datatype`, `host`, `cluster`, `content`, `name`. Note that the `name` field matches the stream name -- often the filename that the data was extracted from.

In addition, you can match arbitrary tags via `tags.tagname`. So for instance, to match chunks with tag `foo="bar"` you could say `tags.foo=bar`. Note that quotes are present in the tag, but not in the filter rule.

A stream matches the search pattern only if every rule matches. So to retrieve HadoopLog data from cluster foo, you might search for `cluster=foo&datatype=HadoopLog`.

1.2. Exploring the Sink or Archive

Another common task is finding out what data has been collected. Chukwa offers a specialized tool for this purpose: `DumpArchive`. This tool has two modes: summarize and verbose, with the latter being the default.

In summarize mode, `DumpArchive` prints a count of chunks in each data stream. In verbose mode, the chunks themselves are dumped.

You can invoke the tool by running `$CHUKWA_HOME/bin/dumpArchive.sh`. To specify summarize mode, pass `--summarize` as the first argument.

```
bin/chukwa dumpArchive --summarize 'hdfs://host:9000/chukwa/logs/*.done'
```

1.3. Using MapReduce

A key goal of Chukwa was to facilitate MapReduce processing of collected data. The next section discusses the file formats. An understanding of MapReduce and SequenceFiles is helpful in understanding the material.

2. Sink File Format

As data is collected, Chukwa dumps it into *sink files* in HDFS. By default, these are located in `hdfs:///chukwa/logs`. If the file name ends in .chukwa, that means the file is still being written to. Every few minutes, the collector will close the file, and rename the file to '*.done'. This marks the file as available for processing.

Each sink file is a Hadoop sequence file, containing a succession of key-value pairs, and periodic synch markers to facilitate MapReduce access. They key type is `ChukwaArchiveKey`; the value type is `ChunkImpl`. See the Chukwa Javadoc for details about these classes.

Data in the sink may include duplicate and omitted chunks.

3. Demux and Archiving

It's possible to write MapReduce jobs that directly examine the data sink, but it's not extremely convenient. Data is not organized in a useful way, so jobs will likely discard most of their input. Data quality is imperfect, since duplicates and omissions may exist. And

MapReduce and HDFS are optimized to deal with a modest number of large files, not many small ones.

Chukwa therefore supplies several MapReduce jobs for organizing collected data and putting it into a more useful form; these jobs are typically run regularly from cron. Knowing how to use Chukwa-collected data requires understanding how these jobs lay out storage. For now, this document only discusses one such job: the Simple Archiver.

4. Simple Archiver

The simple archiver is designed to consolidate a large number of data sink files into a small number of archive files, with the contents grouped in a useful way. Archive files, like raw sink files, are in Hadoop sequence file format. Unlike the data sink, however, duplicates have been removed. (Future versions of the Simple Archiver will indicate the presence of gaps.)

The simple archiver moves every `.done` file out of the sink, and then runs a MapReduce job to group the data. Output Chunks will be placed into files with names of the form `hdfs:///chukwa/archive/clustername/Datatype_date.arc`. Date corresponds to when the data was collected; Datatype is the datatype of each Chunk.

If archived data corresponds to an existing filename, a new file will be created with a disambiguating suffix.

5. Demux

A key use for Chukwa is processing arriving data, in parallel, using MapReduce. The most common way to do this is using the Chukwa demux framework. As data flows through Chukwa, the demux job is often the first job that runs.

By default, Chukwa will use the default TsProcessor. This parser will try to extract the real log statement from the log entry using the ISO8601 date format. If it fails, it will use the time at which the chunk was written to disk (collector timestamp).

5.1. Writing a custom demux Mapper

If you want to extract some specific information and perform more processing you need to write your own parser. Like any M/R program, your have to write at least the Map side for your parser. The reduce side is Identity by default.

On the Map side,you can write your own parser from scratch or extend the AbstractProcessor class that hides all the low level action on the chunk. See `org.apache.hadoop.chukwa.extraction.demux.processor.mapper.Df` for an example of a Map class for use with Demux.

For Chukwa to invoke your Mapper code, you have to specify which data types it should run on. Edit `${CHUKWA_HOME}/conf/chukwa-demux-conf.xml` and add the following lines:

```
      <property>
            <name>MyDataType</name>
<value>org.apache.hadoop.chukwa.extraction.demux.processor.mapper.MyParser</value>
            <description>Parser class for MyDataType.</description>
      </property>
```

You can use the same parser for several different recordTypes.

5.2. Writing a custom reduce

You only need to implement a reduce side if you need to group records together. The interface that your need to implement is `ReduceProcessor`:

```
public interface ReduceProcessor
{
           public String getDataType();
           public void process(ChukwaRecordKey key,Iterator<ChukwaRecord>
values,
                      OutputCollector<ChukwaRecordKey,
                      ChukwaRecord> output, Reporter reporter);
}
```

The link between the Map side and the reduce is done by setting your reduce class into the reduce type: `key.setReduceType("MyReduceClass");`. Note that in the current version of Chukwa, your class needs to be in the package `org.apache.hadoop.chukwa.extraction.demux.processor` See `org.apache.hadoop.chukwa.extraction.demux.processor.reducer.SystemMetrics` for an example of a Demux reducer.

5.3. Output

Your data is going to be sorted by RecordType then by the key field. The default implementation use the following grouping for all records:

1. Time partition (Time up to the hour)
2. Machine name (physical input source)
3. Record timestamp

The demux process will use the recordType to save similar records together (same recordType) to the same directory: `>cluster name>/<record type>/`