**Apache thrift**

The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages

Apache Thrift allows you to define data types and service interfaces in a simple definition file. Taking that file as input, the compiler generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages. Instead of writing a load of boilerplate code to serialize and transport your objects and invoke remote methods, you can get right down to business.

**Rest**

Hadoop provides a [Java native API](http://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/fs/FileSystem.html) to support file system operations such as create, rename or delete files and directories, open, read or write files, set permissions, etc. A very basic example can be found on [Apache wiki](http://wiki.apache.org/hadoop/HadoopDfsReadWriteExample) about how to read and write files from Hadoop. This is great for applications running within the Hadoop cluster but there may be use cases where an external application needs to manipulate HDFS like it needs to create directories and write files to that directory or read the content of a file stored on HDFS. Hortonworks developed an additional API to support these requirements based on standard [REST](http://www.ics.uci.edu/%7Efielding/pubs/dissertation/rest_arch_style.htm) functionalities.

WebHDFS concept is based on HTTP operations like GET, PUT, POST and DELETE. Operations like OPEN, GETFILESTATUS, LISTSTATUS are using HTTP GET, others like CREATE, MKDIRS, RENAME, SETPERMISSIONS are relying on HTTP PUT. APPEND operations is based on HTTP POST, while DELETE is using HTTP DELETE. Authentication can be based on user.name query parameter (as part of the HTTP query string) or if security is turned on then it relies on Kerberos.

**Bulk import in Hbase**

[Apache HBase](http://hbase.apache.org) is all about giving you random, real-time, read/write access to your Big Data, but how do you efficiently get that data into HBase in the first place? Intuitively, a new user will try to do that via the client APIs or by using a MapReduce job with TableOutputFormat, but those approaches are problematic, as you will learn below. Instead, the HBase bulk loading feature is much easier to use and can insert the same amount of data more quickly.

**Overview of Bulk Loading**

If you have any of these symptoms, bulk loading is probably the right choice for you:

* You needed to tweak your MemStores to use most of the memory.
* You needed to either use bigger WALs or bypass them entirely.
* Your compaction and flush queues are in the hundreds.
* Your GC is out of control because your inserts range in the MBs.
* Your latency goes out of your SLA when you import data.

Most of those symptoms are commonly referred to as “growing pains.” Using bulk loading can help you avoid them.  
   
In HBase-speak, bulk loading is the process of preparing and loading [HFiles](http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/) (HBase’s own file format) directly into the RegionServers, thus bypassing the [write path](http://blog.cloudera.com/blog/2012/06/hbase-write-path/) and obviating those issues entirely. This process is similar to ETL and looks like this:

**1. Extract the data from a source, typically text files or another database.**HBase doesn’t manage this part of the process. In other words, you cannot tell HBase to prepare HFiles by directly reading them from MySQL — rather, you have to do it by your own means. For example, you could run mysqldump on a table and upload the resulting files to HDFS or just grab your Apache HTTP log files. In any case, your data needs to be in HDFS before the next step.

**2. Transform the data into HFiles.**This step requires a MapReduce job and for most input types you will have to write the Mapper yourself. The job will need to emit the row key as the Key, and either a KeyValue, a Put, or a Delete as the Value. The Reducer is handled by HBase; you configure it using HFileOutputFormat.configureIncrementalLoad() and it does the following:

* Inspects the table to configure a total order partitioner
* Uploads the partitions file to the cluster and adds it to the DistributedCache
* Sets the number of reduce tasks to match the current number of regions
* Sets the output key/value class to match HFileOutputFormat’s requirements
* Sets the reducer up to perform the appropriate sorting (either KeyValueSortReducer or PutSortReducer)

At this stage, one HFile will be created per region in the output folder. Keep in mind that the input data is almost completely re-written, so you will need at least twice the amount of disk space available than the size of the original data set. For example, for a 100GB mysqldump you should have at least 200GB of available disk space in HDFS. You can delete the dump file at the end of the process.

**3. Load the files into HBase by telling the RegionServers where to find them.**This is the easiest step. It requires using LoadIncrementalHFiles (more commonly known as the [completebulkload](http://hbase.apache.org/book.html#completebulkload) tool), and by passing it a URL that locates the files in HDFS, it will load each file into the relevant region via the RegionServer that serves it. In the event that a region was split after the files were created, the tool will automatically split the HFile according to the new boundaries. This process isn’t very efficient, so if your table is currently being written to by other processes, it’s best to get the files loaded as soon as the transform step is done.

**What is meant by a work flow and what is an oozie workflow.**

Oozie is a server based *Workflow Engine* specialized in running workflow jobs with actions that run Hadoop Map/Reduce and Pig jobs.

Oozie is a Java Web-Application that runs in a Java servlet-container.

For the purposes of Oozie, a workflow is a collection of actions (i.e. Hadoop Map/Reduce jobs, Pig jobs) arranged in a control dependency DAG (Direct Acyclic Graph). "control dependency" from one action to another means that the second action can't run until the first action has completed.

Oozie workflows definitions are written in hPDL (a XML Process Definition Language similar to [JBOSS JBPM](http://www.jboss.org/jbossjbpm/) jPDL).

Oozie workflow actions start jobs in remote systems (i.e. Hadoop, Pig). Upon action completion, the remote systems callback Oozie to notify the action completion, at this point Oozie proceeds to the next action in the workflow.

Oozie workflows contain control flow nodes and action nodes.

Control flow nodes define the beginning and the end of a workflow ( start , end and fail nodes) and provide a mechanism to control the workflow execution path ( decision , fork and join nodes).

Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task. Oozie provides support for different types of actions: Hadoop map-reduce, Hadoop file system, Pig, SSH, HTTP, eMail and Oozie sub-workflow. Oozie can be extended to support additional type of actions.

Oozie workflows can be parameterized (using variables like ${inputDir} within the workflow definition). When submitting a workflow job values for the parameters must be provided. If properly parameterized (i.e. using different output directories) several identical workflow jobs can concurrently.

**Tracking a Oozie job.**

The shell action runs a Shell command.

The workflow job will wait until the Shell command completes before continuing to the next action.

To run the Shell job, you have to configure the shell action with the =job-tracker=, name-node and Shell exec elements as well as the necessary arguments and configuration.

A shell action can be configured to create or delete HDFS directories before starting the Shell job.

Shell *launcher* configuration can be specified with a file, using the job-xml element, and inline, using the configuration elements.

Oozie EL expressions can be used in the inline configuration. Property values specified in the configuration element override values specified in the job-xml file.

Note that Hadoop mapred.job.tracker and fs.default.name properties must not be present in the inline configuration.

As with Hadoop map-reduce jobs, it is possible to add files and archives in order to make them available to the Shell job. Refer to the [WorkflowFunctionalSpec#FilesAchives][Adding Files and Archives for the Job] section for more information about this feature.

The output (STDOUT) of the Shell job can be made available to the workflow job after the Shell job ends. This information could be used from within decision nodes. If the output of the Shell job is made available to the workflow job the shell command must follow the following requirements:

* The format of the output must be a valid Java Properties file.
* The size of the output must not exceed 2KB.

kind of jobs can be scheduled with Oozie.

Oozie has another type of a job called a [coordinator application](http://archive.cloudera.com/cdh4/cdh/4/oozie/CoordinatorFunctionalSpec.html#a1._Coordinator_Overview). Coordinator applications allow users to schedule more complex workflows, including workflows that are scheduled regularly, or that have dependencies on the output from other workflows. For this application, which is stored on an HDFS cluster in a file named add-partition-coord-app.xml, the add partition workflow is executed on an hourly basis:

Oozie includes a number of different methods for specifying frequency intervals, which can be found in the [documentation](http://archive.cloudera.com/cdh4/cdh/4/oozie/CoordinatorFunctionalSpec.html#a4.4._Frequency_and_Time-Period_Representation). You also have to specify a start time and end time for the job, which are represented by the jobStart and jobEndvariables.

the second major component of the coordinator app is the [datasets](http://archive.cloudera.com/cdh4/cdh/4/oozie/CoordinatorFunctionalSpec.html#a5._Dataset) entity. A dataset specifies the location of a set of input data. In this case, there is a dataset called tweets, which is updated every hour, as specified by the frequency. For each execution of the Hive workflow, you will have a separate instance of the tweets dataset, starting with the initial instance specified by the dataset. A particular instance of a dataset is identified by its creation time in [ISO-8601 format](http://en.wikipedia.org/wiki/ISO_8601), such as 2013-01-04T09:00Z.

Having a completed dataset is one of the criteria for executing an instance of the Hive workflow. The other requirement is for any input events to be satisfied. Currently, Oozie is restricted to input events in the form of available datasets. This means that an input event will not be satisfied until a particular instance of a dataset exists. In the context of a more complex data pipeline, this means that a job can be configured to execute only after all of its input data has been successfully generated.

**Oozie coordinator:**

Coordinator applications allow users to schedule complex workflows, including workflows that are scheduled regularly. Oozie Coordinator models the workflow execution triggers in the form of time, data or event predicates. The workflow job mentioned inside the **Coordinator** is started only after the given conditions are satisfied.