THREEGLAV

Threeglav.com

StreamHorizon - User Guide

Version 3.0.2

STREAM HORIZON

DataProcessingPlatform

TABLE OF CONTENTS

INTRODUCTION

GETTING STARTED

INSTALLATION REQUIREMENTS

Installation of StreamHorizon platform is simple and easy. Download installation archive (available at www.threeglav.com), uncompress it to your local hard drive in directory of your choice (we will refer to this directory as $ENGINE_HOME from now on).

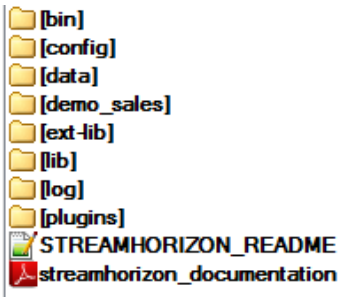
There are very few minimum requirements that need to be met in order to run StreamHorizon.

1. Mainstream operating system (Linux, Windows, Solaris)
2. JDK 1.7+ (we recommend Oracle HotSpot)
3. Database that supports JDBC (most likely data produced by StreamHorizon will go into some kind of database). This is not hard requirement if you only want to produce output files.

NOTE: Even if you intend to run multiple (clustered) StreamHorizon instances there is no need to install binaries more than once per physical machine.

STREAMHORIZON DIRECTORY STRUCTURE

After uncompressing StreamHorizon archive to $ENGINE_HOME directory you will see following directory structure inside:



Directory name	Purpose
$ENGINE_HOME/bin/	All startup scripts should be placed here. There are few default scripts that can be used for starting single instance. If you are adding new startup scripts always place them in this directory.
$ENGINE_HOME/config/	All configuration files are here: • What engine should log and at what level • Here is where you copy your main engine configuration file(s) • Configuration for clustered caches used by engine
$ENGINE_HOME/data/	This is engine private directory used for housekeeping. You should not add, delete or modify anything inside.
$ENGINE_HOME/demo_sales/	Demo feature showing capabilities of engine (performance and functionality).
$ENGINE_HOME/ext-lib/	Additional dependencies needed by plugins.
$ENGINE_HOME/lib/	External dependencies needed by engine are placed here. You should not add, delete or modify anything

	inside this directory.
$ENGINE_HOME/log/	Engine log files will be found here.
$ENGINE_HOME/plugins/	Java plugins. Plain *.java files can be placed here and engine will compile and use them (when configured to do so).

STREAMHORIZON CONFIGURATION FILES

There are multiple configuration files available and all of them can be found in $ENGINE_HOME/config/ directory.

ENGINE CONFIGURATION FILE (ENGINE-CONFIG.XML)

The main configuration file for StreamHorizon engine is $ENGINE_HOME/config/engine-config.xml

This file describes to StreamHorizon engine where to find input data, how to process it and where to send output results.

LOGGING CONFIGURATION

Logging is configured in $ENGINE_HOME/config/logback.xml

Please refer to logback (http://logback.qos.ch/manual/configuration.html) documentation for further details how to change logging levels.

CACHING CONFIGURATION

StreamHorizon uses clustered cache to avoid unnecessary trips to database. It is possible to choose between two clustered cache technologies: Infinispan and Hazelcast. By default Infinispan is used and this is recommended.

Configuration files for each of these cache technologies can be found in $ENGINE_HOME/config/

Please refer to appropriate online documentation to understand how to tune these libraries properly.

MAPPING DIMENSIONS

TODO: explain why we map dimensions, how etc

SUPPORTED DIMENSION TYPES

Explain INSERT_ONLY type

PRECACHING DIMENSION DATA

It is possible to configure every dimension to pre-cache its data from database into cache. This is executed on startup and will significantly speed up execution.

Pre-caching SQL statement is configured in <preCacheRecords>. It is very important that this statement returns primary key first and then a list of all mapped natural keys, in the order specified in <mappedColumns> for that particular dimension.

PROCESSING INPUT DATA (FEEDS)

StreamHorizon engine is configured to watch particular directory on disk (<sourceDirectory>) and process files found there. Different feeds can decide to watch for different files in the same directory and this is achieved by setting file name regular expression in <fileNameMasks>.

It is safe for multiple threads to watch the same source directory and all threads will evenly partition work. This feature works out of the box.

It is also possible to configure multiple StreamHorizon engine instances to watch the same source directory, but this requires explicit configuration.

All ETL threads watch source directory, try to match them against their file name pattern and files that pass criteria are processed in a round-robin fashion by one of available threads in ETL threadpool.

Files are processed as configured in engine-config.xml file.

StreamHorizon engine will not start processing any file that was created or modified in the last 2 seconds. This is to ensure data consistency and avoid reading file someone is writing to. This delay is configurable and if your file system supports atomic file move operations you can decrease this value.

CHECKING INTEGRITY OF INPUT FILES

TODO: explain footer, header and control characters

CREATING OUTPUT FILES

The result of processing input data files can be written directly to disk (location is configured by using <bulkOutputDirectory> tag). Files written to disk are always text files. When writing files to disk it is important to define extension for final output file and delimiter to be used when outputting data values.

All input data files are processed by ETL thread pool.

Here is configuration sample how to output results of processing directly to disk

```
…
<bulkLoadDefinition outputType="file">
        <bulkLoadOutputExtension>ext1</bulkLoadOutputExtension>
        <bulkLoadFileDelimiter>,</bulkLoadFileDelimiter>
…
```

All input files (if successfully processed) will be output to <bulkOutputDirectory> with extension **ext1** and resolved values will be delimited using comma separator. Every line in input file will have corresponding line in output file.

Output files are created as temporary (with different extension) and atomically renamed only after all data has successfully been written. This prevents threads in DB pool (or any other process) to read half-written files.

LOADING OUTPUT DATA FILES INTO DATABASE

It is possible to configure StreamHorizon to load data from output file into database. In this case DB threads will watch folder <bulkOutputDirectory> for files with given extension (<bulkLoadOutputExtension>) and will execute command specified in <bulkLoadInsert>. It is possible to specify SQL or shell command that will be used to load bulk files.

For example, this can be used to load data to Oracle (external tables), to execute shell script to invoke MSSQL to load data or let MySQL execute LOAD DATA INFILE statement.

INSERTING OUTPUT DATA DIRECTLY INTO DATABASE (VIA JDBC)

It is possible to configure StreamHorizon to insert output data into database using JDBC interface. This avoids generation of interim data files on disk and can help reducing IO. Processing logic and steps are exactly the same as if data is written to disk; the only difference is that loading happens in batches as data is being processed. Batch size can be configured.

ERROR HANDLING

TODO: Describe error handling onCompletion/onFailure/onSuccess

By default all warnings and errors are written to corresponding log files in $ENGINE_HOME/log/ directory. This happens in case when some of user-defined SQL statements are invalid, data format is not correctly described in engine-config.xml or similar cases.

CONFIGURATION PARAMETERS

Almost all parts of StreamHorizon can be configured by supplying configuration parameters. It is possible to turn off certain features (like remote commands, local caches) or tune them to behave differently (for example use Hazelcast instead of Infinispan for caching, use different read buffer size etc).

Configuration parameters can be specified in engine configuration file and can also be overridden by providing Java system parameters with the same name. This is useful when multiple instances of StreamHorizon want to perform processing in the same way but only some parts need to be different. In this case engine configuration is template for processing, containing default tuning, and every instance overrides some of parameters by specifying them in startup scripts.

CONTEXT ATTRIBUTES

Context attributes are very simple extension points of StreamHorizon engine. They can originate from StreamHorizon engine itself (implicit context attributes) and from Java plugins provided by users.

Context attributes can be used in engine configuration files and in Java plugins.

Some attributes are available only during execution of ETL threads, some only during execution of DB threads and some attributes are available in all cases.

Context attributes can be used in engine configuration file by reference – enclosed in ${ and } characters. Engine will recognize these and replace them with appropriate values during execution. For example ${ engineInstanceStartTimestamp} will always be replaced with time when that particular StreamHorizon engine was started (Unix timestamp).

Java plugins get all the context attributes passed as one of parameters passed to their methods and they can change existing attributes or add new ones.

If some attribute is not available then NULL value will be passed to all SQL commands using it and Java null value will be passed to all Java plugins trying to look it up.

Implicitly provided context attributes are:

Attribute name	Description	Available in
engineInstanceStartTimestamp		ETL,DB
engineInstanceIdentifier		ETL,DB
bulkProcessingThreadID		DB
feedProcessingThreadID		ETL
feedInputFilePath		ETL
feedInputFileName		ETL
feedInputFileReceivedTimestamp		ETL
feedInputfileReceivedDateTime		ETL
feedInputFileProcessingStartedTimestamp		ETL
feedInputFileProcessingFinishedTimestamp		ETL
feedInputfileProcessingStartedDateTime		ETL
feedInputFileSize		ETL
feedBulkLoadOutputFilePath		ETL
bulkFilePath		ETL,DB
bulkFileAlreadySubmittedForLoading		DB
bulkFileName		ETL,DB
bulkFileReceivedForProcessingTimestamp		DB
bulkFileProcessingStartedTimestamp		DB
feedInputFileJdbcInsertStartedTimestamp		ETL (with JDBC output)
feedInputFileJdbcInsertFinishedTimestamp		ETL (with JDBC output)
bulkFileProcessingFinishedTimestamp		DB
bulkCompletionProcessingSuccessFailureFlag		DB
bulkCompletionProcessingErrorDescription		DB
feedCompletionProcessingSuccessFailureFlag		DB
feedCompletionNumberOfTotalRowsInFeed		DB
feedCompletionProcessingErrorDescription		DB

CREATING PLUGINS

StreamHorizon platform provides extension points where users can add their own processing logic written in SQL, Java or shell scripts. This enables to interact with StreamHorizon engine and customize processing steps and actions to be taken.

SQL AND SHELL COMMANDS

SQL and shell commands can be used at certain points in engine-config.xml file. All those commands can use context attributes (implicit or user defined).

It is the responsibility of user to ensure correctness of those commands and speed of execution.

JAVA PLUGINS

Java plugins are user provided Java classes implementing one of interfaces in StreamHorizon API. These plugins can be compiled, packaged as jar files and added to $ENGINE_HOME/ext-lib/ directory or source code (.java files) can be copied to $ENGINE_HOME/plugins/ directory where StreamHorizon engine will compile and use them.

Currently there are three types of plugins:

1. User-defined file name parser – must implement com.threeglav.sh.bauk.feed.FeedFileNameProcessor interface
2. User-defined input feed header processor – must implement com.threeglav.sh.bauk.header.HeaderParser interface
3. User-defined input feed data row processor – must implement com.threeglav.sh.bauk.feed.FeedDataLineProcessor interface

CLUSTERING STREAMHORIZON

It is possible to run multiple StreamHorizon engine instances (multiple JVMs) and let them partition processing and data loading. It is not requirement to run all StreamHorizon instances on the same physical machine. Minimum requirement is to have networking (TCP or UDP) correctly setup between all clustered instances.

There are multiple ways to achieve clustering of StreamHorizon engine instances.

STREAMHORIZON INSTANCES ARE INDEPENDENT

This is easiest way where multiple instances are started and all of them point to different source directories and process files independently. There is no need for any partitioning or interaction between StreamHorizon instances in this case.

STREAMHORIZON INSTANCES SHARE SAME SOURCE DIRECTORY

If it is needed for multiple StreamHorizon instances to watch same source directory then it is very important to partition work between engine instances correctly. Otherwise different instances can start processing same files.

In order to do this it is important to assign unique integer identifier to every StreamHorizon engine instance and also set a total number of instances running. This is all achieved by using customized startup script and is shown in demo_sales/local_cluster_example/.

It is recommended to contact StreamHorizon support for more details.

STREAMHORIZON INSTANCES SHARE DIMENSION CACHES

This option can be combined with previous option (sharing same source directory).

In order to minimize access to database it can be useful to share cached data between multiple StreamHorizon engine instances. This is achieved by configuring caching to be shared (replicated or distributed). Please contact StreamHorizon support for more details.

PERFORMANCE TUNING

TUNING OUTPUT TYPE

TODO: explain how output type affects performance and how to choose types

TUNING DATABASE

TODO: explain how to tune database, connection pool, depending on output type used

TUNING THREAD POOLS

StreamHorizon uses two different thread pools. First thread pool (ETL) is used for processing input feed files and the second one (DB) is used to load bulk files. In case when data is inserted into database using JDBC then ETL thread pool will execute that logic.

It is recommended to set number of threads to a number very close to number of cores your machine has. It is always recommended to thoroughly test your configuration.

STORAGE, READ AND WRITE BUFFERS

TODO: explain how to tune read/write buffers and when to do this

DIMENSION CACHES

There are two levels of caching used by StreamHorizon engine.

1. Per-thread dimension cache
2. Local dimension cache
3. Distributed dimension cache (Infinispan or Hazelcast)

Tuning these caches properly can reduce memory consumption and significantly speed-up execution.

TODO: explain tuning each cache here.

REMOTE COMMANDS

StreamHorizon engine accepts remote commands via HTTP. This feature can be turned off. Currently we support following commands:

FLUSHING DIMENSION CACHE REMOTE COMMAND

In case when dimension cache is stale (for example data was loaded into table bypassing StreamHorizon engine) it is important to inform engine to stop using old data. This can be done by executing

http://localhost:<remoting.server.port>/flushDimensionCache/?dimension=DIMENSION_NAME_AS_DEFINED_IN_CONFIG

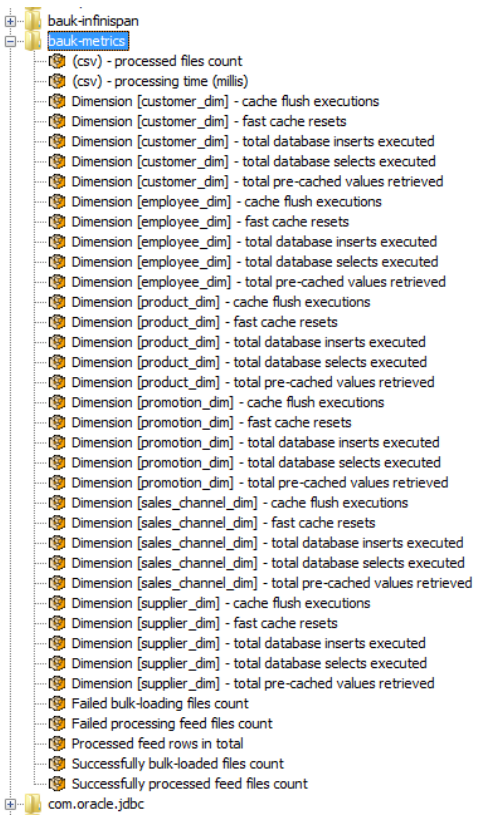
When this is executed engine will go through a set of steps:

1. Wait for all threads to finish their current work
2. Pause all threads so that they do not accept new work
3. Flush data cached for specified dimension
4. Notify all threads that they can continue processing data

Please note that after flushing dimension cache there is expected slow-down in execution because cache will be populated as needed and this might require execution of database queries.

MONITORING

StreamHorizon exposes internal processing metrics via JMX.



It is also possible to use extension points to collect and calculate metrics about performance of engine. See the sales demo provided with installation for more details (sh_metrics table).

MISCELLANEOUS

COMMON MISTAKES

TODO: Mladen da objasni sta je sve pokusavao, to nije radilo pa je mislio da je bug :P

PLANNED PLATFORM EXTENSIONS

There are many planned extension to StreamHorizon. Here are only few of them:

- Accepting streams and being able to process them (no need to read data from disk)
- Being able to write output file in format different than plain text (we plan to make this pluggable so that customers can plug-in their own format)
- More extension points

RUNNING STREAMHORIZON DEMO

ORACLE DEMO

Oracle deployment comes in two flavours, JDBC and external tables. JDBC performs slower than external tables generally, however setup script for external tables requires you to create Oracle directories (as instructed in execution script).

TO TEST ORACLE JDBC DEPLOYMENT:

1. Move StreamHorizon/demo_sales/oracle/jdbc_deploy/engine-config.xml file into StreamHorizon/config/ folder.
2. Open engine-config.xml file and change parameters commented out with string "SET ME!"
3. Execute Oracle script StreamHorizon/demo_sales/oracle/jdbc_deploy/oracle_create_schema.sql. Please read NOTE's at the beginning of the script prior to execution
4. Copy sample file XXXXXXXXXXX and file multiplier script XXXXXXX (bat or sh) to your <sourceDirectory> as stated in engine-config.xml. Execute XXXXXXX script in order to create 100 files.
5. NOTE: by default data in fact table (sales_fact) will be stored in your default tablespace. This tablespace (if not big enough) may be filled up during processing and if so will cause errors in StreamHorizon logs (StreamHorizon/log/sh-engine*log). To avoid this either edit file multiplier script XXXXXX to create less than 1000 copies of the feed files or extend your default tablespace or change create table statement for sales_fact table to point to tablespace of your choice.
6. Start StreamHorizon instance
7. Use select * from sh_dashboard to see throughput of the instance
8. To see detail performance logs look at sh_metrics table
9. To troubleshoot please look at StreamHorizon/log/sh-engine*log
10. Please read "Realistic Testing" section below

TO TEST ORACLE IN EXTERNAL TABLE (BULK LOAD) MODE

1. Move StreamHorizon/demo_sales/oracle/ext_table_deploy/engine-config.xml into StreamHorizon/config/ folder.
2. Open engine-config.xml and change parameters commented out with string "SET ME!"
3. Execute Oracle script StreamHorizon/demo_sales/oracle/ext_table_deploy/oracle_create_schema.sql. Please read NOTE's at the beginning of the script prior to execution
4. If you have already executed StreamHorizon/demo_sales/oracle/ext_table_deploy/oracle_create_schema.sql as part of install of JDBC setup you do not have to do step (3) again!
5. Execute Oracle script StreamHorizon/demo_sales/oracle/ext_table_deploy/externalTableCreate.sql. Please read NOTE's at the beginning of the script prior to execution
6. Copy sample file XXXXXXXXXXX and file multiplier script XXXXXXX (bat or sh) to your <sourceDirectory> as stated in engine-config.xml. Execute XXXXXXX script in order to create 100 files.
7. NOTE: by default data in fact table (sales_fact) will be stored in your default tablespace. This tablespace (if not big enough) may be filled up during processing and if so will cause errors in StreamHorizon logs (StreamHorizon/log/sh-engine*log). To avoid this either edit file multiplier script XXXXXX to create less than 1000 copies of the feed files or extend your default tablespace or change create table statement for sales_fact table to point to tablespace of your choice.
8. Start StreamHorizon instance
9. Use select * from sh_dashboard to see throughput of the instance
10. To see detail performance logs look at sh_metrics table
11. To troubleshoot please look at StreamHorizon/log/sh-engine*log
12. Please read "Realistic Testing" section below

REALISTIC TESTING

As first run of StreamHorizon will form all cardinalities of the all dimensions from scratch average throughput will be 3 times slower than usual.

After first successful run, after dimensions are repopulated kill instance of StreamHorizon. Copy files back from <archiveDirectory> to <sourceDirectory>. Start instance of StreamHorizon again and observe the throughput.

Note that throughput should be observed by running 'select * from sh_dashboard' query against schema you have created stream by using scripts supplied. View sh_dashboard will only show throughput for data loaded since last startup.

NOTE: if you run one set of files, than pause let's say an hour (don't kill StreamHorizon instance) and then run another batch of files view sh_dashboard will show very low throughput. This is because view calculates time window of first file processed by instance after startup and last file processed since instance startup. If batch was loaded in 1 minute, if idle time of server server was 98 minutes, and if second batch was running for 1 minute, throughput shown by sh_dashboard will be 2% of actual throughput of the StreamHorizon.

To avoid this simply observe throughput during the batch immediately after all files are loaded and make sure you restart StreamHorizon before next test.

StreamHorizon increases data throughput as batch progresses.

MYSQL DEMO

MySQL deployment comes in two flavours, JDBC and reading rows from a text file (LOAD DATA INFILE).

It is very important to understand how to tune MySQL database engine for high throughput and bulk loading. Please refer to official MySQL documentation.

TO TEST MYSQL JDBC DEPLOYMENT:

1. Move StreamHorizon/demo_sales/mysql/engine-config.xml file into StreamHorizon/config/ folder.
2. Open engine-config.xml file and change parameters commented out with string "SET ME!"
3. Execute script StreamHorizon/demo_sales/mysql_create_schema.sql.
4. Copy sample file XXXXXXXXXXX and file multiplier script XXXXXXX (bat or sh) to your <sourceDirectory> as stated in engine-config.xml. Execute XXXXXXX script in order to create 100 files.
5. Start StreamHorizon instance
6. To see detail performance logs look at sh_metrics table
7. To troubleshoot please look at StreamHorizon/log/sh-engine*log
8. Please read "Realistic Testing" section below

TO TEST MYSQL IN EXTERNAL TABLE (BULK LOAD) MODE

1. Execute first four steps needed for running MySQL JDBC demo (see above)
2. In StreamHorizon/config/engine-config.xml change <bulkLoadInsert> so that it uses first command (with LOAD DATA INFILE syntax)
3. Change <bulkLoadDefinition outputType="jdbc"> to <bulkLoadDefinition outputType="file">
4. Increase value for <databaseProcessingThreadCount>0</databaseProcessingThreadCount>
5. Copy sample file XXXXXXXXXXX and file multiplier script XXXXXXX (bat or sh) to your <sourceDirectory> as stated in engine-config.xml. Execute XXXXXXX script in order to create 100 files.
6. Execute remaining steps (5-8) needed for running MySQL JDBC demo (see above)

REALISTIC TESTING

As first run of StreamHorizon will form all cardinalities of the all dimensions from scratch average throughput will be 3 times slower than usual.

After first successful run, after dimensions are repopulated kill instance of StreamHorizon. Copy files back from <archiveDirectory> to <sourceDirectory>. Start instance of StreamHorizon again and observe the throughput.

StreamHorizon increases data throughput as batch progresses