Rubix Documentation

Release 0.2.12

Qubole

Nov 24, 2020

Contents:

1 RubiX **3**
- 1.1 Use Case . . . 3
- 1.2 Supported Engines and Cloud Stores . . . 3

2 Installation Guide **5**
- 2.1 Getting Started . . . 5
- 2.2 Data Engines . . . 7

3 Configuration **11**
- 3.1 Cache . . . 12
- 3.2 Network . . . 13
- 3.3 Cluster . . . 13
- 3.4 Metrics . . . 14

4 Metrics **15**
- 4.1 BookKeeper Server Metrics . . . 15
- 4.2 Client side Metrics . . . 17

5 Contribution Guidelines **19**
- 5.1 Developer Environment . . . 19
- 5.2 How to contribute code on Github . . . 20
- 5.3 Coding conventions . . . 21
- 5.4 Testing . . . 21
- 5.5 Commit Message . . . 26
- 5.6 How to report issues . . . 28
- 5.7 Documentation Style Guide . . . 28

6 Release Notes **29**
- 6.1 Release 0.3.10 . . . 29
- 6.2 Release 0.3.11 . . . 29
- 6.3 Release 0.3.12 . . . 29
- 6.4 Release 0.3.13 . . . 30
- 6.5 Release 0.3.14 . . . 30
- 6.6 Release 0.3.15 . . . 30
- 6.7 Release 0.3.16 . . . 31
- 6.8 Release 0.3.17 . . . 31
- 6.9 Release 0.3.18 . . . 31

6.10 Release 0.3.19 31
6.11 Release 0.3.20 32
6.12 Release 0.3.21 (next release) 32

RubiX is a light-weight data caching framework that can be used by Big-Data engines. RubiX can be extended to support any engine that accesses data in cloud stores using Hadoop FileSystem interface via plugins. Using the same plugins, RubiX can also be extended to be used with any cloud store.

CHAPTER 1

RubiX

RubiX is a light-weight data caching framework that can be used by Big-Data engines. RubiX can be extended to support any engine that accesses data in cloud stores using Hadoop FileSystem interface via plugins. Using the same plugins, RubiX can also be extended to be used with any cloud store

1.1 Use Case

RubiX provides disk or in-memory caching of data, which would otherwise be accessed over network when it resides in cloud store, thereby improving performance.

1.2 Supported Engines and Cloud Stores

1.2.1 Amazon S3

- Presto
- Spark
- Any engine using Hadoop 2.x (e.g. Hive)

CHAPTER 2

Installation Guide

This section provides instructions to install RubiX and use it with Presto, Hive, or Spark. If you want help to install on-prem or on other distributions, please contact the community with your questions or issues.

These instructions require RubiX Admin to be installed on your cluster.

2.1 Getting Started

Note: Make sure passwordless SSH is set up for your cluster before starting.

2.1.1 Install RubiX

Install RubiX Admin using PIP:

```
pip install rubix_admin
```

Run `rubix_admin -h` to generate a config file for RubiX Admin at `~/.radminrc`. Add the worker node IPs to the **workers** list so the file looks like the following:

```
coordinator:
  - localhost
workers:
  - <worker-ip1>
  - <worker-ip2>
  ..
remote_packages_path: /tmp/rubix_rpms
```

Once RubiX Admin is configured, install the latest version of RubiX on all nodes specified in `~/.radminrc`:

```
rubix_admin installer install --cluster-type <type>
```

To install a specific version of RubiX:

```
rubix_admin installer install --cluster-type <type> --rpm-version <rubix-version>
```

To install from an RPM file:

```
rubix_admin installer install --cluster-type <type> --rpm <path-to-rubix-rpm>
```

2.1.2 Start RubiX Daemons

Use the following command to start the BookKeeperServer and LocalDataTransferServer on all nodes specified in `~/.radminrc`:

```
rubix_admin daemon start
```

To verify that the daemons are running, run the following command on each node:

```
sudo jps -m
```

You should see the following two entries in the resulting list:

```
<pid> RunJar ... com.qubole.rubix.bookkeeper.BookKeeperServer
<pid> RunJar ... com.qubole.rubix.bookkeeper.LocalDataTransferServer
```

If there was an issue starting the daemons, logs for RubiX can be found at `/var/log/rubix/`.

2.1.3 Configure engine to use RubiX

Presto

In order for Presto to use RubiX, you will first need to create an external table through Hive using RubiX as the URL scheme in place of S3.

Start Hive with the following command. This will restart the metastore server, allowing the *rubix://* scheme to be recognized:

```
hive --hiveconf hive.metastore.uris="" \
     --hiveconf fs.rubix.impl=org.apache.hadoop.fs.s3a.S3AFileSystem
```

You will also need to set your AWS access & secret keys for authenticating with S3:

```
hive> set fs.s3n.awsAccessKeyId=<access-key>
hive> set fs.s3n.awsSecretAccessKey=<secret-key>
```

Once this is done, create your external table, but specify `rubix://` instead of `s3://` as the URL scheme:

```
CREATE EXTERNAL TABLE...
...
LOCATION 'rubix://<s3-path>'
```

Once your table is created, it will now be configured to use RubiX.

Spark

In order to use Spark with S3, you will need to specify your AWS access & secret keys when running your application:

```
...
--conf spark.hadoop.fs.s3.awsAccessKeyId=<access-key>
--conf spark.hadoop.fs.s3.awsSecretAccessKey=<secret-key>
...
```

Alternatively, you can add the following lines to your Spark properties file to set them for every application (default location: `$SPARK_HOME/conf/spark-defaults.conf`):

```
spark.hadoop.fs.s3.awsAccessKeyId       <access-key>
spark.hadoop.fs.s3.awsSecretAccessKey   <secret-key>
```

Note:

RubiX client configurations will also need to be set this way.
(Format: `spark.hadoop.<rubix-conf-key>`)

2.1.4 Run your first RubiX-enhanced query

Once you have properly configured your data engine, RubiX will now cache data when it is being fetched from S3.

You can verify this in the logs for your data engine, which should show usage of a Caching... S3FileSystem, as well as in the BookKeeper logs at `/var/log/rubix/bks.log`.

Troubleshooting

ClassNotFoundException: org.apache.hadoop.fs.s3native.NativeS3FileSystem

Hadoop requires the `hadoop-aws` JAR in order to access files stored on S3. If Hadoop is unable to find the `NativeS3FileSystem` class, make sure this JAR is included in your Hadoop classpath. This JAR should be provided as part of your Hadoop installation.

Check Hadoop's S3 documentation for more details.

ClassNotFoundException: org.jets3t.service.ServiceException

Spark requires JetS3t in order to execute applications using S3. If Spark is unable to find this class, make sure `jets3t-x.x.x.jar` is included in `spark.driver.extraClassPath` and `spark.executor.extraClassPath` in your Spark configuration. This JAR should be provided as part of your Hadoop installation.

2.2 Data Engines

This section provides additional information for using RubiX with various data engines.

These instructions require that RubiX Admin has been installed on your cluster.

2.2.1 Presto

Enable RubiX Caching for Table

In order for Presto to use RubiX, you will first need to create an external table through Hive using RubiX as the URL scheme in place of S3.

Start Hive with the following command. This will restart the metastore server, allowing the *rubix://* scheme to be recognized:

```
hive --hiveconf hive.metastore.uris="" \
     --hiveconf fs.rubix.impl=org.apache.hadoop.fs.s3a.S3AFileSystem
```

You will also need to set your AWS access & secret keys for authenticating with S3:

```
hive> set fs.s3n.awsAccessKeyId=<access-key>
hive> set fs.s3n.awsSecretAccessKey=<secret-key>
```

Once this is done, create your external table, but specify `rubix://` instead of `s3://` as the URL scheme:

```
CREATE EXTERNAL TABLE...
...
LOCATION 'rubix://<s3-path>'
```

Once your table is created, it will now be configured to use RubiX.

2.2.2 Spark

Configuration

In order to use Spark with S3, you will need to specify your AWS access & secret keys when running your application:

```
...
--conf spark.hadoop.fs.s3.awsAccessKeyId=<access-key>
--conf spark.hadoop.fs.s3.awsSecretAccessKey=<secret-key>
...
```

Alternatively, you can add the following lines to your Spark properties file to set them for every application (default location: `$SPARK_HOME/conf/spark-defaults.conf`):

```
spark.hadoop.fs.s3.awsAccessKeyId       <access-key>
spark.hadoop.fs.s3.awsSecretAccessKey   <secret-key>
```

Note:

RubiX client configurations will also need to be set this way.
(Format: `spark.hadoop.<rubix-conf-key>`)

2.2.3 Hive

Add RubiX JARs to Hadoop Classpath

Add RubiX JARs through the Hive CLI:

```
hive> add jar /usr/lib/rubix/lib/rubix-hadoop2.jar
hive> add jar /usr/lib/rubix/lib/rubix-common.jar
hive> add jar /usr/lib/rubix/lib/rubix-core.jar
hive> add jar /usr/lib/rubix/lib/rubix-spi.jar
```

OR

Copy JARs to Hadoop's shared lib directory:

```
cp /usr/lib/rubix/lib/rubix-hadoop2.jar \
   /usr/lib/rubix/lib/rubix-common.jar \
   /usr/lib/rubix/lib/rubix-core.jar \
   /usr/lib/rubix/lib/rubix-spi.jar \
   /usr/lib/hadoop/lib/share/hadoop/tools/lib
```

Configure Hive to use RubiX CachingFileSystem for S3/S3N/S3A schemes

If you use this option, all file system interactions with tables with their location in AWS S3 will automatically use RubiX's CachingFileSystem.

```
hive --hiveconf \
    fs.s3.impl=com.qubole.rubix.hadoop2.CachingNativeS3FileSystem \
    fs.s3n.impl=com.qubole.rubix.hadoop2.CachingNativeS3FileSystem \
    fs.s3a.impl=com.qubole.rubix.hadoop2.CachingS3AFileSystem
```

CHAPTER 3

Configuration

3.1 Cache

Option	Description	Type	Default	Client/Server	Applicable to Embedded mode
rubix.cache.block.size	The amount of data downloaded per block requested for caching. (if block size = 10MB, request for 45MB of data will download 5 blocks of 10MB)	integer (bytes)	1048576 (1MB)	C & S	Yes
rubix.cache.dirprefix.list	The list of directories to be used as parents for storing cache files. Example: **/media/ephemeral**0/fcache/	list (comma-separated)	/media/ephemeral	C & S	No
rubix.cache.max.disks	The number of (zero-indexed) disks within the parent directory to be used for storing cached files. Example: /media/ephemeral**0** to /media/ephemeral**4**	integer	5	C & S	No
rubix.cache.dirsuffix	The name of the subdirectory to be used for storing cache files. Example: /media/ephemeral0**/fcache/**	string	/fcache/	C & S	Yes
rubix.cache.expiration.after-write	The time after which files will be kept in cache prior to eviction.	integer (ms)	MAX_VALUE	S	Yes
rubix.cache.usage.check.max.percentage	The percentage of the disk space that will be filled with cached data before cached files will start being evicted.	integer (%)	80	S	Yes
rubix.cache.strict.mode	Propagate exceptions if there is an error while caching data if true, otherwise fall back on reading data directly from remote file system.	boolean	false	C	No
rubix.cache.file.staleness-check.enable	When true, always check for updates to file metadata from remote filesystem. When false, file metadata will be cached for a period of time before being fetched again.	boolean	true	S	Yes
rubix.cache.staleness.ttl	(**rubix.cache.file.staleness-check.enable** must be false) The time file metadata will be cached before it will be fetched again from the remote filesystem.	integer (s)	36000	S	Yes
rubix.cache.parallel.warmup	When true, cache will be warmed up asynchronously.	boolean	false	C & S	No
rubix.cache.dummy.mode	When true, the cache is not populated with data and queries are handled directly from the source, but metadata is updated so	boolean	false	C	Yes

3.2 Network

Option	Description	Type	Default	Client/Server	Applicable to Embedded mode
rubix.network.bookkeeper.server.port	The port on which the BookKeeper server is listening	integer	8899	C & S	No
rubix.network.local.transfer.server.port	The port on which the Local Data Transfer server is listening.	integer	8898	C	No
rubix.network.client.num.retries	The maximum number of retry attempts for executing calls to the BookKeeper server.	integer	3	C & S	Yes
rubix.network.server.connect.timeout	The maximum time to wait for a connection to the BookKeeper server.	integer (ms)	1000	C & S	Yes
rubix.network.server.socket.timeout	The maximum time to wait for a response to requests sent to the BookKeeper server.	integer (ms)	3000	C & S	Yes
rubix.network.client.read.timeout	The maximum time to wait when reading data from another node.	integer (ms)	3000	C	Yes

3.3 Cluster

Option	Description	Type	Default	Client / Server	Applicable to Embedded mode
rubix.cluster.node.refresh.time	The frequency at which the cluster nodes membership will be checked.	integer (s)	300 sec	C & S	Yes
rubix.cluster.manager.for.hadoop.class	The `ClusterManager` class to use for fetching node-related information for Hadoop clusters.	string	com.qubole.rubix.hadoop2.Hadoop2ClusterManager	C & S	No
rubix.cluster.manager.for.presto.class	The `ClusterManager` class to use for fetching node-related information for Presto clusters.	string	com.qubole.rubix.presto.PrestoClusterManager	C & S	No

3.4 Metrics

Option	Description	Type	Default	Client / Server
rubix.metrics.cache.enabled	Collect cache-level metrics if true.	boolean	true	S
rubix.metrics.health.enabled	Collect heartbeat metrics if true.	boolean	true	S
rubix.metrics.jvm.enabled	Collect JVM-level metrics if true.	boolean	false	S
rubix.metrics.reporters	The reporters to be used for collecting metrics. Options: JMX, GANGLIA	list (comma-separated)	JMX,GANGLIA	S
rubix.metrics.reporting.interval	The interval at which all registered reporters will report their metrics.	integer (ms)	10000	S
rubix.metrics.ganglia.host	The host at which the Ganglia server (gmond) is running.	string	127.0.0.1 (localhost)	S
rubix.metrics.ganglia.port	The port on which the Ganglia server (gmond) is listening.	integer	8649	S

CHAPTER 4

Metrics

4.1 BookKeeper Server Metrics

These metrics are available on the BookKeeper server.

Health Metrics

Metrics relating to daemon & service health.

Metric	Description	Abnormalities
rubix.bookkeeper.gauge.live_workers	The number of workers currently reporting to the master node.	Mismatch with number reported by engine (Presto, Spark, etc.)
rubix.bookkeeper.gauge.caching_validated_workers	The number of workers reporting caching validation success.	Mismatch with live worker count (one or more workers failed validation)

Cache Metrics

Metrics relating to cache interactions.

Metric	Description	Abnormalities
rubix.bookkeeper.gauge.cache_size_mb	The current size of the local cache in MB.	Cache size is bigger than configured capacity
rubix.bookkeeper.gauge.available_cache_size_mb	The current disk space available for cache in MB.	
rubix.bookkeeper.count.cache_eviction	The number of files removed from the local cache due to size constraints.	No cache evictions & cache has exceeded configured capacity
rubix.bookkeeper.count.cache_invalidation	The number of files invalidated from the local cache when the source file has been modified.	
rubix.bookkeeper.count.cache_expiry	The number of files removed from the local cache once expired	
rubix.bookkeeper.gauge.cache_hit_rate	The percentage of cache hits for the local cache.	Cache hit rate near 0%
rubix.bookkeeper.gauge.cache_miss_rate	The percentage of cache misses for the local cache.	Cache miss rate near 100%
rubix.bookkeeper.count.total_request	The total number of requests made to read data.	
rubix.bookkeeper.count.cache_request	The number of requests made to read data cached locally.	No cache requests made
rubix.bookkeeper.count.nonlocal_request	The number of requests made to read data from another node.	No non-local requests made
rubix.bookkeeper.count.remote_request	The number of requests made to download data from the data store	No remote requests made
rubix.bookkeeper.count.total_async_request	The total number of requests made to download data asynchronously	
rubix.bookkeeper.count.processed_async_request	The total number of asynchronous download requests that have been processed.	
rubix.bookkeeper.gauge.async_queue_size	The current number of queued asynchronous download requests.	High queue size (requests not being processed)
rubix.bookkeeper.count.async_download	The amount of data asynchronously downloaded, in MB. (When there are no cache evictions, this should match `cache_size_mb`.)	
rubix.bookkeeper.count.async_download_time	Total time spent on downloading data in sec	

JVM Metrics

Metrics relating to JVM statistics, supplied by the Dropwizard Metrics `metrics-jvm` module. (https://metrics.dropwizard.io/3.1.0/manual/jvm/)

Metric	Description	Abnormalities
rubix.bookkeeper.jvm.gc.* rubix.ldts.jvm.gc.*	Metrics relating to garbage collection (GarbageCollectorMetricSet)	
rubix.bookkeeper.jvm.memory.* rubix.ldts.jvm.memory.*	Metrics relating to memory usage (MemoryUsageGaugeSet)	
rubix.bookkeeper.jvm.threads.* rubix.ldts.jvm.threads.*	Metrics relating to thread states (CachedThreadStatesGaugeSet)	

4.2 Client side Metrics

These metrics are available on the client side i.e. Presto or Spark where the jobs to read data are run.

Client side Metrics is divided into two:

1. Basic stats: These stats are available under name *rubix:name=stats*
2. Detailed stats: These stats are available under name *rubix:name=stats,type=detailed*

If Rubix is used in embedded mode, an engine specific suffix is added to these names, e.g., Presto adds *catalog=<catalog_name>* suffix.

Following sections cover the metrics available under both these types in detail.

Basic stats

Metric	Description
mb_read_from_cache	Data read from cache by the client jobs
mb_read_from_source	Data read from Source by the client jobs
cache_hit	Cache Hit ratio, between 0 and 1

Detailed Stats

Data unit in all metrics above is MB

CHAPTER 5

Contribution Guidelines

This section provides guidelines to contribute to the project through code, issues and documentation.

5.1 Developer Environment

Rubix is a Maven project and uses Java 8. It uses JUnit as the testing framework. Ensure that you have a development environment that support the above configuration.

5.1.1 Pre-requisites

- `thrift` binary needs to be available at `/usr/local/bin/thrift`. Rubix will not compile with the newer versions of thrift, it is recommended to install thrift version 0.9.3 by downloading the source from here and installing it using the steps mentioned here
- Java JDK 8 needs to be used. If you see an error like `Fatal error compiling: invalid target release: 1.8` during compilation then setup your system to use Java JDK 8 for the build.
- For generating the RPM you need the `rpmbuild` command available. On Debian-based systems `sudo apt-get install rpmbuild` and on RPM-based systems `sudo yum install rpm-build` make it available.

5.1.2 Building

- Fork your own copy of RubiX into your github account by clicking on the "Fork" button
- Navigate to your account and clone that copy to your development box

  git clone https://github.com/<username>/rubix
- Run tests in the RubiX root directory.

  mvn test

- Add Qubole RubiX as upstream

  git remote add upstream https://github.com/qubole/rubix.git git fetch upstream

5.2 How to contribute code on Github

5.2.1 1. Create a branch and start working on your change.

```
cd rubix
git checkout -b new_rubix_branch
```

5.2.2 2. Code

- Adhere to code standards.
- Include tests and ensure they pass.
- Add release notes, if required, in (next-release) rst file under `docs/release/release_notes/`

5.2.3 3. Commit

For every commit please write a short (max 72 characters) summary in the first line followed with a blank line and then more detailed descriptions of the change.

Don't forget a prefix!

More details in Commit Guidelines

5.2.4 4. Update your branch

```
git fetch upstream
git rebase upstream/master
```

5.2.5 5. Push to remote

```
git push -u origin new_rubix_branch
```

5.2.6 6. Issue a Pull Request

- Navigate to the Rubix repository you just pushed to (e.g. https://github.com/your-user-name/rubix)
- Click *Pull Request*.
- Write your branch name in the branch field (this is filled with *master* by default)
- Click *Update Commit Range*.
- Ensure the changesets you introduced are included in the *Commits* tab.
- Ensure that the *Files Changed* incorporate all of your changes.

- Fill in some details about your potential patch including a meaningful title.
- Click *Send pull request*.

5.2.7 7. Respond to feedback

The RubiX team may recommend adjustments to your code. Part of interacting with a healthy open-source community requires you to be open to learning new techniques and strategies; *don't get discouraged!* Remember: if the RubiX team suggest changes to your code, they care enough about your work that they want to include it, and hope that you can assist by implementing those revisions on your own.

5.2.8 8. Postscript

Once all the changes are approved, one contributor will push the change to the upstream code.

5.3 Coding conventions

- two spaces, no tabs
- no trailing whitespaces, blank lines should have no spaces
- Do not mix multiple fixes into a single commit.
- Add comments for your future selves and for your current/future peers
- Do not make whitespace changes as part of your regular/feature commits.
- If you feel whitespace issues need to be fixed, please push a separate commit for the same. It will be approved quickly without any discussion.

5.4 Testing

This section provides contribution guidelines specific to testing.

5.4.1 Robot Framework Integration Tests

For more detailed info regarding Robot Framework and its capabilities, read the Robot Framework user guide.

Test Suites

Each `.robot` file is a test suite. Related tests should be kept in the same test suite, to allow for reuse of variables and suite-level keywords.

Test suites contain the following sections:

*** Settings ***

This section is for specifying suite-level documentation, as well as keywords for suite-level setup & teardown.

This section also specifies other sources for keywords used by the test suite. These can be:

- Robot Framework built-in libraries, such as *Collections* and *OperatingSystem*
- Other `.robot` files containing helpful keywords, such as `setup.robot` for setting up & tearing down tests.
- A fully-qualified Java class from a custom library containing methods that can be run as keywords (more info *below*)

Example:

```
*** Settings ***
Resource  OperatingSystem
Resource  bookkeeper.robot
Resource  com.qubole.rubix.client.robotframework.BookKeeperClientRFLibrary
```

*** Variables ***

This section contains any variables common to the test cases in the suite.

Common variables needed across test suites include the following:

```
${WORKINGDIR}   ${TEMPDIR}${/}<test-suite-name>
${DATADIR}      ${WORKINGDIR}${/}data

${CACHE_DIR_PFX}    ${WORKINGDIR}${/}
${CACHE_DIR_SFX}    /fcache/
${CACHE_NUM_DISKS}  <number-of-cache-disks>
```

Note: `${TEMPDIR}` is supplied by Robot Framework at points to the operating system's temp directory, while `${/}` is the operating system's path separator.

*** Test Cases ***

This is where test cases are defined.

Test cases include a name on its own line, followed by the keywords to be executed for the test on indented lines following it.

For RubiX, we use test templates to run the test and verify that it passes with different modes of execution.

*** Keywords ***

This section contains any suite-level keywords, as well as keywords for running tests defined as templates.

Like test cases, keywords include their name on its own line, and the keywords to be run on indented lines after it.

Keywords should include a **[Documentation]** tag to provide details regarding the purpose and/or usage of the keyword.

Test Cases

In general, a test case will require the following components:

- Setup
- Body
  - Data generation
  - Execution
  - Verification
- Teardown

Setup

Start the test case with the **Cache test setup** keyword to start a BookKeeper server with the provided configuration options and create the directory used for storing generated data for the test.

The following example starts a server as a master, and configures the cache directory settings and its maximum size.

```
Cache test setup
...  ${DATADIR}
...  rubix.cluster.is-master=true
...  rubix.cache.dirprefix.list=${CACHE_DIR_PFX}
...  rubix.cache.dirsuffix=${CACHE_DIR_SFX}
...  rubix.cache.max.disks=${CACHE_NUM_DISKS}
...  rubix.cache.fullness.size=${CACHE_MAX_SIZE}
```

Test Body

Integration tests need to:

- generate any files needed for test execution
- execute whatever steps necessary to sufficiently test the desired scenario
- verify the state of the BookKeeper & cache using metrics and other helper keywords

Generation

You can generate data files individually:

```
${fileName} =  Generate single test file  ${filePath}  ${fileLength}
```

or as a batch of files with similar characteristics:

```
@{fileNames} =  Generate test files  ${filePathPrefix}  ${fileLength}  $
↪{numberOfFiles}
```

Execution

In order to execute calls using the BookKeeper server, you will need to make a request object.

Similar to generating test files, requests can be generated individually:

```
${request} =  Make read request
...  ${fileName}
...  ${startBlock}
...  ${endBlock}
...  ${fileLength}
...  ${lastModified}
...  ${clusterType}
```

or as a batch of requests with similar characteristics:

```
@{requests} =  Make similar read requests
...  ${fileNames}
...  ${startBlock}
...  ${endBlock}
...  ${fileLength}
...  ${lastModified}
...  ${clusterType}
```

For read calls, current execution modes include combinations of:

- Caching data by **directly calling the BookKeeper server** OR **using a client file system**
- Executing caching calls **sequentially** OR **concurrently**

The execution mode is determined by the keyword name passed into the test template. For example, the template below will first run the test with sequential calls to the BookKeeper server, and then with concurrent calls using the client file system.

```
Cache eviction
    [Template]  Test cache eviction
    Execute read requests using BookKeeper server call           runConcurrently=$
↪{false}
    Concurrently execute read requests using client file system  runConcurrently=$
↪{true}
```

The actual execution of the keyword is controlled by the following step, which will run the keyword concurrently on the specified number of threads if the flag is set to true, or sequentially otherwise.

```
RUN KEYWORD IF  ${runConcurrently}
...  Execute concurrent requests
...  ${executionKeyword}
...  ${numThreads}
...  ${requests}
...  ELSE
...  Execute sequential requests
...  ${executionKeyword}
...  ${requests}
```

Verification

Test execution can be verified by comparing metrics values to expected values.

```
Verify metric value  ${metricName}  ${expectedValue}
```

As well, you can verify that the size of the cache is the expected size.

```
Verify cache directory size
...  ${cacheDirPrefix}
...  ${cacheDirSuffix}
...  ${cacheDirNumDisks}
...  ${expectedCacheSize}
```

Teardown

Finish the test case with the **Cache test teardown** keyword as a **[Teardown]** step; this ensures the BookKeeper server used for this test is properly shut down and the environment is cleaned before execution of the next test.

```
[Teardown]  Cache test teardown  ${DATADIR}
```

Style Guide

Variables

Variables within a keyword are **camelCase**.

```
@{testFileNames} =  Generate test files  ${REMOTE_PATH}  ${FILE_LENGTH}  ${NUM_TEST_
↪FILES}
```

Variables in the "Variables" section of a test case are **ALL_CAPS_AND_UNDERSCORE_SEPARATED** (like Java constants).

```
${NUM_EXPECTED_EVICTIONS}  3
```

Keywords

Names

Built-in keywords are **ALL CAPS**.

```
CREATE DIRECTORY  ${directoryName}
```

Test keywords (defined in a test or resource `.robot` file) are **Sentence capitalized**.

```
Generate single test file  ${fileName}  ${fileLength}
```

Custom library keywords (eg. from BookKeeperRFClientLibrary) are **camel Case**.

```
&{metrics} =  get Cache Metrics
```

Arguments

Arguments have **2 spaces** between the keyword and each other.

```
Verify metric value  ${METRIC_EVICTION}  ${NUM_EXPECTED_EVICTIONS}
```

If the keyword needs more than 3 arguments, place the arguments on separate lines.

```
${request} =  Create test client read request
...  ${fileName}
...  ${startBlock}
...  ${endBlock}
...  ${fileLength}
...  ${lastModified}
...  ${clusterType}
```

Use named arguments for keywords where possible to enhance clarity.

```
Verify cache directory size
        .....
...  expectedCacheSize=${CACHE_MAX_SIZE}
```

For sets of keywords with similar arguments, alignment of arguments is preferred.

```
[Template]   Test cache eviction
Download requests               runConcurrently=${false}
Concurrently download requests  runConcurrently=${true}
Read requests                   runConcurrently=${false}
Concurrently read requests      runConcurrently=${true}
```

Custom Keywords

If a test requires more functionality than what Robot Framework can offer (such as when executing requests using the BookKeeper server), keywords can be created as functions in `BookKeeperClientRFLibrary`. All public methods in this class are exposed as keywords to be used by Robot Framework.

In the following example, `getCacheMetrics()` in `BookKeeperClientRFLibrary` is accessible for use by our custom Robot Framework keyword `Verify metric value`:

```
public Map<String, Double> getCacheMetrics() throws IOException, TException
{
  try (RetryingBookkeeperClient client = createBookKeeperClient()) {
    return client.getCacheMetrics();
  }
}
```

```
Verify metric value
    [Arguments]  ${metricName}  ${expectedValue}
    &{metrics} =  get Cache Metrics
    ...
```

5.5 Commit Message

Commits are used as a source of truth for various reports. A couple of examples are:

- Release Notes
- Issues resolved for QA to plan the QA cycle.

To be able to generate these reports, uniform commit messages are required. All your commits should follow the following convention:

For every commit please write a short (max 72 characters) summary in the first line followed with a blank line and then more detailed descriptions of the change.

Format of summary:

```
ACTION: AUDIENCE: COMMIT_MSG
```

Description:

```
ACTION is one of 'chg', 'fix', 'new'
Is WHAT the change is about.
'chg' is for refactor, small improvement, cosmetic changes...
'fix' is for bug fixes
'new' is for new features, big improvement

AUDIENCE is one of 'dev', 'usr', 'pkg', 'test', 'doc'
Is WHO is concerned by the change.
'dev' is for developers (API changes, refactors...)
'usr' is for final users
```

You will use your environment's default editor (EDITOR=vilemacs) to compose the commit message. Do NOT use the command line git commit -m "my mesg" as this only allows you to write a single line that most of the times turns out to be useless to others reading or reviewing your commit.

5.5.1 Example

```
new: dev: #124: report liveness metric for BookKeeper daemon (#139)

Add a liveness gauge that the daemon is up & alive. Right now, this
is a simple check that a thread (reporter to be added in a subsequent
commit) is alive. In the future, this simple framework will be used
to add more comprehensive health checks. Ref: #140
```

The above example shows the commit summary is:

- a single line composed of four columns
- column 1 tells us the nature of the change or ACTION: new
- a short one-line summary of WHAT the commit is doing

The description or the body of the commit message delves into more detail that is intended to serve as a history for developers on the team on how the code is evolving. There are more immediate uses of this description however. When you raise pull requests to make your contributions into the project, your commit descriptions serve as explanations of WHY you fixed an issue. HOW you fixed an issue is explained by code already. This is also the place where the peer-reviewers will begin understanding your code. An unclear commit message is the source of a lot of back and forth resulting in frustration between reviewers and committers.

Reference: http://chris.beams.io/posts/git-commit/

5.6 How to report issues

A bug report means something is broken, preventing normal/typical use of Rubix.

Make sure the bug isn't already resolved. Search for similar issues.

Make sure you have clear instructions to reproduce your problem.

If possible, submit a Pull Request with a failing test, or;if you'd rather take matters into your own hands, try fix the bug yourselfMake a report of everything you know about the bug so far by opening an issue about it.When the bug is fixed, you can usually expect to see an update posted on the reporting issue.

5.7 Documentation Style Guide

- Documentation uses Sphinx documentation generator.
- Documentation is hosted on ReadTheDocs
- File issues if you notice bugs in documentation or to request more information. Label issues with `doc`
- Contributions to documentation is accepted as a Pull Request.
- Choose Markdown if you will add new pages
- Choose Rich Structured Text (rst) for indexes or if the documentation needs tables.
- To locally test docs changes run `python -msphinx . _build` inside `docs` directory

CHAPTER 6

Release Notes

6.1 Release 0.3.10

- Bypass getFileInfo network call when staleness check is enabled
- Remove runtime dependencies to fix ClassNotFound errors in Embedded mode
- Ensure InputStream for DirectReadRequestChain is always closed
- Make CachingFileSystem extend FilterFileSystem
- Fix connection leak in RetryingPooledThriftClient that happens if a connection terminates with an exception

6.2 Release 0.3.11

- Fix a regression from 0.3.10 that caused wrong BlockLocations to be returned in CachingFileSystem#listLocatedStatus
- Presto's native NodeManager can now be used as cluster manager in embedded mode

6.3 Release 0.3.12

6.3.1 Fixes and Features

- Prevent RemoteFetchProcessor from stopping on exception
- Fail fast in BookKeeper startup if no disks are available for caching
- Fix over-estimation of disk usage by cache
- Enable FileSystem object cache in Rubix servers

- Allow configuring Rubix via a separate xml file. `rubix.site.location` can be used to provide location of Rubix configuration file
- Removed shading of GCS connector to fix caching over GoogleHadoopFileSystem

6.3.2 New Extensions

- CachingPrestoAliyunOSSFileSystem: Caching over AliyunOSSFileSystem
- CachingPrestoAdlFileSystem: Caching over AdlFileSystem

6.4 Release 0.3.13

6.4.1 Fixes and Features

- Generation numbers are added for files cached on disk to avoid several race conditions with invalidations
- Scavenger service has been added to reap the idle connections
- Local Data Server connections are now pooled
- Fail fast when BookKeeper or Local Data Server sockets cannot be created
- Use bounded thread pools BookKeeper and Local Data Server
- Parallel warmup is now enabled by default

6.5 Release 0.3.14

6.5.1 Fixes and Features

- Fixed a regression from 0.3.11 which slows down split generation.
- Jmx stats refactoring to for better accounting of stats.
- Added support to plug in custom reporter for metrics that can send metrics to custom sinks. It can be set used by setting *rubix.metrics.reporters=CUSTOM* and providing implementation class using *rubix.metric-collector.impl*.

6.6 Release 0.3.15

6.6.1 Fixes and Features

- Run scavenger thread in daemon mode to allow jvm to exit. This prevents Spark apps from getting stuck during exit.

6.7 Release 0.3.16

6.7.1 Fixes and Features

- Allow ClusterManager implementations to provide hostname and host-address instead of always using fixed ones
- Initialize caching stats in embedded-mode initialisation instead of CachingFileSystem initialisation

6.8 Release 0.3.17

Note: This release removes shading of thrift jars from rubix-spi. If you are using rubix-spi as a dependency, you will not find the thrift classes from this jar and you will need to use rubix-build jars instead.

6.8.1 Fixes and Features

- Improvements in consistent hashing logic to minimize redistributions during change in the membership of the cluster
- Moved shading of Thrift classes into a common sub-module rubix-build. Project is now traversable in IDE and mvn test works at the root. Clients should now include rubix-build artifact instead of including sub-modules independently
- Consider requests served from another node's cache under cache hit
- Added *total_system_source_mb_read* stat in detailed metrics to show total data read from source: during read + warmups

6.9 Release 0.3.18

6.9.1 Fixes and Features

- Cleanup cache directories before initializing the cache to correctly measure available disk space
- Update cache status with each successful readRequest in case of parallel warmup to minimize the errors in accounting disk space. Maximum read-request length is also limited to 100MB by default to minimize accounting errors.

6.10 Release 0.3.19

6.10.1 Fixes and Features

- Add an implementation of PrestoClusterManager that does not cache the list of worker nodes. Set *rubix.cluster.manager.presto.class* as *com.qubole.rubix.prestosql.SyncPrestoClusterManager* to use the new implementation.

6.11 Release 0.3.20

6.11.1 Fixes and Features

- Fix for Presto cluster managers to return current NodeName from Presto's NodeManager when it is available

6.12 Release 0.3.21 (next release)

6.12.1 Fixes and Features