General overview

Consisting of one large deployable and two distinct types of client interface, the Cache engine comprises of ActiveMQ, HSQLDB, SpringBoot, Tomcat, Spring, EhCache and Hibernate components. All components are embedded. The Cache accepts price input from messages submitted to a JMS queue. The message sent must conform to a PriceNotification json format, an example of which is:

"{\"vendorName\":\"NickPricer\",\"instrumentIsin\":\"XYZ876\",\"instrumentName\":\"The XYZ876 instrument\",\"bidPrice\":96.0,\"offerPrice\":98.0}"

When such a price notification is received, the CacheEngine stores it, along with creating a new Instrument record and a new Vendor record (if they do not already exist.) The cache then publishes the price onto a topic, available for interested downstream clients to consume. Cached price data is also available on request via several Restful web services. This offers clients the facility of obtaining all prices for a specified instrument, all prices from a specified vendor or a specific instrument’s price from a specific vendor.

The Cache does not create a permanent record of price data, maintaining the prices it receives in memory. A number of options exist to offer price record persistence, ranging from a permanent database store to storing each JMS incoming message, for subsequent replaying into the cache.

Prices older than thirty days are removed by a scheduled task. An external properties file sets the thirty-day threshold. The properties file additionally contains details of the host, port and names for the inbound prices queue and outbound topic. The external properties file is also available for two test clients, also included in this submission. These clients post messages to the inbound prices queue and consume messages from the outbound publisher topic and can be used as a basis for vendor API/ client API implementations to interact with the Cache.

The Cache can pre-populate its embedded HSQLDB using an external sql script, an example of which is in this document’s appendix.

Design Considerations

Given price data volume can be volatile, an architecture that separates the cache engine’s processing from the means to capture the incoming prices allows the cache engine more time to process price records. A JMS queue provides that entry point, absorbing high volumes while the cache engine continually processes each queued message. In the event that inbound data volumes exceed that which is manageable by an embedded JMS solution, the JMS implementation could be migrated to an external broker, requiring a small amount of application refactoring.

The data store is an embedded HSQLDB. Chosen due to its simplicity, this is fronted by Hibernate, backed by EhCache. To revise HSQLDB out in favour of an external database would require changing the hibernate config (one file) which would include changing to a new data source and possibly SQL dialect.

SpringBoot and Spring affords the application separation of concerns, assisting with quick and maintainable development along with unit testable components.

Restful web service endpoints are exposed to provide on-demand price data. This allows the cache to continue processing and publishing incoming price data while servicing client requests for specific price data on a request/response basis. The controllers for these services are simply reusing the services available within the cache.