Aerospike REST Client package

This package contains the following files

- `swagger.json` The swagger specification for the REST API.
- `api-doc.html` Generated HTML documentation for the API.
- `as-rest-client##<VERSION>.war` A `.war` file to be deployed in Tomcat, or another server accepting `.war` files. Directions for installation are provided further down in this document.
- `stocks/` A directory containing our demo application. Information on Demo Application usage is provided at the end of this documentation.

Installation and Configuration

Requirements

- The REST Client requires Java 8.
- The REST Client requires an Aerospike Server to be installed and reachable. See Configuration for details on specifying the location of this server.

Running on Tomcat

- If not already installed, download and install Tomcat . We recommend the Core distribution of Tomcat 9, found under the Binary Distributions section.

This will create a root installation folder which looks something like

```
./bin/
./conf/
./logs/
./webapps/
```

- Place the REST Client `.war` file in your tomcat installation's `webapps` folder.
- For more detailed server configurations, refer to the Documentation for the version of Tomcat which you are using. For Tomcat 9 these are located at: https://tomcat.apache.org/tomcat-9.0-doc/introduction.html
- Start tomcat. One way to do this is by running `bin/catalina.sh run` or `bin/catalina.sh start` from the root folder of your Tomcat installation.

Verifying installation

Note: The following steps assume Tomcat's root is at `http://localhost:8080` and the REST Client's base path is `http://localhost:8080/as-rest-client`

if this is not the case, the provided URLs will need to be modified accordingly.

To test that the rest client is up and running, and connected to the Aerospike database you can run:

```
curl http://localhost:8080/as-rest-client/v1/cluster
```

This will return basic information about the cluster.

Interactive API documentation may be found at http://localhost:8080/as-rest-client/swagger-ui.html . This will allow you to test out various commands in your browser.

The Swagger specification, in `JSON` format, can be found at http://localhost:8080/as-rest-client/v2/api-docs .

Configuration

By default the REST Client looks for an Aerospike Server available at `localhost:3000` . The following environment variables allow specification of a different host/port.

- `aerospike_restclient_hostname` This is the IP address or Hostname of a single node in the cluster. It defaults to `localhost`
- `aerospike_restclient_port` The port to communicate with the Aerospike cluster over. Defaults to `3000`
- `aerospike_restclient_hostlist` A comma separated list of cluster hostnames and ports. If this is specified, it overrides the previous two environment variables. The format is described below:

```
The string format is : hostname1[:tlsname1][:port1],...
* Hostname may also be an IP address in the following formats.
*
* IPv4: xxx.xxx.xxx.xxx
* IPv6: [xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx]
* IPv6: [xxxx::xxxx]
*
* IPv6 addresses must be enclosed by brackets.
* tlsname and port are optional.
*/
```

The REST Client also allows authentication to an Aerospike Enterprise edition server with security enabled. The following environment variables are used to find authentication information. **The Aerospike REST Client supports a single user only. If more than user is necessary, contact support@aerospike.com**

- `aerospike_restclient_clientpolicy_user` This is the name of a user registered with the Aerospike database. This variable is only needed when

the Aerospike cluster is running with security enabled.
- `aerospike_restclient_clientpolicy_password` This is the password for the previously specified user. This variable is only needed when the Aerospike cluster is running with security enabled.

REST Client Data Formats

API Requests which involve sending data can use the `JSON`, or `MessagePack` formats. By default JSON will be assumed. To use `MessagePack`, set the `Content-Type` header to `"application/msgpack"`. Similarly Responses may be sent in `JSON` or `MessagePack`, `JSON` is the default. To receive `MessagePack` formatted data set the `Accept` header to `"application/msgpack"`.

JSON Use Cases

For many uses `JSON` is a simpler and completely valid option. It provides simplicity of use, and familiarity. If basic Key Value operations are being used, and neither Maps with non string keys, Bytes nor GeoJSON are required, then `JSON` will work completely with the Aerospike data model.

Message Pack Use Cases

Message pack is provided as an option because JSON cannot fully represent certain Aerospike data types. Specifically:

- Aerospike can store arrays of bytes.
- Aerospike maps may have keys which are not strings. e.g `{1:2, 3.14: 159}`.
- Aerospike stores a GeoJSON type. Which is returned as a `MessagePack` extension type.

If you are not handling Maps with non string keys, and not using bytes nor GeoJSON, then JSON as an interchange format will work for the Rest Client.

MessagePack Format

The `MessagePack` sent and received by the REST client is almost completely standard. The one specific detail is that we represent a `GeoJSON` object using the MessagePack Extension format type. The extension type value is `23` and the payload is the string representation of the `GeoJSON`. This is done to differentiate a normal string from GeoJSON. For example to write a bin map usable by the API with a GeoJSON value utilizing Python.

```
# Python 2.7
import msgpack
packed_geojson = msgpack.ExtType(23, "{\"coordinates\": [-122.0, 37.5], \"type\": \"Point\"}
packed_bins = {u'geo_bin': packed_geojson}
mp_bins = msgpack.packb(packed_bins)
```

Or with Java

```
MessageBufferPacker packer = new MessagePack.PackerConfig().newBufferPacker();
String geoString = "{\"coordinates\": [-122.0, 37.5], \"type\": \"Point\"}";
packer.packMapHeader(1);
packer.packString("geo_bin");
packer.packExtensionTypeHeader((byte) 23, geoString.length());
packer.addPayload(geoString.getBytes("UTF-8"));
```

Bytes are a standard Message Pack type. Here is an example of creating a Bin Map to be used with the API

```
# Python 2.7
test_bytes = bytearray([1,2,3])
mp_bytes_bins = msgpack.packb({u'my_bytes': test_bytes}, use_bin_type=True)
```

```
byte[] testBytes = {1, 2, 3};
MessageBufferPacker packer = new MessagePack.PackerConfig().newBufferPacker();

packer.packMapHeader(1);

packer.packString("my_bytes");

packer.packBinaryHeader(3);
packer.writePayload(testBytes);

byte[] payload = packer.toByteArray();
```

Demo Application

This demo application is a React app using react-redux components generated from the rest client's swagger specification.

The application models an example site which allows users to create personalized portfolios, and monitor the portfolios' performance. The application also displays information about the API requests to the rest client utilized in the application.

Note The provided API call details only include information about request and response data; to see more information about headers and timing we suggest utilizing your browser's developer tools.

Usage

Note These instructions assume that the demo is running on a Tomcat server with a base url of `http://localhost:8080`. If this is inaccurate, the provided URLs will need to be updated.

Prerequisites

- Ensure that your Aerospike database has a namespace named test
- Follow the steps for installaing and starting the REST Client
- Make sure that the rest client is available at `http://localhost:8080/as-rest-client`.

Installation

- Place the `stocks` folder which came with this package into your Tomcat Installation's `webapps` folder.
- Restart Tomcat

Using the demo

To try out the demo application go to http://localhost:8080/stocks

You should be prompted to upload some stock data. Follow the directions on the uppload page to acquire this data file. The application will transform this data into a series of records which will be stored into Aerospike. See Data Model section for information on the actual structuring of this data.

In the demo application, you may always return to the homepage by clicking on the trending upward icon in the top left corner of the page.

Model

The input data for this application is a series of daily entries about stock symbols. Those entries include the date, stock symbol, opening, and closing price.

For each daily entry we create a record structured as follows:

```
symbol: string # The stock symbol
date: string # The date of the entry
open: float # The opening price for the day
close: float # The closing price for the day
```

The key of this record is formed by concatenating the stock symbol with the string representation of the date. For example for the `GOOGL` entry for `2013-02-08` the resulting key is `GOOGL-2013-02-08`.

In addition to these daily entries there is a record for each individual stock. This record is structured as followed

```
name: string # The symbol for this stock
dates: [string, ...] # A list of string representations of the daily entries existing for th
```

There is also a record including a list of stock symbols:

```
symbols: [string, ...] # A list of all stock symbols.
```

The Application also includes the concept of a portfolio, which is a collection of stocks. Each portfolio has a record structured as:

```
id: string # Unique identifier for the portfolio
name: string # Name of the portfolio, in this app it is always the same as id
stocks: list[string, ...] # A list of stock symbols contained in this portfolio
```

There is another record containing the most recently created portfolios:

```
portfolios: [string, ...] A list of recently created portfolios
```