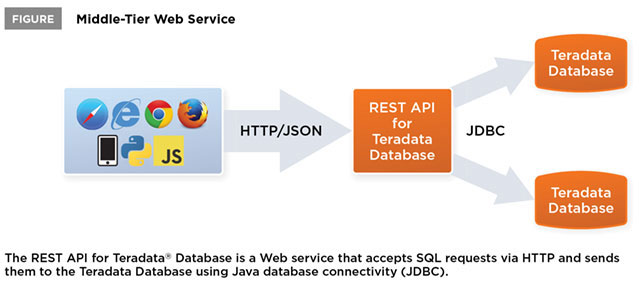
Teradata Hackathon Primer

Teradata REST

Prerequisites: A fundamental, basic familiarization of database systems (especially relational databases) and their structure (databases, tables, rows, columns, etc) and basic understanding of SQL will be necessary in order to understand and user Teradata REST services. If you are not at least somewhat familiar with these concepts, it is suggested that you briefly look at one of the many SQL/Database tutorials available online.

Teradata REST Services will be the interface for retrieving data from the Teradata repository that will be housing all the data for this event. TD REST is a middle-tier web service interface that allows you to retrieve system-specific information, configure monitored Teradata systems, run queries against these systems, and access other import metadata associated with the system, its sessions, or its queries. For the purposes of the hackathon, we will be using TD REST to execute queries against the Teradata system via an HTTP request.

The HTTP request is processed by the REST web service, which connects to the database via a Java JDBC driver, executes your query, and then returns the results of your queries in the HTTP response as a JSON object. The cool thing about Teradata REST is that all of this interaction can be done without the use of language-specific drivers. If you can execute an HTTP request (which can pretty much be done in any language), then you are good to go! The diagram below outlines how this works.

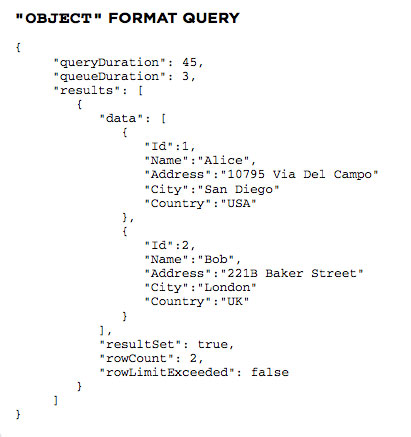


If you are unfamiliar with JSON structure or HTTP, I highly encourage you and your team to briefly read up on the fundamentals of both these concepts. They are fairly straightforward, but it is a good idea to get up to speed with these concepts before the hackathon event. It is also a good idea to look into what type of plugin or technology (specific to your programming language of choice) you will be using to execute HTTP requests against the system. Below is a quick example of how a REST API would be executed via jQuery, a very popular Javascript library.

$.ajax({  
 type: "POST",  
 url: "https://myhost:1080/tdrest/systems/mysystem/queries",  
 contentType: "application/json",  
 headers: {  
 'Accept': "application/vnd.com.teradata.rest-v1.0+json",  
 'Authorization': 'Basic ' + btoa(username + ':' + password)  
 },  
 data: JSON.stringify({  
 query: "SELECT \* FROM Customers",  
 format: 'object'  
 })  
});

In this case, we are declaring the HTTP **type** to be a POST request type, with the **url** above… more on this momentarily. The **contentType** declares the type of response we expect to be returned from the server (the two options for Teradata REST are JSON or CSV… I suggest you stick with JSON). The **headers** declare the Accept HTTP header type. This will be the same for all requests, so do not worry too much about this. Each team will be provided with proper username and password credentials for accessing the database on the day of the event. The **data** field is what will be included in the request body. In this case, we’re expecting to run the query “SELECT \* FROM Customers”, with an ‘object’ return type format. This return type is optional (and defaults to ‘object’), but other types are outlined below, if you so choose to use them:

* "object" is the default format and causes results to be returned as a JSON object per row in which the column name is the field name and the column value is the field value.
* "array" has results returned as a JSON array per row where each column value is an element in the array.
* "CSV" is for a comma-separated values option in which results are returned as a separate line for each row with commas between the column values.



As a side note, typically in a REST interface, if we are simply retrieving data without needing to provide anything in the request body, we would use a GET request as opposed to a POST or PUT. However, GET requests (and DELETEs) do not have a request body. In this situation, we need to provide the query data in the request body, so we use a POST to allow us to provide that information to the REST service. A basic tutorial on HTTP can be found here:

<http://www.tutorialspoint.com/http/>

I recommend that you focus primarily on the Request/Response aspects, and possibly the “Header” section, to get a basic understanding of how these HTTP headers are used.

Let us dissect the “URL” of the request a bit further.

"<https://myhost:1080/tdrest/systems/mysystem/queries>"

In the case above, “myhost” refers to the system running the REST webservice application. This will be provided to you at the beginning of the hackathon. The REST container listens for requests on port 1080. /tdrest/ refers to the rest inferface base API URI - this will be pretty much prefix all your requests. /systems/mysystem/ means that we want to target the “mysystem” system for our request.

Note: if we fire a GET request against /tdrest/systems/ without a specifying a specific system, we retrieve a JSON object containing a list of all configured systems. Similarly, a GET against /tdrest/systems/mysystem/ would return the following:

{  
 "name": "mysystem",  
 "host": "sdlc666.labs.teradata.com",  
 "port": 1025,  
 "logMech": "DEFAULT",  
 "encryptData": true,  
 "defaultCharSet": null,  
 "useXViews": false,  
 "maxIdleSeconds": 600,  
 "defaultTransactionMode": "DEFAULT",  
 "defaultDatabase": null,  
 "maxImplicitSessionsPerUser": 5,  
 "maxExplicitSessionsPerUser": 5,  
 "maxQueuedRequestsPerUser": 100,  
 "includeOrExcludeUserList": null,  
 "includeUsersInList": false,  
 "queryBands": {},  
 "links": [  
 {  
 "rel": "self",  
 "href": "http://sdlc6372.labs.teradata.com:1080/tdrest/systems/mysystem"  
 },  
 {  
 "rel": "databases",  
 "href": "http://sdlc6372.labs.teradata.com:1080/tdrest/systems/mysystem/databases"  
 }  
 ]  
}

Notice the “links” at the bottom of the response? You can continue to go further through the API tree to get even more information. Similarly, a GET to /tdrest/systems/mysystem/databases would retrieve the list of databases for the “mysystem” system. This is important for our purposes, because each dataset will be housed in its individual database on the system. Going one step further, if we had a database on this system called “crime\_data”, and we fired a GET to /tdrest/systems/mysystem/databases/crime\_data, we would get a listing of every table in the crime\_data database. These tables would be what we would eventually want to target for our data retrieving queries.

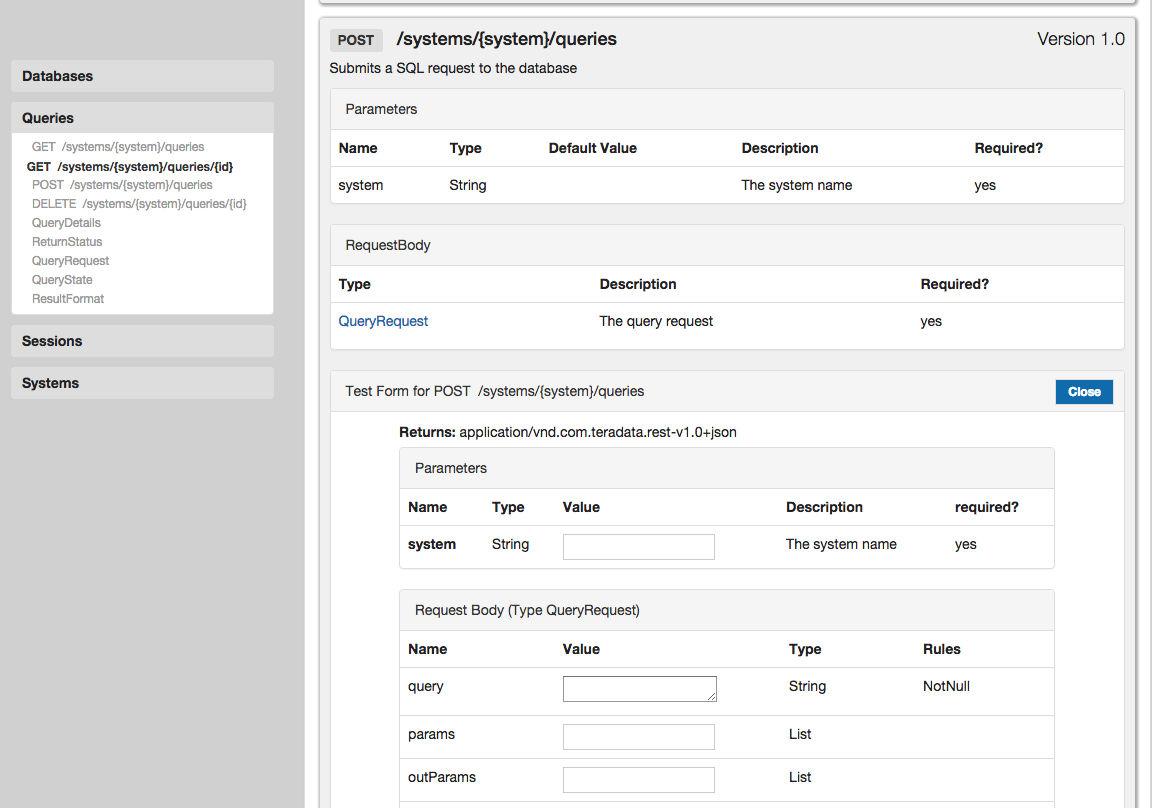
Our example above demonstrates how to fire a request programmatically using javascript. However, there are other manners of testing these APIs without having to code anything. The easiest way is to get a browser plugin that allows you to fire HTTP requests while providing HTTP headers and request bodies. Below are some links for browser plugins that can be used to test these APIs easily. Considering the limited length of the event, with a multi-person team, I would suggest that one person spends the beginning of the hackathon perhaps analyzing the available datasets and data using these REST clients as other members get to design and development.

* Firefox: <https://addons.mozilla.org/en-us/firefox/addon/restclient/>
* Chrome: <https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo?hl=en-US>
* Mac: <http://mmattozzi.github.io/cocoa-rest-client/>
* IE/Safari: Change your browser of choice

Additionally, depending on your language of choice for your project, there are different manners of executing AJAX or HTTP calls. Here are some possible solutions, although there are numerous other options:

* Javascript: jQuery AJAX (<http://api.jquery.com/category/ajax/>)
* Python: Python-Requests (<http://www.python-requests.org/en/latest/>)
* Java: Native URLConnection class (<http://stackoverflow.com/questions/2793150/using-java-net-urlconnection-to-fire-and-handle-http-requests>) - You can also try other, simplified java libraries such as <https://github.com/kevinsawicki/http-request>
* C#: Native WebRequest class (<https://msdn.microsoft.com/en-us/library/456dfw4f(v=vs.110).aspx>)

In addition to the above options, the TD REST Services provides a light-weight, GUI web application that allows you to visually identify some of the possible REST APIs that can be used, along with possible fields associated with each API. Here’s the GUI screengrab for the API that you will likely be using the most. In fact, the TD REST GUI can be used for almost every request that you want to test, as an alternative to using a browser-based or standalone REST client.



I have attempted to focus on the more important or relevant aspects of HTTP and RESTful services, but of course, both subjects are broad and certainly worth a deeper understanding. I **highly** suggest familiarizing yourself with these concepts so that your time during the hackathon can be focus on design and implementation, as opposed to having to learn an entirely new concept. Below are some useful links related to both Teradata REST and to REST services as a whole.

Teradata REST: <http://www.teradatamagazine.com/v15n01/Tech2Tech/Driverless-Connectivity/>

REST tutorial (pages 1 and 2 should be the focus)

<http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069?pgno=1>

<http://www.restapitutorial.com/index.html>

Github

GitHub will be used for source control and teams will be requested to commit their code on a regular basis during the competition. Each team will need to use one of their own personal accounts or create one for this purpose. Each team will also be required to share their repository with Teradata to allow for judging of the source files.

If you are not already familiar with github or version control, I strongly encourage you and your team to take a look at the links below. No need for an in-depth understanding of version control or git in general (although it certainly would be a great idea), but understanding how to use github on a basic level will be required for code/project submission.

<http://readwrite.com/2013/09/30/understanding-github-a-journey-for-beginners-part-1>

Cheers, and happy coding!