Arista – Benchmark Dashboard

Design Document

Author: Bijan Mishra (Happiest Minds Technologies)

Bijan.mishra@happiestminds.com

Contents

[Change Log 3](#_Toc454403284)

[Current System: 4](#_Toc454403285)

[High level Flow Architect: 5](#_Toc454403286)

[Description of flow: 5](#_Toc454403287)

[Database Schema: 7](#_Toc454403288)

[Query Log: 9](#_Toc454403289)

# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SR | Description | Date | Change by | Remarks |
| 1 | Draft | 20-06-2016 | Bijan Mishra | Creation of initial draft |
|  |  |  |  |  |

# Current System:

URL: <http://benchmark>

Script: /src/Artest/www/ArtestCgi.py

Description: Bench marking of the parameters how they behaving.

Technical Components:

* Front End:
  + HTML5,CSS 3
  + JQuery
  + D3.js
  + Java script
* Back End :
  + Node.js
  + Java script
  + SQLite Database
* Client Serve :
  + MySQL Database
* Unit test case :
  + Jasmine

# High level Flow Architect:

MySQL

Node js

Client

Node Scripts

SQLite

# Description of flow:

**Step 1: Node js to MySQL Db**

* Once The code is deployed on the server, One node js script will query to MySQL DB .
* MySQL DB Details :
  + - Host : benchmark.aristanetworks.com
    - Database : benchmark, run
    - User : arastra
    - Port : 3306
* Table Details:
  + - Table name: benchmark
      * Column names: id (varchar), description (text)
    - Table name: run
      * Column names: benchmark (varchar), result (double), dut (varchar), project (varchar), release (varchar), client (varchar), changeNum (integer), testTime (timestamp)
* Firstly it will query for “Run” table to get the benchmark names which are active for last one year starting today.
* Once it got the successful response from the DB,Script will create a SQLite DB instance in the local server and will create a “benchmark\_local” and “run\_local” table in the SQLite DB.
* All the response for first query will be pushed to “benchmark\_local” table.
* Secondly, it will again query for “run” table. It will fetch the last/latest 50 entries of those benchmarks, which have uploaded any data in the past year.
* Once the fetch is successful, it pushes the records to “run\_local” table.

**Step 2 and 3: update “run\_local” table in a interval of 10 minutes**

* Once we got all the required MySQL db values into the local SQLite, Another script will query for the new records that are added in the MySQL db.
* The script will run as a cron job at a time of interval of 10 minutes.
* After the successful fetch of new records from MySQL Db, the data will be pushed/replaced into the “run\_local” table in local SQLite.

**Step 4 and 5: UI on load**

* Once the UI is loaded, an Ajax call will be fired to the node js server to get the the json data for the drop down values.
* When node js gets this request, it will query and fetch all the data from “benchmark\_local” table in local SQLite db.
* Once the data fetch is done, it will give the response to the UI.
* When UI will get the response, it will manipulate the response data for display and populating drop downs.

**Step 6 and 7: Drop down value selection**

* Once Id of the benchmark form the drop down values are selected, another Ajax call will be fired to node js server.
* Only the id of the benchmark will be passed to the Ajax call.
* Once the node js get the request, it will get the id of the benchmark from the request.
* Using that id, one query will be fired to get all the relevant data for that benchmark id from the “run\_local” table in the local SQLite db.
* Once the data fetched successfully, it will be converted in to json and response will be given back to the UI.
* Once the UI get the json response, it will be passed to a JavaScript function to plot the graphs for the drop down selection.

**Step 8: Selection in the Filter**

* By default, user will see data for latest 50 entries. User can select other than 50 entries in the data value filter [20, 50,100, 200…], if he/she wants to display less or more data of the selected id in the graph.

**Step 9: UI to nodeJs server**

* When user selects to get data for more than 50 entries then , An Ajax call with the selected id of the benchmark and selected number, will be fired to node js server.

**Step 10: nodeJs to MYSQL DB**

* The node js fires a query to the MYSQL database to fetch those number of records selected in the filter, for particular benchmark id from Run table.

**Step 11: nodejs to UI**

* Once the data fetched successfully, it will be converted to json and response will be given back to the UI.
* Once the UI gets the json response, it will be passed to a JavaScript function to plot the graph for the drop down selection for those many number of records selected in the filter.

**Step 12: Refresh button**

* When the refresh button is clicked on the UI, both the Ajax call will be fired to get the data for the drop down and the data for the selected id.
* Once the UI get the response, it will update the dropdown values as well as the chart values. But the selection won’t change**.**

# Database Schema:

* MySQL Database schema (master tables from which data is to be fetched):

1. Benchmark table :

|  |  |
| --- | --- |
| Column Name | Type |
| id | Varchar (255) NOT NULL DEFAULT PRIMARY KEY |
| description | text |

1. Run table :

|  |  |
| --- | --- |
| Column Name | Type |
| benchmark | varchar(255) DEFAULT NULL |
| result | double DEFAULT NULL |
| Dut | varchar(255) DEFAULT NULL, |
| project | varchar(255) DEFAULT NULL |
| Release | varchar(255) DEFAULT NULL |
| client | varchar(255) DEFAULT NULL |
| changeNum | int(11) DEFAULT NULL, |
| testTime | timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP |

* Local SQLite Database schema :

1. Local\_benchmark table :

|  |  |
| --- | --- |
| Column Name | Type |
| id | Varchar (255) NOT NULL DEFAULT PRIMARY KEY |
| description | text |
| firstDropDownValue | text |
| dropDownFlag | text |

1. Local\_run table :

|  |  |
| --- | --- |
| Column Name | Type |
| benchmark | varchar(255) DEFAULT NULL |
| result | double DEFAULT NULL |
| Dut | varchar(255) DEFAULT NULL, |
| project | varchar(255) DEFAULT NULL |
| testTime | timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP |

Unit test case design:

* Unit test cases will be written using Jasmin frame work.
* Test cases will be divided into 3 categories :
  + Unit test case for Front end
  + Unit test case for back end
  + Unit test case for end to end
* Unit test case for front end will include the below :
  + Function functionality
  + Dom functionality
  + Data validation
  + Ajax validation
* Unit test case for back end will include the below :
  + Function functionality
  + Response and request validation
  + Data validation and correctness
* Unit test case for end to end will include the below :
  + Response and request validation
  + Time taken at different break points
  + Data correctness
  + Flow correctness

# Query Log:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SR | Query | Raised By &  Date Raised | Response | Remarks |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |