Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

Visualization of Multi-Service System Network with D3.Js & Kdb+/q using Websocket

Ali Asgher Kapadiya

Received: 9 December 2017 Accepted: 4 January 2018 Published: 15 January 2018

Abstract

Visualization of complex web of services running in a multi-service system using D3.js as frontend, KDB+/q as backend and WebSocket JSON for communication.

Index terms— multi-service systems, visualization, Kdb+, q, D3.js, websocket, JSON, SVG.

1 Introduction

ata visualization is everywhere. In the past decade, the JavaScript world has evolved multiple folds. A lot of JavaScript frameworks provide visualization analytics that brings the information to life. It has moved from the world of bar charts and scatter plots to a lot of other interactive charts which are now used heavily like Gauge charts (dial charts) in risk systems, Spider charts (radar charts) to compare the multivariate data, etc.

One important aspect these JavaScript frameworks provides is the visualization of the relationship between the data. For example, during the investigation of Panama Papers, the ICIJ network used Linkurious.js **??**1]. This JavaScript library provided a simple and powerful way to visualize the graph database generated by Neo4j from more than 11.5 million documents, representing around 2.6 terabytes of data, enabling the investigators to uncover the potential stories in a short time frame.

There are multiple JavaScript frameworks available for data visualization D3.js, Linkurious.js, Processing.js, Raphael.js to name a few. The comparison of these visualization frameworks is beyond the scope of this paper.

2 a) Motivation

The idea of this paper is to visualize the complex network of the multi-service system (having 100s of services serving different functionalities) using the WebSocket and JSON for communication. This visualization can help the developers/maintenance team to understand the complexity of the system and analyze the flow of information between the services.

In this paper, we are going to visualize the complex web of Kdb+/q services in a typical financial institution using the D3.js as a visualization tool. However, any system publishing the events to a centralized service supporting the WebSocket and JSON can easily be integrated with the D3.js implementation we are going to discuss in this paper.

3 b) D3.js

D3.js (D3 or Data-Driven Documents) created by Mike Bostock **??**2], is one of the JavaScript frameworks which uses the browser capabilities for producing interactive visualizations. It works perfectly with the DOM (Document Object Model) and is widely used on websites across the world for displaying the data in interactive graphical components.

In general, there are two different ways to create and visualize the graphical components:

? SVG -Remembers the objects in a DOM and enables the event handlers to be associated with objects.

? Canvas -Renders the objects to a picture, but highly performant while handling a large number of objects.

4 c) Kdb+

Kdb+ is column -oriented, in-memory database known for its high-performance in the financial world. It was created by Arthur Whitney and is now one of the highly suitable databases for real-time time-series data analytics

and is capable of handling millions of records effectively. The 32-bit version is free to download and use from https://kx.com/.

Kdb+ is used for tick data and analytics platform in investments banks, hedge funds, etc. for various real-time analytics, P&L, risk assessment, best execution and regulatory purpose. It is slowly making its way to other fields like life-sciences, gaming, space, etc **??**3]. A typical Kdb+ platform has evolved to hundreds of q services serving a different functionality like data capture, in-memory database, historical database, realtime calculation, user query gateways, etc.

5 II.

6 Process Visualizer

There are two main components we are going to discuss in this paper. We will be using the term Process and Service interchangeably in this paper.

7 a) Monitor process

It is a centralized service containing the information of all the processes and connections among those processes. In this paper, it has been implemented in Kdb+ however it can be implemented in any other programming language and for any system having the complex web of connections.

8 Data Structure

Following are the data structures we are going to use in this paper.

Node: A node is a point in the network that holds some properties. In our example, we are going to represent each Kdb+ service as a node in the network with two attributes 'prc' (process name) and 'grp' (process group).

Link: An edge or a link represents a connection between two nodes. In our case, a link is representing a q/Kdb+ process connection to another q/Kdb+ process.

9 IV.

10 Monitor Process

A Kdb/q process that acts as centralized service and has information of all the q processes running in the system and the connection between these q processes or to the users or the third-party apps. The appendix contains the full code for the monitor process. There are three main functionalities of a q monitor service.

? Monitoring -Listening all process/connection events ? Subscriptions Maintaining the list of subscribers ? Notification -Notifying the subscriber about any addition/deletion of process/connections a) Monitoring functionality Following are the functions through which the Monitor service cab be notified about any process/connection addition/drop.

11 ? Add Prcocess

Any service when it comes up needs to notify the monitor process about its name and meta using this function.

12 ? Add Connection

When a process is requested for a connection from some other process, user or third-party app, the process can inform the monitor service using the add Connection method.

13 ? Drop Process

This function can either be called from a process which is gracefully shutting down or can be called by monitor process connection handler after detecting the disconnection.

14 ? Drop Connection b) Subscription Functionality

The subscription functionality maintains a list of subscribers who are interested in process/connection events. The browser while loading the page will connect to Monitor service for any event updates.

15 c) Notification functionality

The monitor process will publish the following event to all the subscribers in case of any events along with the process/connection details. The web component needs to implement/handle these events and render the changes accordingly on the screen. Visualization of Multi-Service System Network with D3.Js & KDB+/q using Websocket -This function will be called when a q process drops a connection to some other process either gracefully or detecting a disconnection by connection close handler.? add_prc_event ? add_conn_event ? remove_prc_event ? remove_conn_event V.

16 Web Component

? WebSocket connection to Monitor service and subscribing itself for any event updates ? Call-back functions to listen and process the process/connection updates. ? Rendering the nodes (processes) and links (connections) on D3 SVG.

17 a) Connection to Monitor service

The following JavaScript code is connecting to the Monitor service using the browser WebSocket functionality and implementing the standard call-back methods of WebSocket.

18 b) Monitor Event call-back methods implementation

Implementation of the monitor process callbacks methods and then delegating the calls for displaying the process/connection updates using d3.js.

19 c) Initializing the D3 SVG object d) Refreshing the nodes and links

The 'refresh' function contains the code to refresh the SVG after each node/link addition/removal update.

20 VI.

21 D3.js in Action

The appendix below contains the full code for creating and displaying the process network. a) Steps to Run 1) Download 32bit version of KDB+/q from https://kx.com/download/ 2) Extract the zip file to 'C:\'; it will produce a folder 'C:\q **??** [4]. 3) Save the 'monitor.q', 'Visualizer.html' and 'simulator.q' files to the 'C:\q\w32' directory. 4) Open a command prompt, go to the 'C:\q\w32' directory and run the 'monitor.q' file C:\q\w32>q monitor.q -p 5555 .nodes(nNodes).links(nLinks).linkDistance(50).charge **??**-200) .on("tick", tick); var svg = d3.select("body").append("svg").attr("width", sWidth).attr("height", sHeight); var arrows = svg.append("svg:defs").selectAll("marker") .data(["arrow"]) .enter() .append("svg:marker") .attr("id", String).attr("viewBox", "0 -5 10 10") .attr("refX", 10).attr("refY", -1) .attr("markerWidth", 4).attr("markerHeight", 4) .attr("orient", "auto").append("svg:path").attr("d", "M0,-5L10,0L0,5"); svg.append("rect").attr("width", sWidth).attr("height", sHeight); var nNodes = force.nodes(), nLinks = force.links(), node = svg.selectAll(".node"), link = svg.selectAll(".link"); function refresh() { link = link.data(nLinks); link.enter().append("path").attr("class", "link") .attr("marker-end", "url(#arrow)"); link.exit().remove(); node = node.data(nNodes); var nodeEnter = node.enter() .insert("g").attr("class", "node").call(force.drag); 5) Open the 'Visualizer.html' in any browser supporting the WebSocket. The page will connect via WebSocket to the monitor service running on port 5555. Now the Visualizer is ready to display and service/connection updates. C:\q\w32>Visualizer.html 6) We will run a simulator to span some dummy services and connections.

w32>q simulator.q 7) Now check the Visualizer page, you will see the simulated network updating real-time.

To stop the continuously running simulation, type the command loop:0b on the simulator service console.

22 b) Simulated Network Visualization

Here is a sample graph generated by D3.js using the simulated process network implemented in q/Kdb+.

23 Conclusion

The whitepaper demonstrated how the visualization platform could connect to a centralized service supporting the JSON and WebSocket and display the network of services interfacing with each other.

The service-network visualization example discussed in this paper could be useful for developers and support to visualize the health of the system for monitoring purpose. The code is not production-ready and has been kept concise for simplicity. Now the monitor service is ready to listen to any service/connection events. C:\q\w32>q simulator.q /Publish-Subscribe . .sim.prcsall:update id:i from ungroup update '$prc,'$grp ,'$"|"vs/:target from 'prc'grp'target!/:(("tp_fx_a";"tp";"");("tp_eqc_a";"tp ";"");("tp_fx_e";"tp";"");("tp_eqc_e";"tp";"");("tp_fx _u";"tp";"");("tp_eqc_u";"tp";"");("ctp_fx_a";"ctp";"t p_fx_a");("ctp_eqc_a";"ctp";"tp_eqc_a");("ctp_fx_e";"c tp";"tp_fx_e");("ctp_eqc_e";"ctp";"tp_eqc_e");("ctp_fx _u";"ctp";"tp_fx_u");("ctp_eqc_u";"ctp";"tp_eqc_u");(" rte_fx_a";"rte";"tp_fx_a");("rte_eqc_a";"rte";"tp_eqc_a");("rte_f p_eqc_e");("rte_fx_u";"rte";"tp_fx_u");("rte_eqc_u";"r te";"tp_eqc_u");("rdb_fx_a";"rdb";"rte_fx_a|gw_fx_a|gw _gf");("rdb_eqc_a";"rdb";"rte_eqc_a|gw_eqc_a|gw_ge");("rdb_fx_e";"rdb";"rte_fx_e|gw_fx_e|gw_gf");("rdb_eqc_e ";"rdb";"rte_eqc_e|gw_eqc_e|gw_ge");("rdb_fx_u";"rdb"; "rte_fx_u|gw_fx_u|gw_gf");("rdb_eqc_u";"rdb";"rte_eqc_u|gw_ _eqc_a";"hdb1";"gw_ge");("hdb1_fx_e";"hdb1";"gw_gf");("hdb1_eqc_e";"hdb1";"gw_ge");("hdb1_fx_u";"hdb1";"gw_g f");("hdb1_eqc_u";"hdb1";"gw_ge");("hdb_fx_a";"hdb";"g w_fx_a");("hdb_eqc_a";"hdb";"gw_eqc_a");("hdb_fx_e";"h db";"gw_fx_e");("hdb_eqc_e";"hdb";"gw_eqc_e");("hdb_fx _u";"hdb";"gw_fx_u");("hdb_eqc_u";"hdb";"gw_eqc_u");("

144 gw_fx_a";"gw";"");("gw_eqc_a";"gw";"");("gw_fx_e";"gw" ;"");("gw_eqc_e";"gw";"");("gw_fx_u";"gw";"");("gw_eqc
145 _u";"gw";"");("gw_ge";"gw";"");("gw_gf";"gw";"");("fh_lse";"fh";"tp_eqc_e");("fh_nyse";"fh";"tp_eqc_u");("fh
146 _nse";"fh";"tp_eqc_u");("fh_ret";"fh";"tp_fx_u");("fh_ebs";"fh";"tp_fx_e");("fh_int";"fh";"tp_fx_u");("c1e";
147 "c";"ctp_eqc_e|gw_ge");("c2e";"c";"ctp_eqc_e");("c3e"; "c";"ctp_eqc_u");("c4e";"c";"ctp_eqc_u");("c5e";"c";"g
148 w_gf");("d1e";"d";"ctp_eqc_e");("c1f";"c";"ctp_fx_e"); ("c2f";"c";"ctp_fx_e");("c3f";"c";"gw_gf");("c4f";"c";
149 "gw_ge");("c5f";"c";"ctp_fx_a");("d1f";"d";"ctp_fx_e") ;("gw_ca";"gw";"gw_gf|gw_ge")); .sim.prcs:delete
150 from .sim.prcsall where null target;
151 .sim.conMap:()!(); resetVars:{show "reset called";.sim.cnt:count .sim.prcs;.sim.pubNodes:()}; newPrc:{[prc]
h:hopen 5555 ;.sim.conMap **??**prc‘prc] [1]

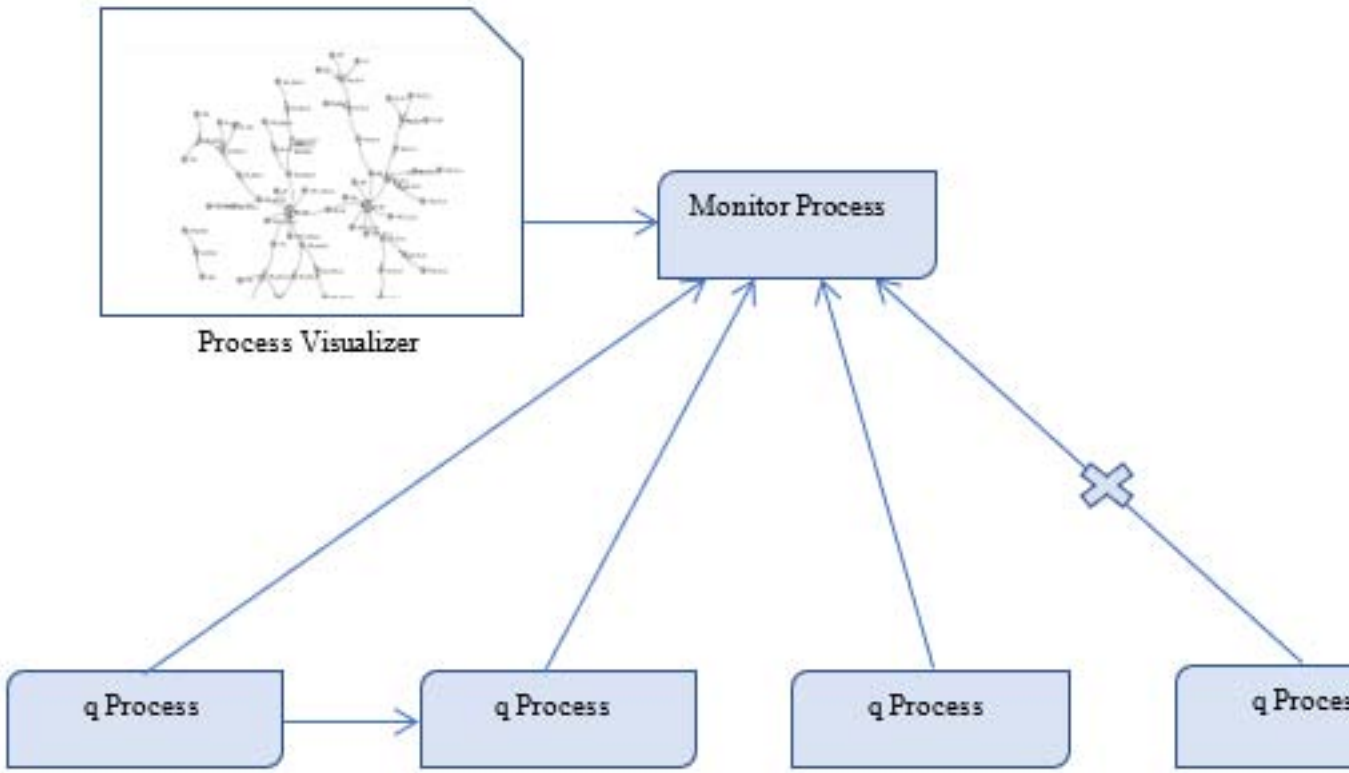

Figure 1:

152

[1]© 2018 Global Journals 1

1

Figure 2: Fig. 1 :

Appendix below contains the complete implementation of the example in q, JavaScript and HTML. The example needs Kdb+ 3.0 or above. **??**‘addprc ;prc); ‘.pm.prclist upsert (.z.w;prc‘prc;prc‘grp); .ps.pub (‘add_prc_event ;prc) }; addcon :{[con] .log.info(‘addcon ;con); ‘.pm.conlist upsert **??**.z.w;con‘prc;con‘target)

[References Références Referencias] , `https://linkurio.us/blog/panama-papers-how-linkurious-enables-icij-t` `2.D3.jshttps://d3js.org/3.https://kx.com/solutions/` *References Références Referencias* 1.

[enter() .append(”svg:marker”) .attr(”id svg:defs”).selectAll(”marker”) .data] ‘enter() .append(”svg:marker”) .attr(”id’. *svg:defs”).selectAll(”marker”) .data*, (var arrows = svg.append. String).attr(”viewBox”, ”0 -5 10 10”) .attr(”refX”, 10).attr(”refY”, -1) .attr(”markerWidth”, 4).attr(”markerHeight”, 4) .attr(”orient”, ”auto”).append(”svg:path”).attr(”d”, ”M0,-5L10,0L0,5”))

[links(nLinks).linkDistance(50).charge(-200) .on(”tick] *links(nLinks).linkDistance(50).charge(-200) .on(”tick*, (tick)

[rect”).attr(”width”, sWidth).attr(”height] *rect”).attr(”width”, sWidth).attr(”height*, (sHeight)

[var svg = d3.select(”body”).append(”svg”).attr(”width”, sWidth).attr(”height] *var svg = d3.select(”body”).append(”svg”).attr(”width”, sWidth).attr(”height*, (sHeight)