Global Journal of Computer Science and Technology: F
Graphics & Vision
Volume 18 Issue 1 Version 1.0 Year 2018
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

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

By Ali Asgher Kapadiya

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

*Keywords:* *multi-service systems, visualization, Kdb+, q, D3.js, websocket, JSON, SVG.*

*GJCST-F Classification:* *C.2.1*


*Strictly as per the compliance and regulations of:*

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

Ali Asgher Kapadiya

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

*Keywords: multi-service systems, visualization, Kdb+, q, D3.js, websocket, JSON, SVG.*

## I. Introduction

Data 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.

### 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.

*Author: Kdb+ Analytics Specialist, Tier I Investment Bank, United Kingdom. e-mail: akthetechie@gmail.com*

### 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.

### 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, real-time calculation, user query gateways, etc.

## II. 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.

### 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.

*b) D3 Process visualizer*

Implemented using D3.js framework for the visualizing the service network.

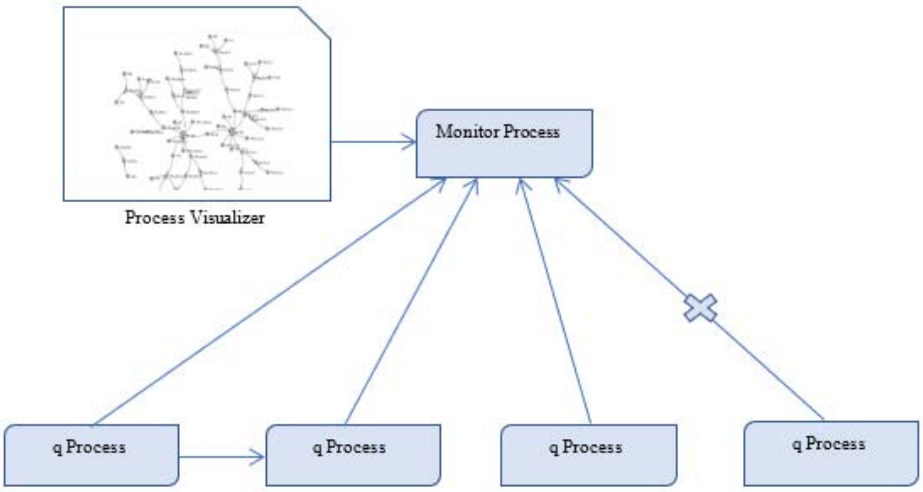


*Fig. 1:* High-level diagram

## III. 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.

## IV. 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.

- Add Prcocess

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

- 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.

- 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.

- Drop Connection

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.

*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.

*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.

- add_prc_event
- add_conn_event
- remove_prc_event
- remove_conn_event

## V. Web Component

It's an HTML & JavaScript component which contains the d3 visualization code and JavaScript functions for:

- 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.

*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.

```
function connect() {
    if ("WebSocket" in window) {
        connWS = new
WebSocket("ws://localhost:5555/");
        console.log("INFO : Requesting WebSocket
connection to q process");
        connWS.onopen = function(obj) {
console.log("INFO : WebSocket connection successful");
}
        connWS.onclose = function(obj) {
            console.log("INFO : WebSocket connection
dropped");
            connWS.close();
        }
        connWS.onmessage = function(obj) {
            var res = JSON.parse(obj.data);
            if (null !== res) {
                var funcName = res.shift();
                var params = res[0];
                window[funcName](params);
            }
        }
        connWS.onerror = function(obj) {
console.log(obj.data); }
    } else alert("WebSockets not supported in this
Browser");
}
```

*b) Monitor Event call-back methods implementation*

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

```
    function add_prc_event(nodeObj) {
        addNode(nodeObj.prc, nodeObj.grp);
    }

    function add_conn_event(linkObj) {
        addLink(linkObj.prc, linkObj.target);
    }

    function remove_prc_event(nodeObj) {
        removeNode(nodeObj.prc);
    }

    function remove_conn_event(linkObj) {
        removeLink(linkObj.prc)
    }
```

*c) Initializing the D3 SVG object*

```
var force = d3.layout.force().size([sWidth, sHeight])

.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");
```

*d) Refreshing the nodes and links*

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

```
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);

    nodeEnter.append("image")
        .attr("xlink:href",
"https://image.ibb.co/dzEqFe/process.png")
        .attr("x", -8).attr("y", -8)
        .attr("width", 16).attr("height", 16);
    nodeEnter
        .append("text").attr("dx", 12).attr("dy",
".35em");
```

## VI. 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

Now the monitor service is ready to listen to any service/connection events.

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.

C:\q\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.

*b) Simulated Network Visualization*

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

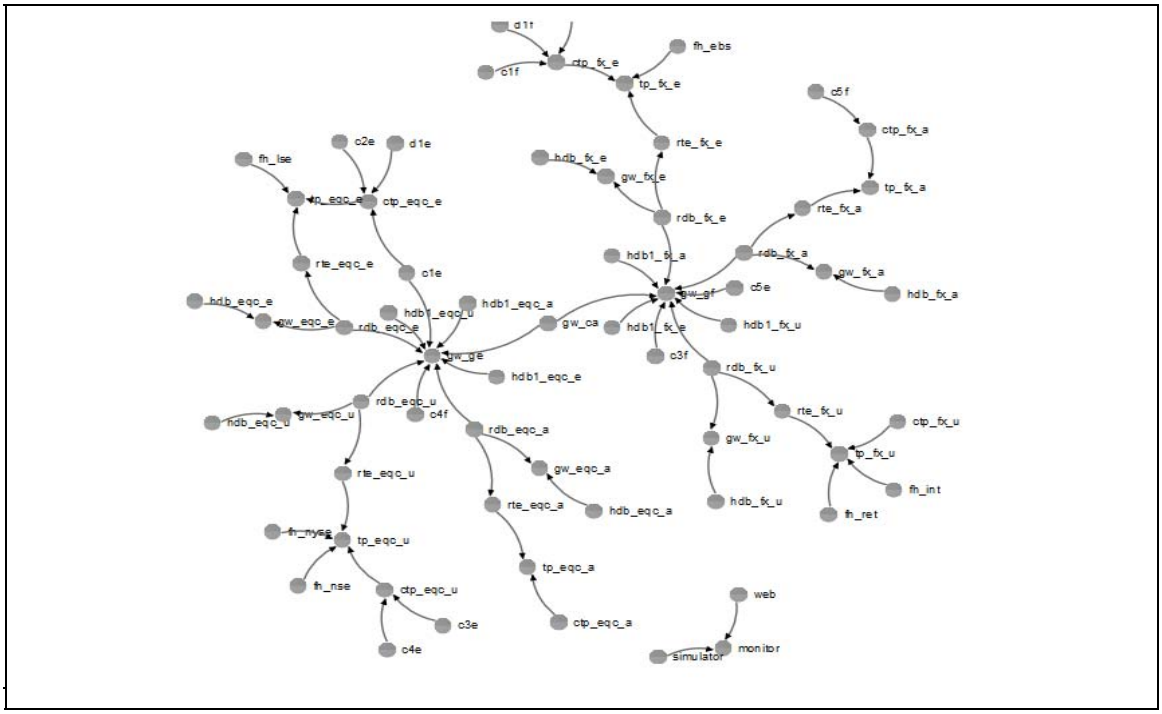


*Fig. 2:* Service Network

## VII. 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.

Appendix below contains the complete implementation of the example in q, JavaScript and HTML.

The example needs Kdb+ 3.0 or above.

## Appendix

*a) Monitor.q*

```
/Publish-Subscribe
.ps.subs:([h:()];u:());
.ps.sub:{[ch;u] `.ps.subs upsert (ch;u)};
.ps.pub:{[msg]  .ws.pub[;msg] each exec h from
.ps.subs };
.ps.unsub:{[ch] delete from `.ps.subs where h=ch};

/Webscoket
.ws.pub:{neg[x].j.j y};
.z.ws:{.log.info["WebSocket Request : ",.Q.s1
x];.ws.pub[.z.w] .debug.r:@[value;x;{`$x}]};
//Send intial process & connection list
.z.wo:{.log.info["WebSocket Connection open : ",
string x];.ps.sub[x;.z.h]  ;
{.ws.pub[x;(`add_prc_event;y)] }[x]each value
.pm.prclist; ; {.ws.pub[x;(`add_conn_event;y)]
}[x]each delete h from .pm.conlist};
.z.wc:{.log.info["WebSocket Connection close :
",string x];.ps.unsub[x]};

/Process Monitor
.pm.prclist:([h:()] prc:(); grp:());
.pm.conlist:( [] h:() ;prc:() ; target:());
```

```
addprc :{[prc] .log.info(`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); .ps.pub
(`add_conn_event;con) };
dropprc:{[prc] .log.info(`dropprc;prc);  .ps.pub
(`remove_prc_event ;prc)};
dropcon:{[con] .log.info(`dropcon;con);  .ps.pub
(`remove_conn_event;con)};

.log.info:{show x};

//Process handlers
.z.po:{[x] .log.info (.z.h;.z.u)};
.z.pc:{
    .d.p:prc:.pm.prclist[x];
    show "Stopping the process : ", string prc`prc;
    c:select from .pm.conlist where h=x ;
    delete from `.pm.prclist where h=x;
    delete from `.pm.conlist where h=x;
    dropprc[prc] ;
    dropcon each c
 };
```

b) *Visualizer.html*

```
<!DOCTYPE html>
<meta charset="utf-8">
<html>
<title>Service Network Visualizer</title>
<body onload="connect();">
    <script
src="https://d3js.org/d3.v3.min.js"></script>
    <script>
    var connWS;
    var nLinks = [];
    var nNodes = [];
    var sWidth = 900, sHeight = 600, radius = 8;
    var nodeIdMap = {} ,nodeCount = 0;

    function connect() {
        if ("WebSocket" in window) {
            connWS = new
WebSocket("ws://localhost:5555/");
            console.log("INFO : Requesting WebSocket
connection to q process");
            connWS.onopen = function(obj) {
console.log("INFO : WebSocket connection successful");
}
            connWS.onclose = function(obj) {
                console.log("INFO : WebSocket
connection dropped");
                connWS.close();
            }
            connWS.onmessage = function(obj) {
                var res = JSON.parse(obj.data);
                if (null !== res) {
                    var funcName = res.shift();
                    var params = res[0];
                    window[funcName](params);
                }
            }
            connWS.onerror = function(obj) {
console.log(obj.data); }
        } else alert("WebSockets not supported in this
Browser");
    }

    function ack(msg) { console.log('Ack -> ' + msg);
}

    function add_prc_event(nodeObj) {
addNode(nodeObj.prc, nodeObj.grp); }
    function add_conn_event(linkObj) {
addLink(linkObj.prc, linkObj.target); }
    function remove_prc_event(nodeObj) {
removeNode(nodeObj.prc); }
    function remove_conn_event(linkObj) {
removeLink(linkObj.prc) }
```

```
    function initMap() {
        nodeIdMap = {};
        nodeCount = 0;
        nNodes.forEach(function(d, i) {
nodeIdMap[d.id] = nodeCount++; })
    }
    function addNode(nodeName, g) {
        var node = { x: sWidth, y: sHeight, id:
nodeName, grp: g };
        var n = nNodes.push(node);
        nodeIdMap[nodeName] = nodeCount++;
        refresh();
    }
    function addLink(srcId, targetId) {
        if (srcId != targetId) {
            var s = {}, t = {};
            nNodes.forEach(function(curNode) {
                if (typeof curNode.id != "undefined")
{
                    if (curNode.id == srcId) { s =
curNode; }
                    if (curNode.id == targetId) { t =
curNode; }
                }
            });
            nLinks.push({ source: s, target: t });
        };
        refresh();
    }
    function removeNode(nodeName) {
        if (nodeName == "") return;
        nLinks = nLinks.filter(function(curLink) {
            return curLink.source.id !== nodeName &&
curLink.target.id !== nodeName;
        });
        force.links(nLinks);
        var i = nodeIdMap[nodeName];
        if (typeof i != "undefined") {
nNodes.splice(i, 1); }
        initMap();
        refresh();
    }


    function removeLink(linkObj) {
        nLinks = nLinks.filter(function(l) { return
!(l.source.id == linkObj.prc && l.target.id ==
linkObj.target); });
        force.links(nLinks);
        initMap();
    }

    initMap();

    var force = d3.layout.force().size([sWidth,
sHeight])

.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");

    refresh();

    function tick() {
        link.attr("d", function(d) {
            var dx = d.target.x - d.source.x, dy =
d.target.y - d.source.y;
            var angle = Math.atan2(dy, dx);
            var offsetX = radius * Math.cos(angle);
            var offsetY = radius * Math.sin(angle);
            dr = Math.sqrt(dx * dx + dy * dy);
            return ("M" + (d.source.x + offsetX) + ","
+ (d.source.y + offsetY) +
                "A" + dr + "," + dr + " 0 0,1 " +
                (d.target.x - offsetX) + "," +
(d.target.y - offsetY)
            );
        });
        node.attr("transform", function(d) { return
"translate(" + d.x + "," + d.y + ")"; });
    }

    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);

        nodeEnter.append("image")
            .attr("xlink:href",
"https://image.ibb.co/dzEqFe/process.png")
            .attr("x", -8).attr("y", -8)
            .attr("width", 16).attr("height", 16);
        nodeEnter
            .append("text").attr("dx", 12).attr("dy",
".35em");

        node.select('text').text(function(d) { return
d.id; });
        node.exit().on('mousedown.drag',
null).remove();
        force.start();
    }


    </script>
</body>

</html>
<style>
rect {
    fill: none;
    pointer-events: all;
}
.node {
    fill: #000;
}
.link {
    stroke: #999;
}
.node text {
    pointer-events: none;
    font: 10px sans-serif;
}
path.link {
    fill: none;
    stroke: #666;
    stroke-width: 1.5px;
}
</style>
```

c) *Simulator.q*

```
.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_fx_e";"rte";"tp_fx_e");("rte_eqc_e";"rte";"t
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_u|gw_ge");("hdb1_fx_a";"hdb1";"gw_gf");("hdb1
_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");("
gw_fx_a";"gw";"");("gw_eqc_a";"gw";"");("gw_fx_e";"gw"
;"");("gw_eqc_e";"gw";"");("gw_fx_u";"gw";"");("gw_eqc
_u";"gw";"");("gw_ge";"gw";"");("gw_gf";"gw";"");("fh_
lse";"fh";"tp_eqc_e");("fh_nyse";"fh";"tp_eqc_u");("fh
_nse";"fh";"tp_eqc_u");("fh_ret";"fh";"tp_fx_u");("fh_
ebs";"fh";"tp_fx_e");("fh_int";"fh";"tp_fx_u");("c1e";
"c";"ctp_eqc_e|gw_ge");("c2e";"c";"ctp_eqc_e");("c3e";
"c";"ctp_eqc_u");("c4e";"c";"ctp_eqc_u");("c5e";"c";"g
w_gf");("d1e";"d";"ctp_eqc_e");("c1f";"c";"ctp_fx_e");
("c2f";"c";"ctp_fx_e");("c3f";"c";"gw_gf");("c4f";"c";
"gw_ge");("c5f";"c";"ctp_fx_a");("d1f";"d";"ctp_fx_e")
;("gw_ca";"gw";"gw_gf|gw_ge"));
.sim.prcs:delete from .sim.prcsall where null target;

.sim.conMap:()!();
resetVars:{show "reset called";.sim.cnt:count
.sim.prcs;.sim.pubNodes:()};
newPrc:{[prc] h:hopen 5555 ;.sim.conMap[prc`prc]:h;
neg[h] (`addprc;prc) ;h""};
newCon:{[c] h:.sim.conMap[c`prc]; neg[h]@(`addcon;c);
h""; };

loop:1b; //To continuously run the simulation
//loop:0b;   //To stop the simulation after adding the
nodes

reset:{ if[not loop;:()];
    $[0<count .sim.pubNodes;
        [   prc:first .sim.pubNodes ;
            .sim.pubNodes:1_.sim.pubNodes;
            if[prc in key .sim.conMap  ; hclose
.sim.conMap[prc]  ]
        ];
        resetVars[]
    ]};

pubNetwork:{
    .sim.cnt-:1;     n:();
    con:.sim.prcs[.sim.cnt];
    if[not con[`prc] in .sim.pubNodes; n,:con`prc;
.sim.pubNodes,:con`prc  ];
    if[not con[`target] in .sim.pubNodes;
n,:con`target; .sim.pubNodes,:con`target];
    nodes:distinct select prc, grp from .sim.prcsall
where prc in n;
    if[count nodes;newPrc  each nodes] ;
    newCon con };

simulation:{$[.sim.cnt>0;pubNetwork[];reset[]]};

resetVars[];
.sim.monh:hopen 5555;  //Connection to Monitor process

.z.ts:simulation;
.z.pc:{show "Connection dropped"};

value "\\t 1000";
```

## References Références Referencias


1. https://linkurio.us/blog/panama-papers-how-linkurious-enables-icij-to-investigate-the-massive-mossack-fonseca-leaks/
2. D3.js https://d3js.org/
3. https://kx.com/solutions/
4. https://code.kx.com/q/tutorials/install/#microsoft-windows