Advanced Dashboarding Tips & Techniques

Adding JavaScript Visualizations to Your Dashboards

Karthik Subramanian | Function1

Thursday – Sept. 28 | Washington, DC
Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2017 Splunk Inc. All rights reserved.
Founded in 2007; offices in Washington D.C., New York City, Toronto, and Chicago

One of Splunk’s premier professional services partners

More than 1,000 successful Splunk engagements spanning all industry verticals: Security, Finance, Energy, E-Commerce, Government, Defense, Healthcare, Entertainment, Retail, and Education


Our team of Splunk experts is credited with designing the base architecture for some of the largest Splunk deployments to-date and have aided in developing the standard for enterprise class governance and data onboarding
Enabling Development Mode

Creating a D3 Visualization

Creating a C3 Visualization
How to Enable Development Mode

▶ /etc/system/local/web.conf
  • [settings]
  • minify_js = False
  • minify_css = False
  • js_no_cache = True
  • cacheEntriesLimit = 0
  • cacheBytesLimit = 0
  • enableWebDebug = True

*Hint:* If you do a lot of development, it's very easy to put this file in an app and enable the app only when you’re developing code!
Why Enable Development Mode?

- Turning off minification
  - JavaScript
  - CSS
- Turning off caching
  - JavaScript
  - All other objects, regardless of size or number
- Turning on debug endpoints
  - For example, splat logs

If you don’t turn these settings off when you’re finished with development, the web interface will be a lot slower!
Build a D3 Visualization

Donut Chart: Photo from d3js.org
Create this file structure in /etc/apps/:

```
<appname>
    appserver
    static
        visualizations
            <visualization_name>
                src
                    visualization_source.js
                    webpack.config.js
                    visualization.js
                    visualization.css
                formatter.html
                package.json
                preview.png

    default
        visualizations.conf
        savedsearches.conf
    metadata
        default.meta
    README
```

Run the following commands from:

- `/etc/apps/<appname>/appserver/static/visualizations/<visualization_name>`:
  - `$ npm install`
  - `$ npm install --save d3`
  - `$ npm install --save jquery`
  - `$ npm install --save underscore`
Creating the Basics of visualization_source.js

```javascript
1. define(
2. ['jquery',
3. 'underscore',
4. 'vizapi/SplunkVisualizationBase',
5. 'vizapi/SplunkVisualizationUtils',
6. 'd3',
7.],
8. function($, _,
9. SplunkVisualizationBase,
10. SplunkVisualizationUtils,
11. d3)
12. {
13. return SplunkVisualizationBase.extend(
14. {
15. initialize: function() {
16. this.$el = $(this.el);
17. // Add a css selector class
18. this.$el.addClass('splunk-radial-meter');
19. },
20. getInitialDataParams: function() {
21. return {
22. outputMode: SplunkVisualizationBase.ROW_MAJOR_OUTPUT_MODE,
23. count: 10000
24. };
25. },
26. updateView: function(data, config) { // Cool stuff here later
27. });
28. });
29. });
30. });
31. ```
What Those Basics Coded For

- Set the dependencies
- Initialize the visualization
- Provide a CSS Selector
Writing the UpdateView Function

```javascript
updateView: function(data, config) {
  // Check for empty data
  if(data.rows.length < 1){
    return;
  }
  // Take the first data point
  datum = data.rows[0][0];
  // Clear the div
  this.$el.empty();
  // Pick colors and value
  var mainColor = 'green';
  var maxValue = 100;
  // Set height and width
  var height = 220;
  var width = 220;
  // Create a radial scale representing 75% of a circle
  var scale = d3.scale.linear()
    .domain([0, maxValue])
    .range([-Math.PI * 0.75, Math.PI * 0.75])
    .clamp(true);
  // Define arc
  var arc = d3.svg.arc()
    .startAngle(function(d){
      return scale(0);
    })
    .endAngle(function(d){
      return scale(d);
    })
    .innerRadius(70)
    .outerRadius(85);
  // SVG setup
  var svg = d3.select(this.el).append('svg')
    .attr('width', width)
    .attr('height', height)
    .style('background', 'white')
    .append('g')
    .attr('transform', 'translate(' + width / 2 + ', ' + height / 2 + ')');
  // Background arc
  svg.append('path')
    .datum(maxValue)
    .attr('d', arc)
    .style('fill', 'lightgray');
  // Fill arc
  svg.append('path')
    .datum(datum)
    .attr('d', arc)
    .style('fill', 'green');
}
```
What That Code Did

- Make sure that it doesn't freeze if there's no data
- Initialize it with the first data point
- Setup our SVG
- Setup the arc
- Draw the arcs
- Fill the arcs
- Place text to show the gauge readings
Adding CSS Stylesheets
/appserver/static/visualizations/<visualization_name>/visualization.css

```css
.meter-center-text {
  font-size: 50px;
  font-weight: 200;
  font-family: "Helvetica Neue", Helvetica, sans-serif;
}

/* Center the SVG */
.splunk-radial-meter svg {
  display: block;
  margin: auto;
}
```
### Updating visualizations.conf and Setting the App Metadata to Share

**visualizations.conf:**
- `<visualization_name>`
- `label = My Visualization`
- `description = "I made this!"

This lets Splunk know that the visualization exists!

**default.meta (or local.meta):**
- `[visualizations/<visualization_name>]`
- `export = system`

This lets other apps have access to the visualization!
Compiling the Visualization

- From the `/appserver/static/visualizations/<visualization_name>/` folder:
  - `$ npm run build`

- Run Splunk, use the visualization, and change settings/code as necessary

- You’ll need to recompile code when you change it, but no need to restart Splunk!
C3 Compared to D3
A Quick Primer

D3: Data-drive Documents
- Focuses on visualizations
- Based on Document Object Models
- Was meant to simplify/replace W3C DOM API

C3: Comfortable, Customizable, and Controllable
- Requires the D3 library
- Focuses on labeling, usability, and readability
- Simpler to code
- Is meant to extend, not replace, the D3 library
Starting from previous file structure, we are adding two libraries and three files:

- **Libraries**
  - d3.js in `/<app_name>/appserver/static/
  - c3.js in `/<app_name>/appserver/static/

- **Files**
  - custom_view.xml in `/<app_name>/default/data/ui/views/
  - customview.js in `/<app_name>/appserver/static/
  - demoview.js in `/<app_name>/appserver/static/`
Creating the View XML

1. `<dashboard script="customview.js">
2.   <label>Custom View</label>
3.   <row>
4.     <html>
5.       <h2>Count by Sourcetype</h2>
6.       <div id="mycustomview"></div>
7.     </html>
8.   </row>
9. </dashboard>`
What That Code Means

- It’s a very simple Splunk dashboard with just one element
- Uses an HTML element that has an id that we will bind to in the script
- In use, you could add more elements or more dashboard panels in addition to this one.
Creating customview.js

```javascript
1. /* customview.js */
2. require([
3.   "//static/app/customviewtutorial/demoview.js",
4.   "splunkjs/mvc/searchmanager",
5.   "splunkjs/mvc/simplexml/ready!"
6. ], function(DemoView, SearchManager) {
7. 
8.   var mySearch = new SearchManager({
9.     id: "mysearch",
10.    preview: true,
11.    cache: true,
12.    search: "index=_internal earliest=-1h | stats count by sourcetype"
13.  });
14. 
15.   var customView = new DemoView({
16.     id: "mycustomview",
17.    managerid: "mysearch",
18.    el: "#mycustomview"
19.  }).render();
20. });
```
What That Code Means and Does

- In the “require” statement at the top, it loads the “demoview” module, as well as other components such as the Splunk search manager. They are referenced in the function as “DemoView” and “SearchManager,” respectively.

- A “SearchManager” object is created, which contains the Splunk search which will power the custom visualization. A simple search here is used for example purposes.

- A “DemoView” object is created, which contains the visualization and then calls the “render” function to display it.
Creating demoview.js

1. //This configuration is required to ensure that c3 can find d3 when it is initializing
2. requirejs.config({
3.     paths: {
4.         "d3": "/static/app/customviewtutorial/d3",
5.         "c3": "/static/app/customviewtutorial/c3",
6.     }
7. })
What That Code Means and Does

- We want to make sure that C3 can find D3 when it is initializing
- Without this function/configuration, C3 will be “undefined” in the code
  - References to C3 will throw errors as it’s undefined
- This syntax is a CommonJS wrapper, and we have to load dependencies through require statements in the script
Creating demoview.js 2

1. // Base class for custom views
2. var SimpleSplunkView = require('splunkjs/mvc/simplesplunkview');
3. 
4. // Require Underscore.js to work with search results
5. var _ = require("underscore");
6. 
7. // Require d3 and c3 libraries to produce custom data visualizations
8. var d3 = require("d3");
9. var c3 = require("c3");
Define SplunkSimpleView as our base class for the custom views

Require dependencies:

- underscore.js
- d3.js
- c3.js
1. `var DemoView = SimpleSplunkView.extend({
2.   // Object to hold the data for the visualization
3.   var chartData = {
4.     seriesData: [
5.       seriesDataArray
6.     ],
7.     xTickValues: xTickValuesArray
8.   }
9.});
What That Code Means and Does

- Extend the SplunkSimpleView to create a DemoView object
- Create objects for our formatData and updateView functions
  - seriesDataArray stores the count of the sourcetypes
  - xTickValuesArray stores the list of the sourcetypes
- Populate a chartData object
  - Note: seriesData is a 2D array because the C3 chart expects a 2D array for the data.columns property
1. // Override this method to put the Splunk data into the view
2. updateView: function(viz, chartData) {
3.     //create the chart object using the c3 generate function
4.     var chart = c3.generate({
5.         //The div ID from the Splunk view
6.         bindto: '#mycustomview',
7.         data: {
8.             //The data for the series, in this case the count of each sourcetype
9.             columns: chartData.seriesData,
10.            //Create a bar chart from the "sourcetype" data series
11.            types: {
12.                sourcetype: 'bar'
13.            },
14.            axis: {
15.                x: {
16.                    //Setting the type of the x-axis
17.                    type: 'category',
18.                    //Setting the x-axis labels
19.                    categories: chartData.xTickValues,
20.                    //rotating the x-axis labels by 30 to better accommodate the length of some of the labels
21.                    tick: {
22.                        rotate: 30
23.                    }
24.                }
25.            }
26.        },
27.    });
28.});
What That Code Means and Does

- Puts Splunk data into the view
- Creates the chart object using the c3.generate function
- Binds to the div ID that we set in the customview.xml dashboard
- Passes in the data
  - Selects a bar chart
  - Sets the x-axis to a category (of sourcetypes)
  - Rotates the x-axis labels by 30 degrees so that certain long sourcetypes do not run into each other
Conclusion/Roadmap

- Development Mode
- D3 Visualizations
- C3 Visualizations
Thank You

Don't forget to rate this session in the .conf2017 mobile app