Tristan Hoffmann, IBM, 07/12/2015

Getting started with Data Visualization and Exploration using IBM Watson Explorer AppBuilder

Table of Contents

1 Introduction / Motivation

I've participated in countless projects involving IBM Watson Explorer Engine and AppBuilder and while most customers and even colleagues have been surprised by how fast we can create a simple, but powerful search UI / dashboard leveraging several sources and a pretty large size of data, getting started is likely not as straight-forward as it should be.

Some time ago I started documenting things for myself and thought I could share a few key insights and tricks with you to hopefully give you a kick-start!

2 Structured v.s. unstructured data

To visualize anything meaningfully in AppBuilder, a certain structure is absolutely necessary. Structure in this context means data should be somehow separated. E.g. we should know which information belongs to what object (e.g. a customer number has to be treated differently than a product number) and what kind of information we are dealing with (separating different things like IDs, descriptions, ratings etc. into fields).

If some or all of your data sources are not structured well enough, this just means some preprocessing is needed. This might be done in a converter or using the integration with Watson Content Analytics for more complex cases.

3 How to configure Collections in Engine

There are several settings, that you should always keep in mind, when indexing data to be used in AppBuilder. If you set those settings, you can prevent having to re-index the collections later and might not run into some problems, that are hard to debug.

Note

Obviously there is a reason, why some settings are not set by default.
Mostly it is because enabling them costs you some performance.
While you can safely use the settings I recommend for small collections, you should be careful with bigger collections or slow systems.

3.1 Enable Term Expansion Support

Term expansion support allows you to search using regular expressions, but is also required to use the quick-filter, that is enabled by default in entity list pages.

Term Expansion has to be enable per collection in Configuration → Indexing → Term expansion support. Just set Generate dictionaries = true.

3.2 Fast-index important attributes

Fast-indexing means to store some attributes in memory, for better performance.
All fields that are used for entity associations as well as for refinements have to be fast-indexed.
To fast-index those attributes simply set Configuration → Indexing → General → Fast Index to Modified and enter one attribute per line into the text box below.

3.3 Normalize variable names

If you are crawling multiple data sources that have logically identical attributes, like an incident id, you want to always use the same variable name.
This problem often occurs in databases, where you can have different prefixes, that prevent you from refining on a field across several collections.
You can map these variables using a converter while crawling to prevent those issues.

4 Entities

Entities describe different objects types, such as a product, employee or customer.
Watson Explorer Engine usually indexes on a document level, so you have to map the documents in the index to the appropriate entities in AppBuilder.
To visualize the contents of your data instead of the technical representation, the extraction of entities is the key idea. In AppBuilder all data belongs to a certain entity type.
The easiest way to separate entities is to already separate them on a collection level in Engine. If this is not possible you can use filters in the entity configuration in AppBuilder.

4.1 Associate data

4.2 Auto completion

4.3 Defining synonyms and spelling variations

In App Builder there is a feature called query expansion. You can find the documentation on configuring query expansion here:

http://www-01.ibm.com/support/knowledgecenter/SS8NLW_10.0.0/com.ibm.swg.im.infosphere.dataexpl.appbuilder.doc/t_de-ab-devapp-search-qe.html

The main thing you will need to do is make a collection in Engine that has a mapping of terms to other terms. The easiest way to make this collection is probably to just make a csv file and crawl it. Engine ships with a default expansion collection called ontolection-english-spelling-variations that you can take a look at to see the format the documents need to be in. Once you have the collection you just point App Builder at the collection and it will handle the rest.

4.4 Preview

The preview function in AppBuilder is supposed to show you an inline preview of the original documents behind the metadata that is displayed in the app.
If you are using spreadsheets, databases or csv-files to populate any kind of 360° view or dashboard, likely you don't want to show any preview and it won't be pretty anyways.
To disable the preview link just disable the caching in the respective collections in Data Explorer Engine. This should look like this:
…
.
If you happen to have any office documents or PDFs, that would actually make sense to preview to the user, but likely get a preview that only shows some weird text. This is because text caching is enabled by default, but rich-caching would be what you expect.
To fix this just enable rich-cache for the appropriate document types in your collections:

The most important document types are:

File type	Document type in Watson Explorer
HTML (*.htm *.html)	Text/html
XML (*.xml)	Text/xml
Plain Text (*.txt)	Text/plain
PDF (*.pdf)	Application/pdf
Excel (*.xls *.xlsx)	Application/excel Application/ms-ooxml-excel
Power Point (*.ppt *.pptx)	Application/powerpoint Application/ms-ooxml-powerpoint
Word Document (*.doc *.docx)	Application/word Application/word2 Application/ms-ooxml-word
Rich text (*.rtf)	Application/rtf

5 Faceting (provide a few ruby examples)

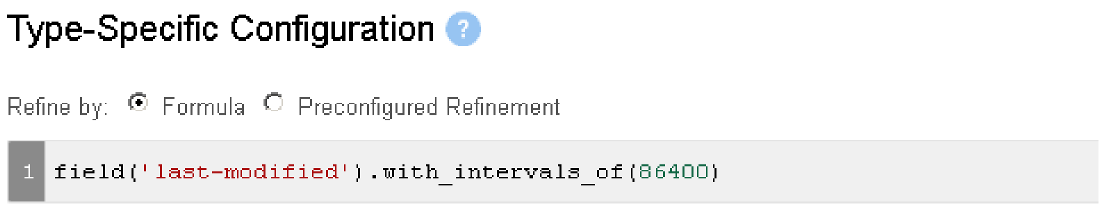
6 The search page

6.1 Refinements

The most simple refinement is based on a specific field. A standard field that is automatically populated, such as filetype (when crawling folders) is perfectly suited.
To refine make sure the specific field is fast-indexed in the underlying collections (in Engine go to Configuration → Indexing → Edit → General → Fast Indexed and set it to “Modified” then add the field in a new line). If a field you want to refine by is not fast-indexed, your refinement will not display at all.

The refinement widget can easily be added to the search screen from the AppBuilder configuration. Go to Pages & Widgets → search detail page and click create widget → choose refinement.
You only have to add a name that is displayed (Id) and field("Fieldname") into the "Enter ruby Code here" field. Save the refinement and add it to one of the visible columns. If a search result contains elements with the field specified, the refinement widget should appear.

If you want to refine by date, make sure to specify the desired granularity and select



Type-Specific Configuration

Refine by: [✓ Formula] Preconfigured Refinement

```
field('last-modified').with_intervals_of(86400)
```

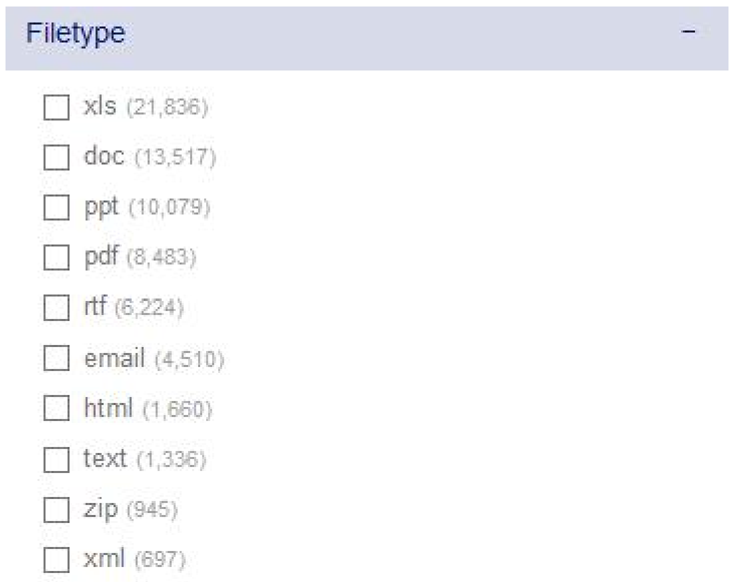
For example 60 Seconds * 60 Minutes * 24 hours = 86400 Seconds = 1 day

6.1.1 Allow refining by several values of one field (or-logic)

Example: You want to refine by priority 3 and 5, but this is not possible easily

Solution: Use field('fieldname).with_or_logic

Background: By default the faceting will use and logic. When you switch it to or logic the widget will use checkboxes and will allow multiple selections within a facet.



Filetype –

- ☐ xls (21,836)
- ☐ doc (13,517)
- ☐ ppt (10,079)
- ☐ pdf (8,483)
- ☐ rtf (6,224)
- ☐ email (4,510)
- ☐ html (1,660)
- ☐ text (1,336)
- ☐ zip (945)
- ☐ xml (697)

Figure: Refinement with or-logic

6.2 Customizing the result display

6.2.1 Limit the amount of text displayed

Example: You want to show descriptions but truncate very long ones

Solution: Enter “<p style='max-height: 6.5em;'+values.uniq.join(', ')+”</p>” into the Value formula on the entity configuration page

Note: Modify max-height depending on your line-height and desired length

7 Ruby examples

7.1 Widget display conditions

7.1.1 Only display widget, when a variable is set

Example: You want to display details about a certain information linked to the entity, but only if the information is present for the current item

Assumption: One-dimensional fields (non-array) are used

Code: subject['fieldname'].first *(replace fieldname with the actual value)*

Explanation: If the field is empty, the first value will be nil, which evaluates to false

7.1.2 Only display widget, when a variable contains a certain value

Example: Only display detailed information if hasDetails = true

Assumption: hasDetails contains the string “true” instead of a boolean value

Code: subject['hasDetails'].equals(“true”)

7.1.3 Only display widget, when an array contains a certain value

Example: A support ticket might contain related records of different types (e.g. Incident, Interaction etc.). You create a widget to show all the interactions linked to a ticket, but this widget should only be displayed if the array field contains an interaction number starting with IN.

Assumption: The source actually contains multiple elements of the same property, so that engine creates an array of the values.

This does not work, if all values are stored in a different way, e.g. comma separated in the index.

Code: subject['field'].index {|x| x.match /IN/}

Explanation: The code just runs a regular expression over all values in the field array.
If one field matches the regular expression the code will return true and your widget is displayed.

7.2 Access to variables

7.2.1 Access to the query string (search detail page)

Solution: context.subject.query

7.2.2 Access entity variables

Solution: subject['varname'].first (returns only first element if array)

Solution: subject['varname'].values.uniq.join(', ')

(returns all array elements as one comma separated string)

7.3 Widgets

7.4 Search API

7.4.1 Query against an entity type

ntity_types('Name of your entity')

entity_types('Name of your entity').where(field('content').contains(“Example string”)).requesting(30)

7.4.2 Where conditions

7.5 Custom HTML formatting

7.6 Appendix

7.6.1 Xsl converter for WCA annotations

```
<xsl:template match="/">
 <vce>
   <xsl:for-each select="//document">
     <document>
       <xsl:for-each select="./content">
         <xsl:text disable-output-escaping="yes"><![CDATA[<content name="]]></xsl:text>
         <xsl:value-of select="@name" />
         <xsl:text disable-output-escaping="yes"><![CDATA[">]]></xsl:text>
         <xsl:value-of select="." />
```

```
          <xsl:text disable-output-escaping="yes"><![CDATA[</content>]]></xsl:text>
        </xsl:for-each>
        <xsl:for-each select="./Metadata/Facets/Facet">
          <!-- combine value of Path nodes to form content name -->
          <xsl:text disable-output-escaping="yes"><![CDATA[<content name="]]></xsl:text>
          <xsl:value-of select="./Path[1]" /><![CDATA[_]]><xsl:value-of select="./Path[2]" />
          <xsl:text disable-output-escaping="yes"><![CDATA[">]]></xsl:text>
          <!-- select value of Keyword node as content value -->
          <xsl:value-of select="./Keyword" />
          <xsl:text disable-output-escaping="yes"><![CDATA[</content>]]></xsl:text>
        </xsl:for-each>
      </document>
    </xsl:for-each>
  </vce>
</xsl:template>
```

7.6.2 Free normalizer xsl to rename <content name='textBody'> to <content name='snippet'>

```
<xsl:template match="node( ) | @*">
  <xsl:copy>
    <xsl:apply-templates select="@* | node( )" />
  </xsl:copy>
</xsl:template>
<xsl:template match="content[@name='textBody']">
  <content name="snippet">
    <xsl:value-of select="." />
  </content>
</xsl:template>
```

7.6.3