Watson Content Hub and Connect:Direct Integration

Authors:	Mehmet Cambaz
Version:	1.0
Date Created:	12.12.2018
Last Updated:	02.01.2019

Document Revision History

Revision	Date	Description	Author
1.0	12.12.2018	First Draft	Mehmet Cambaz
1.1	02.01.2019	Modification	Mehmet Cambaz

Document Approval History

Name	Signature	Date	Version Approved/Comments

1. Introduction

This document's purpose is to give helpful information on how to integrate IBM Connect:Direct (C:D) with IBM Watson Content Hub (WCH) to show capabilities of an on-premise software such as Connect:Direct could be used in Hybrid Cloud environments such as IBM Watson Content Hub via REST APIs.

There is a common use case in banks which credit card transactions are gathered from core banking (such as applications in z/OS or some other distributed application) then sent to 3rd party to generate the pdf/image which will be sent to bank's customer whom uses the credit card as an email statement. At our use case the data is sent by C:D to 3rd party; 3rd party processes the data, sends the credit card statement as pdf/image by C:D to the bank. Since the credit card statement is a digital asset needs to be shown at a web user interface easily, it is possible to leverage WCH as the digital asset management tool and show these at a HTML/JS single page application which is also hosted on WCH.

The components that are used to demonstrate a use case which is a step in a Connect:Direct process **assuming** that a file is copied to PNODE and all the JSON (JSON is lightweight data format that is used in REST APIs. For more details see: http://www.json.org/) transformations are done (It can be done via IBM Transformation Extender) and the content is ready to sent to C:D and C:D will send the pdf to WCH via REST APIs, WCH's one single page application website would be used to reach to the assets uploaded, bank customers could click the links to access their credit card statements, bills etc.

WCH API documentation:
https://developer.ibm.com/api/view/dx-prod:ibm-watson-content-hub:title-IBM_Watson_Content_Hub#doc

https://ibm-wch.github.io/wch-openapi-documentation/

- **Curl** for command line based REST API processing, could be publicly downloaded from here: https://curl.haxx.se/ (command line tool and library for transferring data with URLs) After downloading the relevant curl version, you can unzip and copy the "bin" folder to your virtual machine's C: folder so that you can use C:\bin as path to use curl

You can test the APIs with RESTClient firefox add on: https://addons.mozilla.org/en-US/firefox/addon/restclient/ or with Postman https://www.getpostman.com/

2. Content

Connect:Direct can call the REST APIs WCH provides via command line execution leveraging the curl libraries and executables. Sample curl usage for json POST action:

```
curl -H "Content-Type: application/json" -X POST -d
'{"username":"xyz","password":"xyz"}' http://localhost:3000/api/login
```

blue is the json data part

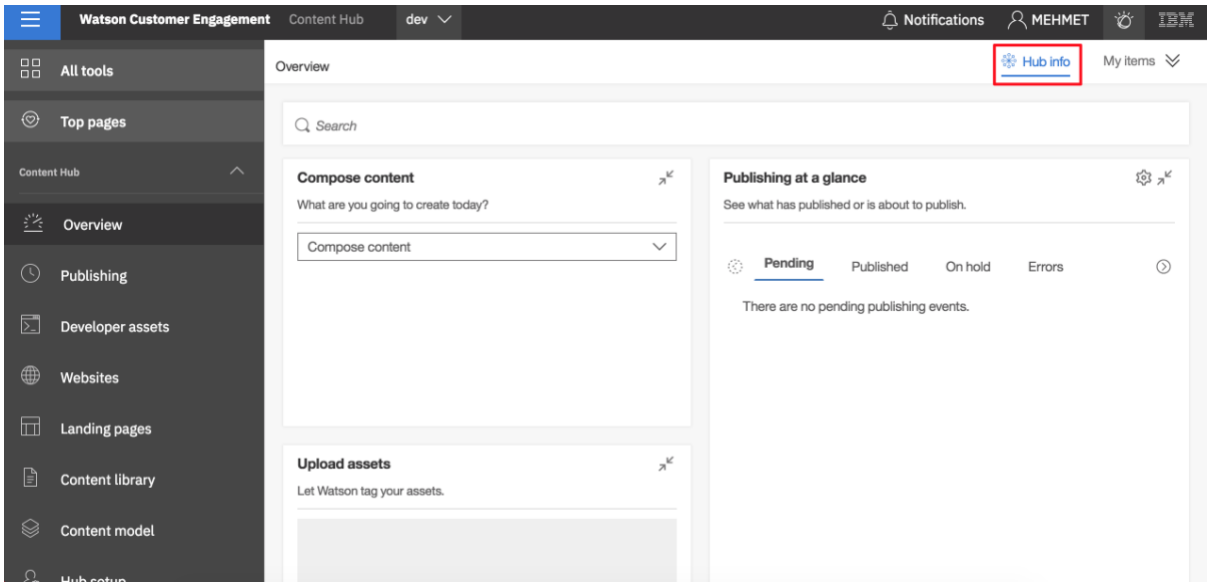
red is the url part to post

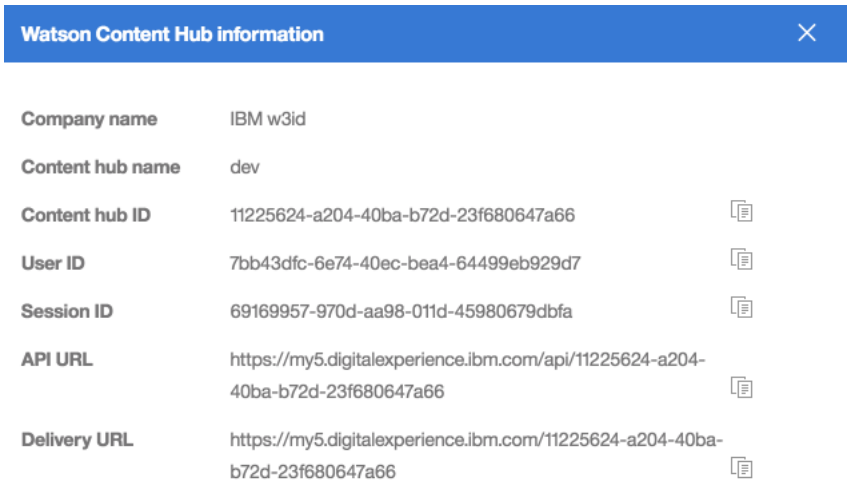
When using windows OS the escape characters should included in \ for example:

```
curl -X POST -H "Content-Type: application/json" -d "{ \"key1\": \"value1\"
}" http://localhost:3000/api/method
```

See more details here: https://stackoverflow.com/questions/7172784/how-to-post-json-data-with-curl-from-terminal-commandline-to-test-spring-rest

We need our WCH tenant information

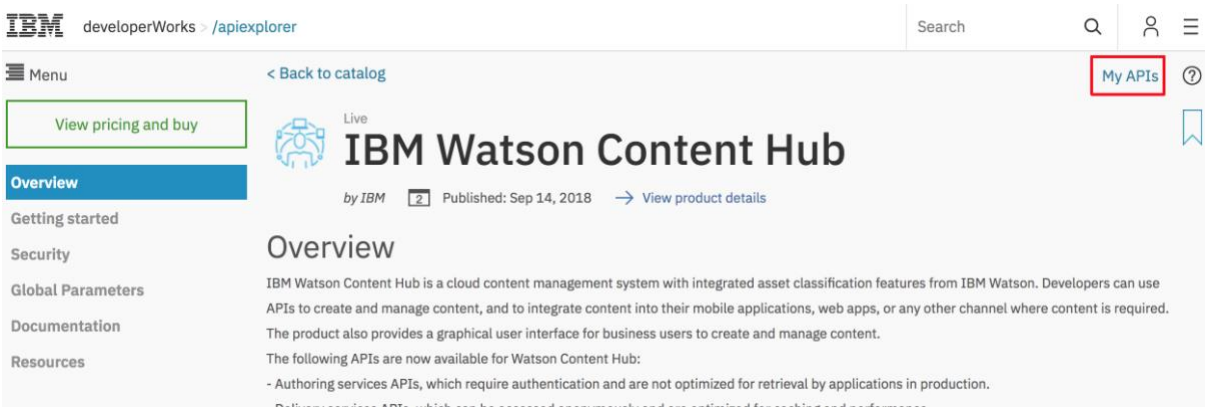


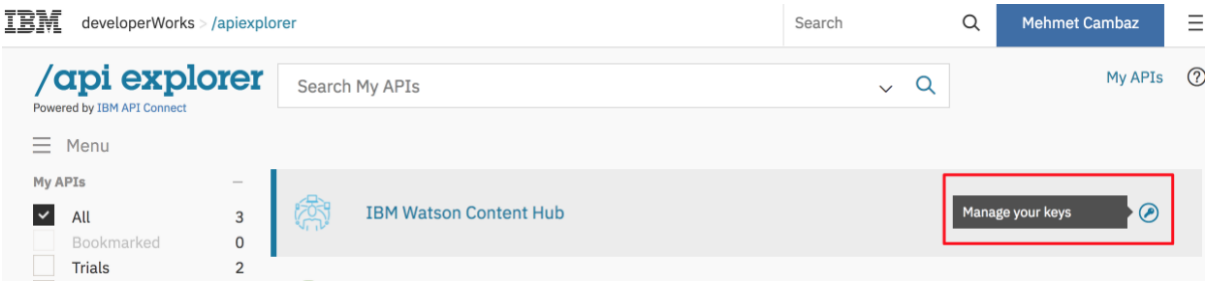


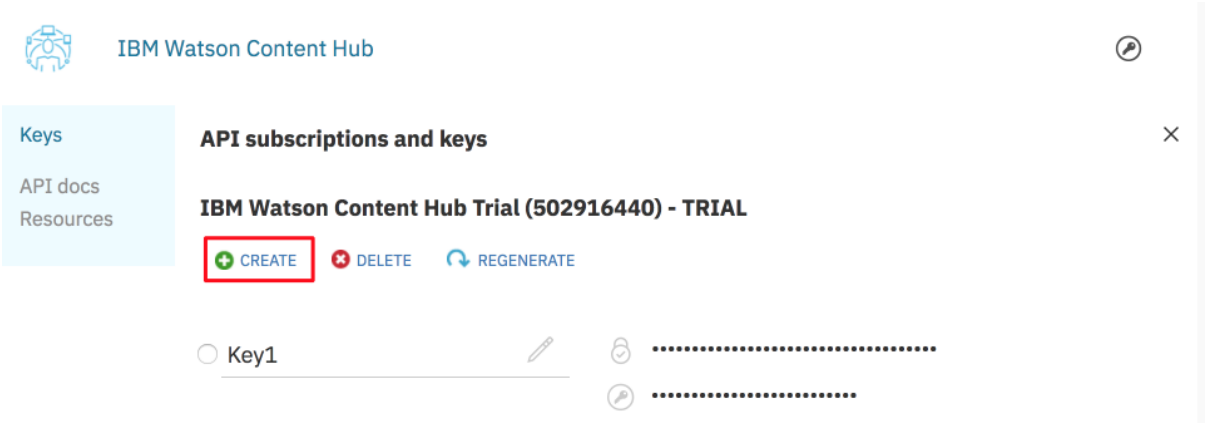
This is my tenant data, we will leverage the **API URL**

We also need an **APIKEY** to use for authentication, we can get it from here:

https://developer.ibm.com/api/view/dx-prod:ibm-watson-content-hub:title-IBM_Watson_Content_Hub







Save the secretkey to re-use when authenticating.

At our sample use case,

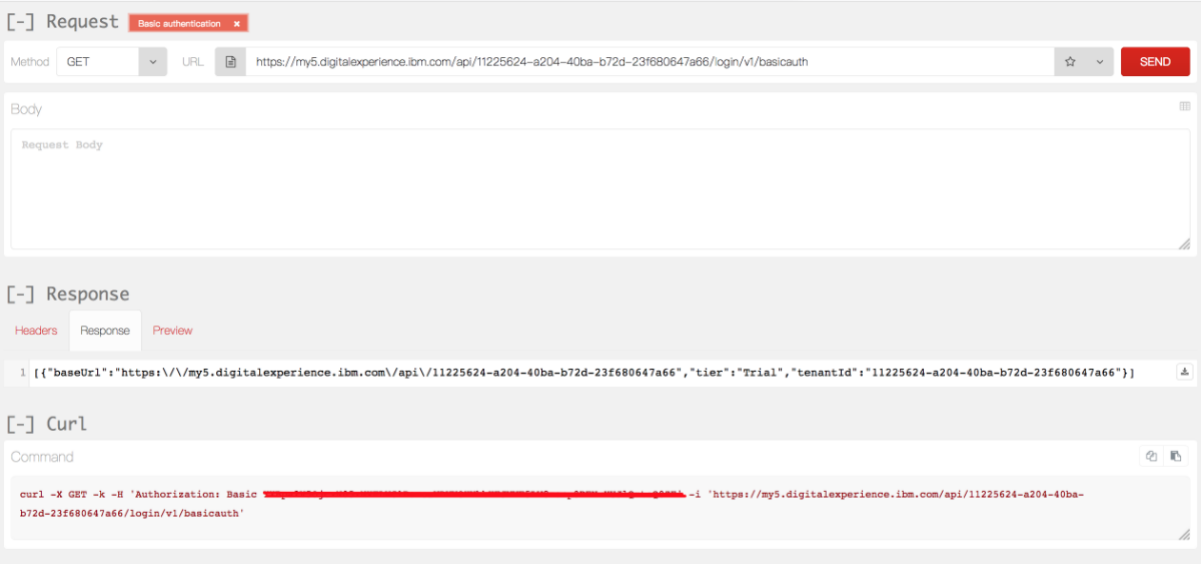
Step1: we need to authenticate with

```
/login/v1/basicauth
```

to WCH via REST API, we need to add basic authentication header **apikey** as Username and secretkey we generated at previous step as Password:

```
curl -X GET -k -H apikey@GENERATEDSTRING -i
'TENANTAPIURL/login/v1/basicauth'
```

(SAMPLE REQUEST AND RESPONSE via firefox RESTClient add-on)



Step2: We need to create the resource with

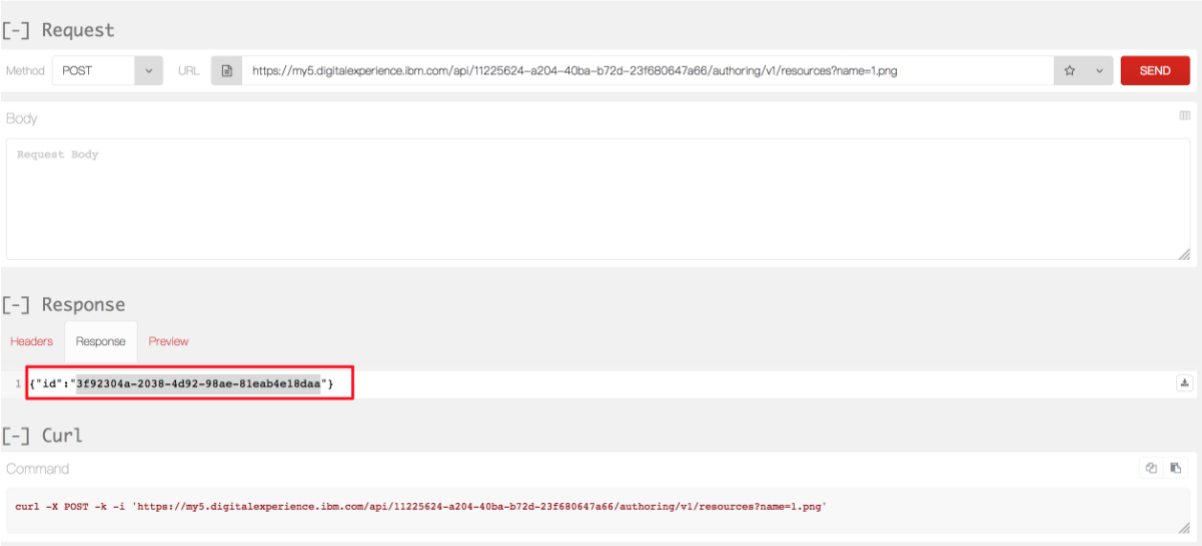
```
/authoring/v1/resources
```

to WCH via REST API, we need to give the filename part of url (our sample is 1.png)

```
curl -X POST -k -i 'TENANTAPIURL/authoring/v1/resources?name=1.png' >>
c:\bin\resourceId.log 2>~&1
```

it will return the resourceId we will use to author asset which we will use at next step

(SAMPLE REQUEST AND RESPONSE via firefox RESTClient add-on)



We assume that id got from response and data prepared for next step (it can be done via scripting or ITX map)

Step3: We need to create the asset with

```
/authoring/v1/assets
```

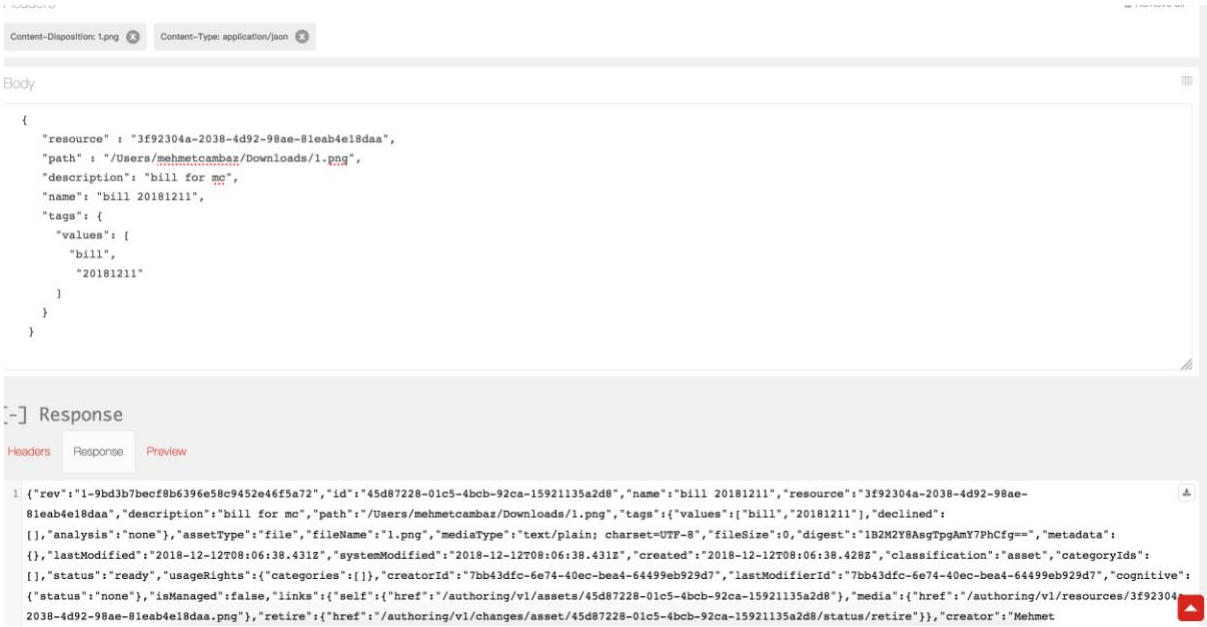
to WCH via REST API

```
curl -X POST -k -H 'Content-Disposition: 1.png' -H 'Content-Type:
application/json' -i 'TENANTAPIURL/authoring/v1/assets' -F 'image=@/1.jpg'
--data ' {
    "resource" : "3f92304a-2038-4d92-98ae-81eab4e18daa",
    "path" : "/1.png",
    "description": "bill for mc",
    "name": "bill 20181211",
    "tags": {
      "values": [
        "bill",
         "20181211"
      ]
    }
  }'
```

If you give path starting with /dxdam then it will be visible at WCH web UI, like this:

```
"path" : "/dxdam/1.png"
```

(SAMPLE REQUEST AND RESPONSE via firefox RESTClient add-on)



We save all three steps to this script to C:\bin\ as **postwithcurl.bat**

This will be used to send asset to WCH from C:D

In order to execute this batch file reading the file content of 1.png and posting the info via curl to WCH as an asset we will use Connect:Direct process like this:

(For more information on Connect:Direct processes:

https://www.ibm.com/support/knowledgecenter/en/CD_PROC_LANG/com.ibm.help.cdprocoverview.doc/cdprc_over_what_is_a_cd_process.html)

```
BMTEST1 PROCESS
    SNODE=CDW2008
    PNODEID=(Administrator,passw0rd)
    SNODEID=(Administrator,passw0rd)
BMJOB1 RUN JOB PNODE (DSN=Windows)
    SYSOPTS="cmd(C:\bin\postwithcurl.bat >> c:\bin\out.log 2>~&1)"
PEND
```

At this Connect:Direct process we use the submit job step to execute the batch file we prepared and using some scripting to generate logs about the action.

```
BMJOB1 RUN JOB PNODE (DSN=Windows)
    SYSOPTS="cmd(C:\bin\postwithcurl.bat >> c:\bin\out.log 2>~&1)"
```

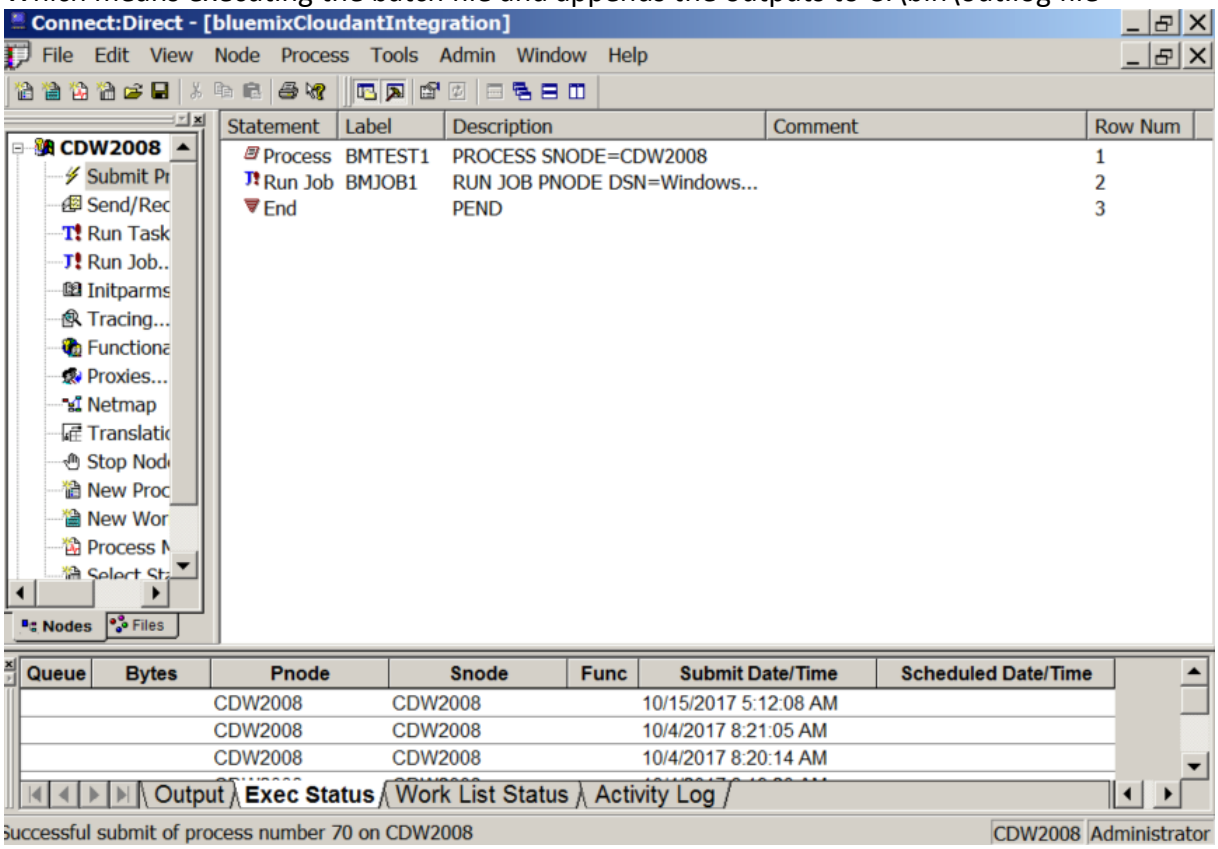
DSN=Windows means that this job is a Windows job execution

SYSOPTS="cmd()" means that it will run windows command line which is cmd.exe

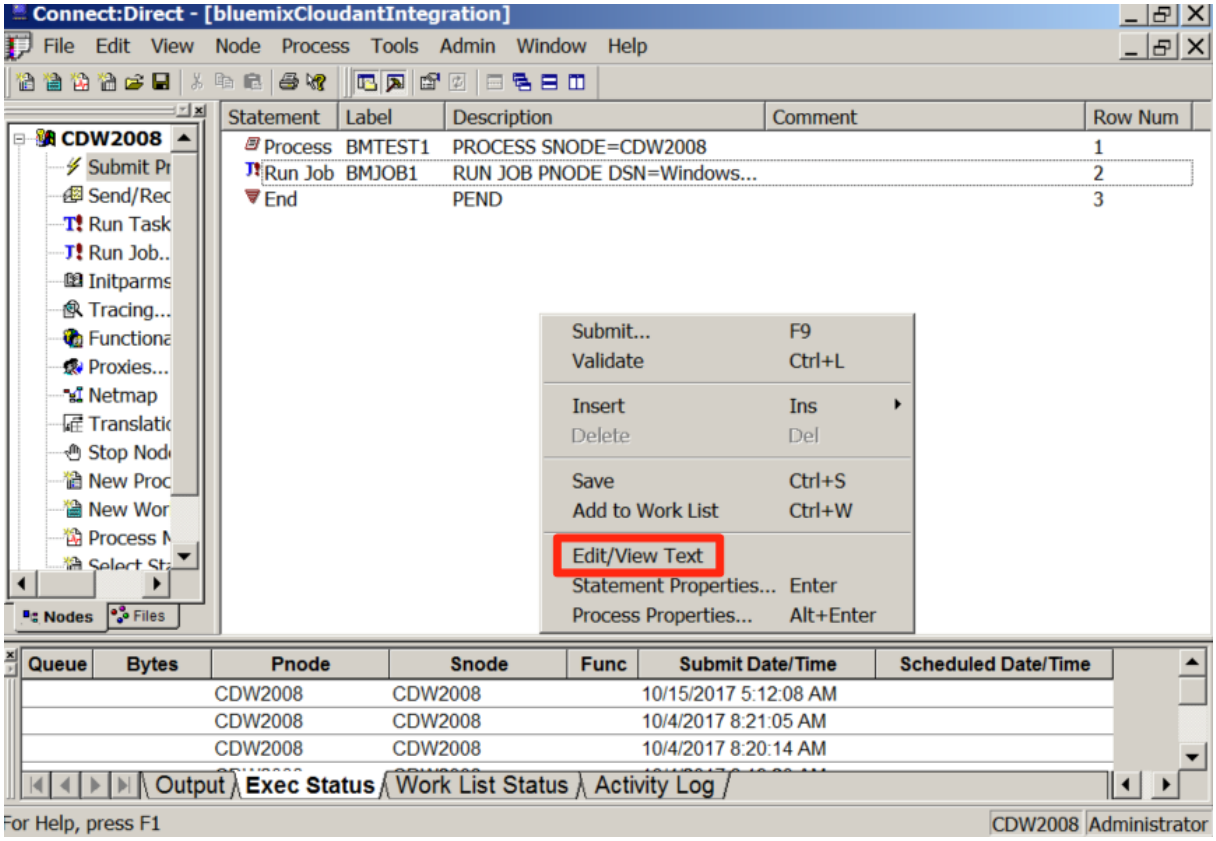
At the windows command line, it will execute the command below:

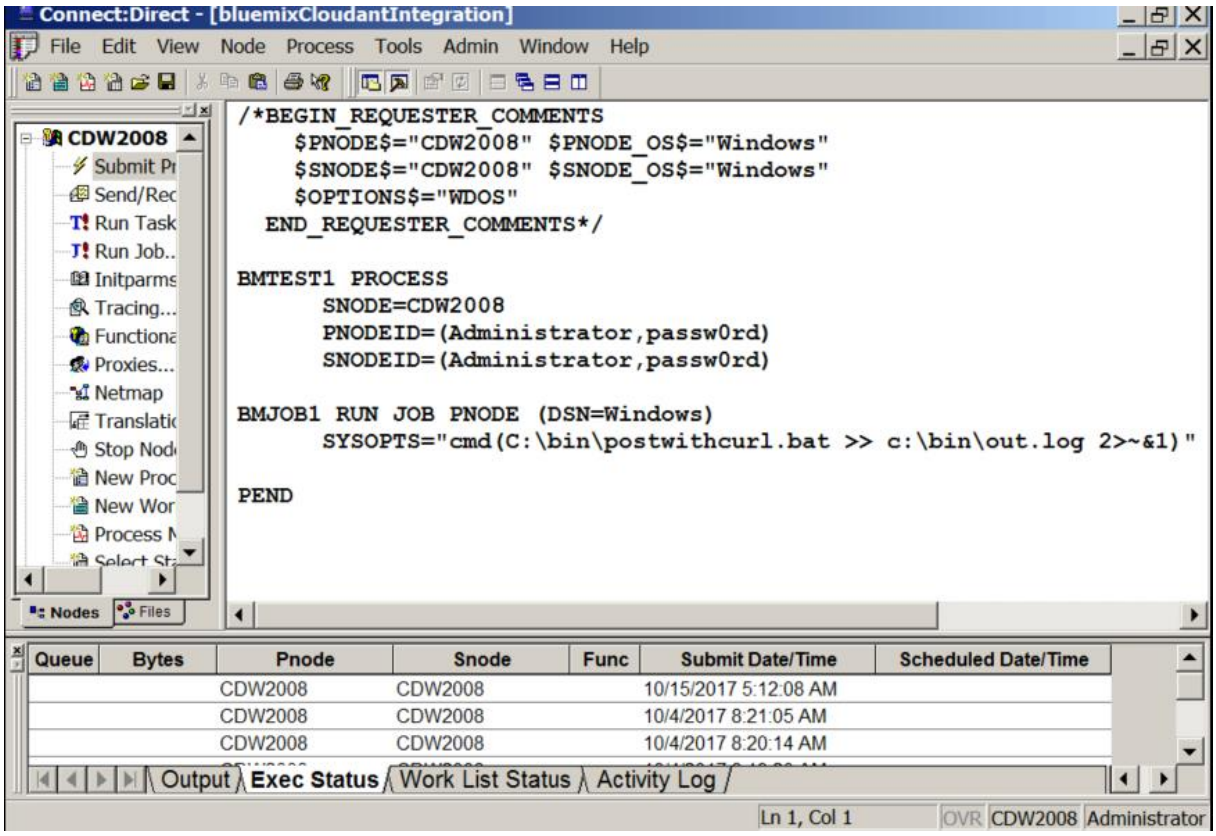
```
C:\bin\postwithcurl.bat >> c:\bin\out.log 2>~&1
```

Which means executing the batch file and appends the outputs to C:\bin\out.log file

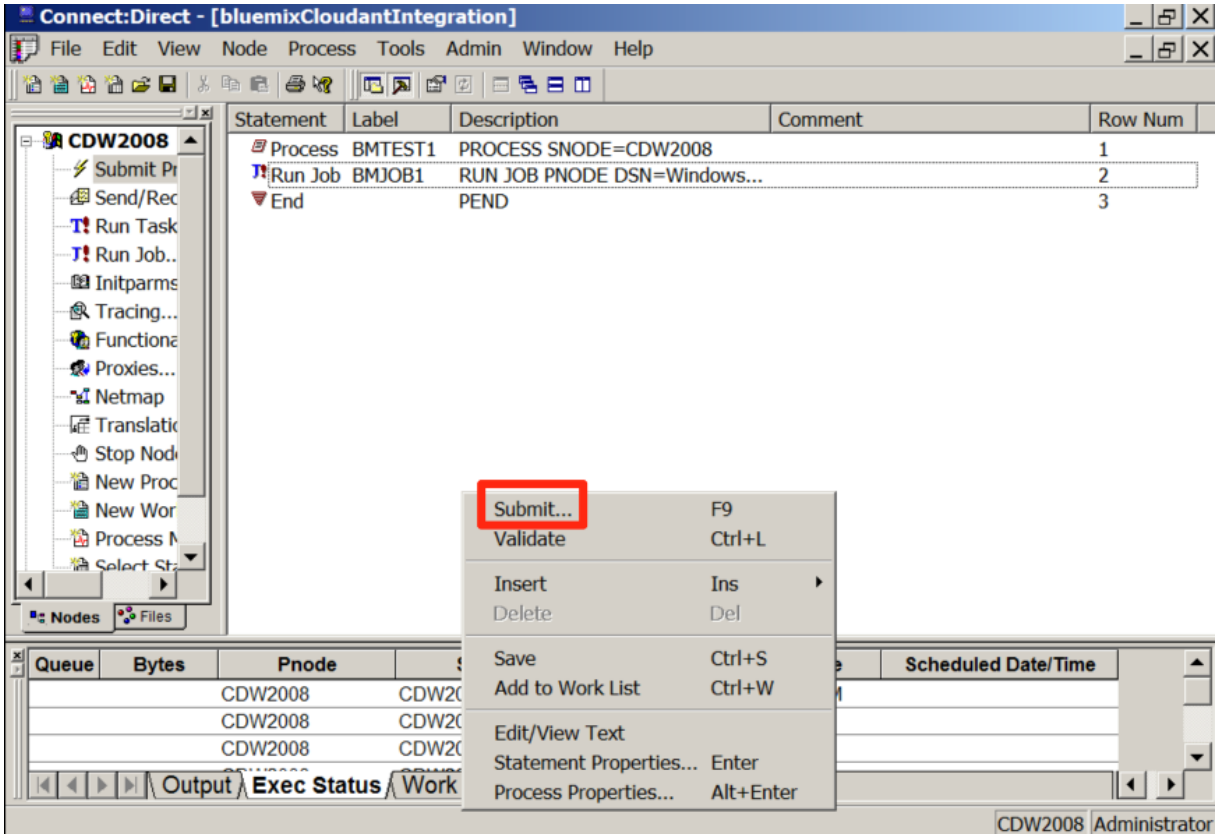


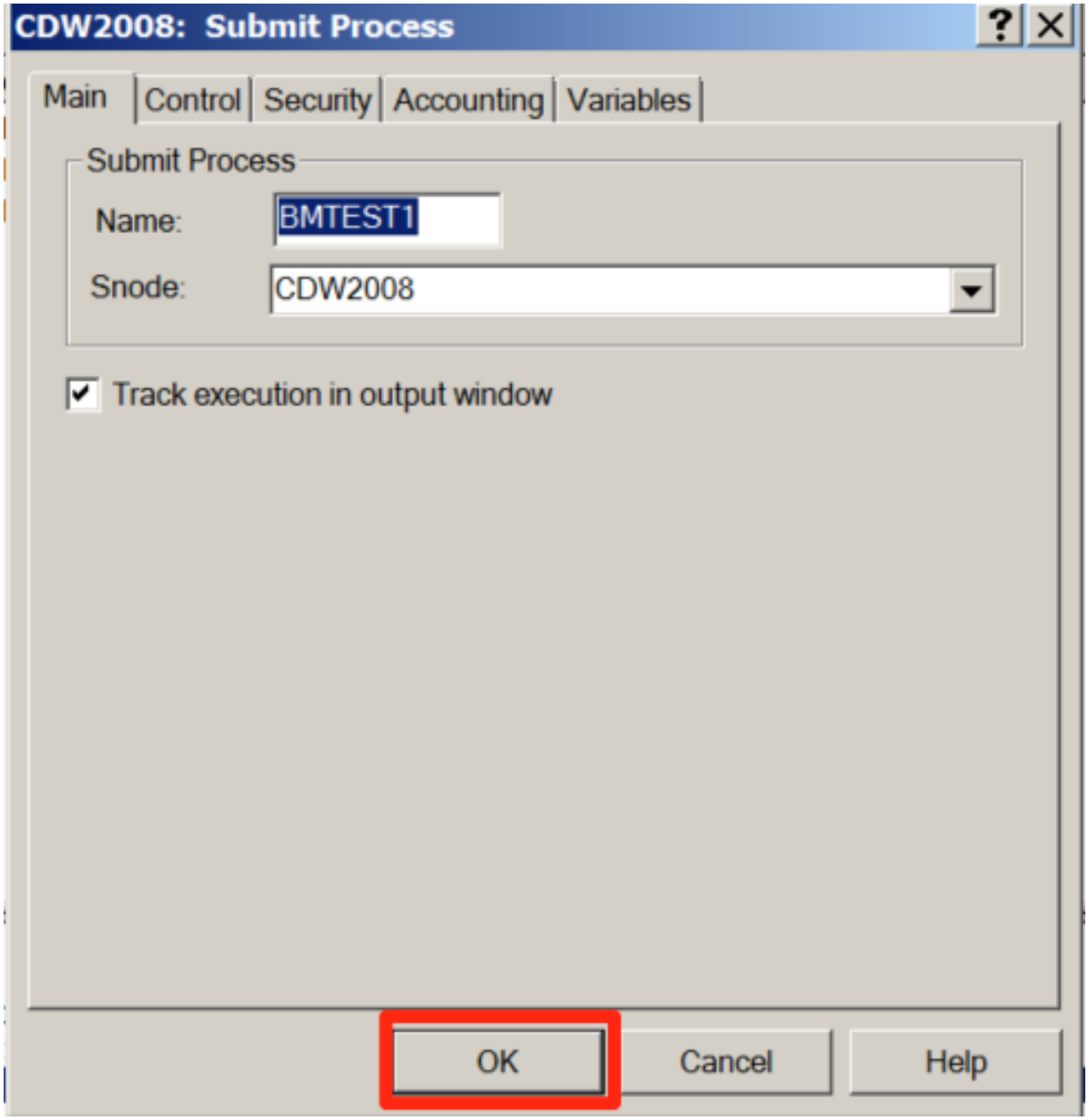
you can right click and click the "Edit/View Text" to see the Connect:Direct process source code





We will submit this process via right click to process and click the “Submit Process” menu item


Connect:Direct - [bluemixCloudantIntegration]
File Edit View Node Process Tools Admin Window Help
CDW2008
Submit Pr
Send/Rec
Run Task
Run Job..
Initparms
Tracing...
Functiona
Proxies...
Netmap
Translatic
Stop Nod
New Proc
New Wor
Process N
Nodes
Files
Statement Label Description Comment Row Num
Process BMTEST1 PROCESS SNODE=CDW2008 1
Run Job BMJOB1 RUN JOB PNODE DSN=Windows... 2
End PEND 3
Submit... F9
Validate Ctrl+L
Insert Ins
Delete Del
Save Ctrl+S
Add to Work List Ctrl+W
Edit/View Text
Statement Properties... Enter
Process Properties... Alt+Enter
Queue Bytes Pnode Scheduled Date/Time
CDW2008
CDW2008
CDW2008
Output Exec Status Work
CDW2008 Administrator


CDW2008: Submit Process
Main
Control
Security
Accounting
Variables
Submit Process
Name:
BMTEST1
Snode:
CDW2008
Track execution in output window
OK
Cancel
Help

Step4: See the assets from WCH single page application web site.

For this I had modified an open source WCH sample https://github.com/ibm-wch/sample-search-api which provides and example of search by WCH REST APIs

you need to go step by step to configure the sample.

I added below **index-bill.html** (modified version of provided index-nodejs-search.html) as the UI to be used by bank clients, it helps you search for bills:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Bills</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.6/handlebars.min.js"></script>
    <link rel="stylesheet" type="text/css" href="styles.css">
    <script type="text/javascript" src="app-nodejs-search.js"></script>
    <!--script>
    function updateInput(sel) {
        var value = sel.value;
        $('#sample-search-params').val(value);
    }
    </script-->
</head>

<body class="container">
    <div class="text-left" style="padding-bottom: 10px;">
        <button onclick="doSearch();" class="btn btn-primary" data-toggle="modal">Get My
Bills</button>
        <h2 class="color-a">Bill List</h2>
    </div>
    <!--div class="form-group row">
        <div class="col-xs-2">
            <label for="sample-search-params" class="col-form-label">Tenant Data: </label>
        </div>
        <!--div class="col-xs-8">
            <div>
                <select onchange="updateInput(this)" id="search-dropdown">
                    <option
value="q=*:*&fl=name,document,id,classification,type,status,categories,path&fq=classification
:(content OR asset)&fq=tags:(bill)">All bills</option>
```

```
                </select>
            </div>
        </div-->
    </div-->
    <!--div class="form-group row">
        <div class="col-xs-2">
            <label for="sample-search-params" class="col-form-label">Search
Parameters</label>
        </div>
        <div class="col-xs-10">
            <input class="form-control" type="text"
value="q=*:*&fl=name,document,id,classification,type,status,categories,path&fq=classification
:(content OR asset)&fq=tags:(bill)"
                id="sample-search-params">
        </div>
    </div-->
    <div id="contentContainer">
    </div>
    <!-- Handlebars template for the content table -->
    <script id="contentTemplate" type="text/x-handlebars-template">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th colspan="2">Num Found: {{searchResults.numFound}}</th>
                    <th>Category</th>
                    <th>Status</th>
                    <!-- <th>Last Modified</th>-->
                </tr>
            </thead>
            <tbody>
                {{#each searchResults.documents}}
                <tr>
                    <!--td>
                        <button onclick="showJson('{{@index}}');" class="btn btn-primary"
data-toggle="modal">JSON</button>
                    </td>
                    <td>
                        <button onclick="showDocument('{{@index}}');" class="btn btn-
primary">Document JSON</button>
                    </td-->
                    <td><a href="{{path}}" about="_blank" >{{name}}</a></td>
                    <td>{{type}}</td>
                    <td>{{categories}}</td>
                    <td>{{status}}</td>
                    <!-- <td>{{lastModified}}</td> -->
                </tr>
                {{/each}}
            </tbody>
        </table>
    </script>
    <script id="contentTemplate" type="text/x-handlebars-template">
```

```
        <table class="table table-striped">
            <thead>
                <tr>
                    <th colspan="2">Num Found: {{searchResults.numFound}}</th>
                    <th>Name</th>
                    <th>ID</th>
                    <th>Content Type</th>
                    <th>Category</th>
                    <th>Status</th>
                    <!-- <th>Last Modified</th>-->
                </tr>
            </thead>
            <tbody>
                {{#each searchResults.documents}}
                <tr>
                    <!--td>
                        <button onclick="showJson('{{@index}}');" class="btn btn-primary"
data-toggle="modal">JSON</button>
                    </td>
                    <td>
                        <button onclick="showDocument('{{@index}}');" class="btn btn-
primary">Document JSON</button>
                    </td-->
                    <td><a href="{{path}}" about="_blank" >{{name}}</a></td>
                    <td>{{id}}</td>
                    <td>{{type}}</td>
                    <td>{{categories}}</td>
                    <td>{{status}}</td>
                    <!-- <td>{{lastModified}}</td> -->
                </tr>
                {{/each}}
            </tbody>
        </table>
    </script>
    <script>

    // Button click handlers
    $('#sample-search-params').keypress(function(e) {
        if (e.which == 13) {
            doSearch();
            return false;
        }
    });
    // The search results are in variable "displayedSearchResults"
    function showJson(index) {
        $("#jsonContent").html(JSON.stringify(displayedSearchResults.documents[
                index], '',
            4));
        $("#myModal").modal();

    }
```

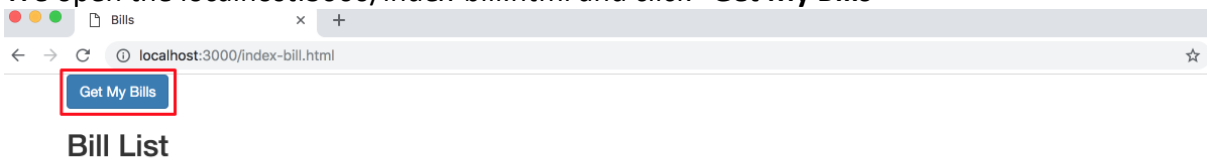
```
    function showDocument(index) {
        var doc = displayedSearchResults.documents[index]['document'];
        var docDisplay = 'no document';
        if (!(doc === undefined)) {
            docDisplay = JSON.stringify(JSON.parse(doc), '', 4);
        }
        $("#jsonContent").html(docDisplay);
        $("#myModal").modal();

    }

    // The currently displayed results
    var displayedSearchResults = {};

    function doSearch() {
        $("#contentContainer").html("");

        var params =
"q=*:*&fl=name,document,id,classification,type,status,categories,path&fq=classification:(cont
ent OR asset)&fq=tags:(bill)";

        //var params = $('#sample-search-params').val();
        //console.log('params: ', params);
        wchDoSearch(params, function(searchResults) {
            // console.log('json: ', searchResults);
            // update HTML from template
            var innerDivScript = $("#contentTemplate").html();
            var innerDivTemplate = Handlebars.compile(innerDivScript);
            var compiledHTML = innerDivTemplate({
                searchResults
            });
            $("#contentContainer").html(compiledHTML);



            displayedSearchResults = searchResults;
        });
    }

    $(document).ready(function() {
        doSearch();
    });
    </script>
    <!-- Modal -->
    <div id="myModal" class="modal" tabindex='-1' role="dialog">
        <div class="modal-dialog">
            <!-- Modal content-->
            <div class="modal-content">
                <div class="modal-header">
                    <button type="button" class="close" data-dismiss="modal">&times;</button>
```

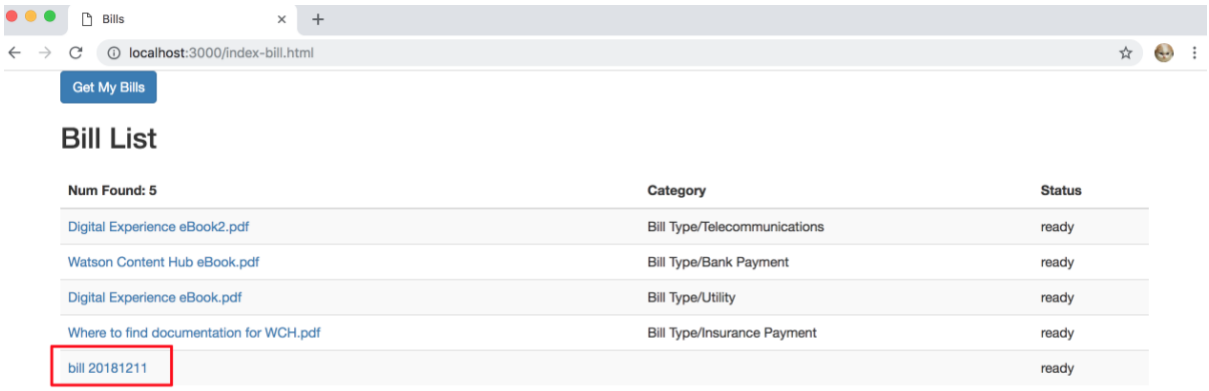
```
                    <h4 class="modal-title">JSON</h4>
                </div>
                <div class="modal-body">
                    <pre id="jsonContent">
                    </pre>
                    <div class="modal-footer">
                        <button type="button" class="btn btn-default" data-
dismiss="modal">Close</button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>

</html>
```

After adding the new html we will start node.js server by **node main.js** command at our commandline or terminal

```
sample-search-api-master — node main.js — 80×24
Last login: Wed Dec 12 12:04:32 on ttys000
Mehmets-MacBook-Pro:~ mehmetcambaz$ cd Downloads
Mehmets-MacBook-Pro:Downloads mehmetcambaz$ cd WCH-codes/
Mehmets-MacBook-Pro:WCH-codes mehmetcambaz$ cd sample-search-api-master
Mehmets-MacBook-Pro:sample-search-api-master mehmetcambaz$ node main.js
Listening on port 3000
Sample application that uses client JS to call WCH Delivery Search:
    http://localhost:3000/delivery-search.html
Sample application that uses client JS to call WCH Authoring Search:
    http://localhost:3000/authoring-search.html
Sample application that calls WCH search via Node.js:
    http://localhost:3000/index-nodejs-search.html
The Delivery Search API can be called with any search parameters like this:
    http://localhost:3000/api/delivery-search?q=*:*&wt=json&sort=name%20desc&row
s=1
The Authoring Search API can be called with any search parameters like this:
    http://localhost:3000/api/authoring-search?q=*:*&wt=json&sort=name%20desc&ro
ws=1
```

We open the localhost:3000/index-bill.html and click “**Get My Bills**”





There are links to show the uploaded assets.

In summary; this exercise shows that IBM Connect:Direct could be used to integrate with public, hybrid or private cloud environments easily and it is possible to send file content/record rows/items etc. to applications with REST APIs. Also it shows that Watson Content is capable of receiving information and files via REST APIs easily and able to host your own single page application web-sites for your needs.

Addendum:

(Thanks to Goktug Demir for below samples)

Other sample curl commands you can use:

Creating a resource

with name `test.png`

using the file in `/Users/Goktug/Desktop/istanbul_mobil.png`

```
curl -X POST -k -i -H 'Content-Type: image/png' --data-binary
"@/Users/Goktug/Desktop/istanbul_mobil.png" -b cookie.txt
'https://my3.digitalexperience.ibm.com/api/abab8a9e-07a9-4f25-9a7b-
193356fe3ac8/authoring/v1/resources?name=test.png'
```

Creating the asset from the resource

```
curl -X POST -k -H 'Content-Disposition: test.png' -H 'Content-Type: application/json' -i -b
cookie.txt 'https://my3.digitalexperience.ibm.com/api/abab8a9e-07a9-4f25-
9a7b-193356fe3ac8/authoring/v1/assets' --data ' {
"resource" : "6f65124f-8249-415a-9ea9-61680917aa8e",
"description": "demo demo3",
"status":"ready",
"tags": {
"values": [
"demo",
"demo2"
]
}
}'
```

His asset on Business Process integration between IBM B2B Integrator and WCH: https://w3-connections.ibm.com/communities/service/html/communityview?communityUuid=e9ee00af-9740-4ab1-aa42-250e098b952c#fullpageWidgetId=W7f9d5c89b222_40ae_bd66_af3e4358c099&file=78a8b03c-c875-4f9e-8783-9440cca3b758