***HCL Commerce V9.1***

***ElasticSearch – Crawl and Ingest Content Pages data***

**Implementation guide**

**Draft - Version 0.1**

**Jan 03, 2023**



**Document Source**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version #** | **Date** | **Version Description and Summary of changes** | **Editor** |
| 0.1 | Jan 03, 2023 | Draft - First version for review | Chaitanya Mulaguru |

DISCLAIMER:

The information contained and discussed in connection with this document is **confidential and proprietary to HCL**. The use and dissemination of this document and the information contained and discussed therewith are subject to confidentiality obligations and restrictions on use. Any information contained herein is subject to change at the sole discretion of HCL and its related entities and is not a binding commitment or representation on the development, release, or timing of any products, features, or functionality

This documented was created with the intent to showcase how-to work with NiFi. It was intended as a sample exercise of how to crawl content pages from website and ingest content into elastic search through nifi container for HCL Commerce V9.1 Search based on Elasticsearch.

The documentation was created using HCL Commerce Developer v9.1.11

Business Scenario / Functional Requirements

Ingest static content pages data

High Level Design (HLD)

create new custom index and ingest data in new schema.

Implementation guide steps

**Step 1:** Create a directory into nifi container (commerce/search-nifi-app:9.1.x.0) (This directory will be used to save crawled content for ingestion in ES)

1. docker exec -it -u 0 search\_nifi\_app bash
2. mkdir /opt/nifi/extDocs/
3. chown nifi:nifi /opt/nifi/extDocs
4. chmod 755 /opt/nifi/extDocs

**Step 2:** Create a directory into nifi container (commerce/search-nifi-app:9.1.x.0) (This directory will be used to save crawled content for ingestion in ES)

1. docker exec -u root -it search\_nifi\_app bash
2. mkdir /opt/nifi/nifi-crawl/
3. cd /opt/nifi/nifi-crawl/
4. yum install nodejs (make sure it is above v16)
5. yum install npm
6. npm install puppeteer --unsafe-perm=true --allow-root --ignore-scripts
7. yum install chromium
8. node node\_modules/puppeteer/install.js

Note : If you face any issue with missing libraries, kindly install those libraries

yum install -y nodejs alsa-lib.x86\_64 atk.x86\_64 cups-libs.x86\_64 gtk3.x86\_64 libXcomposite.x86\_64 libXcursor.x86\_64 libXdamage.x86\_64 libXext.x86\_64 libXi.x86\_64 \

libXrandr.x86\_64 libXScrnSaver.x86\_64 libXtst.x86\_64 pango.x86\_64 xorg-x11-fonts-misc xorg-x11-fonts-Type1 libdrm libgbm nss

1. cp -r /root/.cache/puppeteer /home/nifi/.cache/
2. vi crawl\_script.js

copy and paste the content of the below file in “crawl\_script.js”



**Step 3:** Import the following connectors into Runtime registry (commerce/search-registry-app:9.1.x.0).

1. sudo docker cp custom-crawl-SiteContentIndexSchemaUpdate.json search-registry-app:/opt/nifi-registry/flows/
2. sudo docker cp custom-crawl-SiteContentIndexSchemaUpdateConnector-attachment.json search\_registry\_app:/opt/nifi-registry/flows/.
3. sudo docker cp custom-crawl-SiteContentIndexDatabaseConnectorPipe-Attachment.json search\_registry\_app:/opt/nifi-registry/flows/.

**Open the nifi registry container, run the following command**

docker exec -it -u 0 search\_registry\_app bash

**Run the following commands from the registry terminal**

1. /opt/nifi-registry/scripts/import\_flow.sh custom-crawl-SiteContentIndexSchemaUpdate /opt/nifi-registry/flows/custom-crawl-SiteContentIndexSchemaUpdate.json
2. /opt/nifi-registry/scripts/import\_flow.sh custom-crawl-SiteContentIndexSchemaUpdateConnector-attachment /opt/nifi-registry/flows/custom-crawl-SiteContentIndexSchemaUpdateConnector-attachment.json
3. /opt/nifi-registry/scripts/import\_flow.sh custom-crawl-SiteContentIndexDatabaseConnectorPipe-Attachment /opt/nifi-registry/flows/custom-crawl-SiteContentIndexDatabaseConnectorPipe-Attachment.json

**Step 4:** Create a Connector using Ingest Swagger. Open the postman and add the following details

URL - http://localhost:30800/connectors

Method - POST

Body - json body is available into next page.

{

"name": "auth.sitecontent",

"description": "This is the connector for the sitecontent processing",

"pipes": [

{

"name": "custom-crawl-SiteContentIndexSchemaUpdate"

},

{

"name": "custom-crawl-SiteContentIndexSchemaUpdateConnector-attachment"

},

{

"name": "custom-crawl-SiteContentIndexDatabaseConnectorPipe-Attachment",

"properties": [

{

"name": "Database Driver Location(s)",

"value": "${AUTH\_JDBC\_DRIVER\_LOCATION}",

"scope": {

"name": "Database Connection Pool",

"type": "CONTROLLER\_SERVICE"

}

},

{

"name": "Database Connection URL",

"value": "${AUTH\_JDBC\_URL}",

"scope": {

"name": "Database Connection Pool",

"type": "CONTROLLER\_SERVICE"

}

},

{

"name": "Database User",

"value": "${AUTH\_JDBC\_USER\_NAME}",

"scope": {

"name": "Database Connection Pool",

"type": "CONTROLLER\_SERVICE"

}

},

{

"name": "Password",

"value": "${AUTH\_JDBC\_USER\_PASSWORD}",

"scope": {

"name": "Database Connection Pool",

"type": "CONTROLLER\_SERVICE"

}

}

]

},

{

"name": "AliasLink",

"label": "AliasLink - SiteContent",

"properties": [

{

"name": "environment.alias.name",

"value": "auth",

"scope": {

"name": "Alias Index",

"type": "PROCESS\_GROUP"

}

},

{

"name": "index.alias.name",

"value": "sitecontent",

"scope": {

"name": "Alias Index",

"type": "PROCESS\_GROUP"

}

}

]

},

{

"name": "Terminal"

}

]

}

**Step 4a):** The “Remove Duplicates” process uses a “DistributedMapCacheClientService” as temporary storage. The “DistributedMapCacheClientService” needs to be configured.

In the nifi dashboard, right click on a blank space on the “NiFi Flow” background and select “Configure”.

Select the “CONTROLLER SERVICES” tab.

Click the “+” button to add a new service.

Double click on “DistributedMapCacheServer” to set up the cache server on port 4557.

Click on the “lightning” icon of the “DistributedMapCacheServer” to enable the service.

Click the “+” button to add a new service.

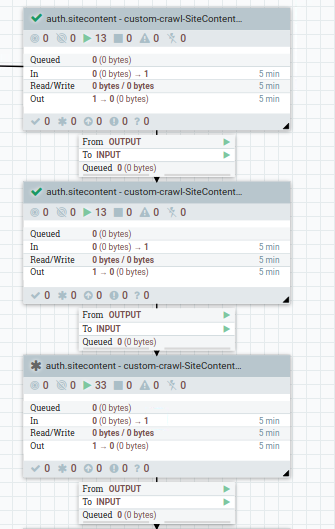
Double click on “DistributedMapCacheClientService” to set up the cache client service.

Click on the “gear” icon of the “DistributedMapCacheClientService” to go to configuration. Set the “Server Hostname” in the “PROPERTIES” tab to the value “localhost”

Click on “APPLY” to save the configuration.

Click on the “lightning” icon of the “DistributedMapCacheClientService” to enable the service.

**Step 5**: After running the connector, the following five process group pipes will be available in nifi



Graphical user interface, application

Description automatically generated

**Process Groups(pipes) :** In the above image using the four process groups.

1. **custom-crawl-SiteContentndexSchemaUpdate :** This process group will be used to set new schema, it will skip if the schema will already be available in elastic search.

We can use existing schema also, just update the index name in the following properties file for processor set sitecontent schema name.

Go to the process group “**Set SiteContent index name**” and change the index name

**NOTE:** Change index name as below

**Index.name** - .${environment.name}.${param.storeId}.staticcontent.${time.id}

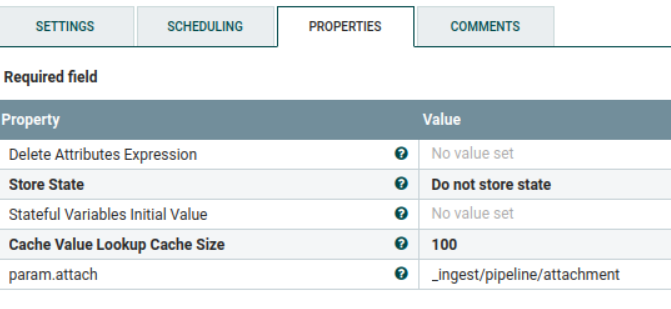
Go to process group **“Populate SiteContent Index schema**” process group and update the replace value with below one



1. **custom-crawl-SiteContentIndexSchemaUpdateConnector-attachment:**  This process group will used for enabling the attachment setting in elastic search, it will skip if the attachment setting will already be available.

The following setting has been used by default; we can update as per our requirement.

The param.attach is available in ‘Set sitecontent attachment ’ processor.



The following json is available in ‘Populate SiteContent Index schema’ processor.

Here we can add/update keyword, this keyword will be used for ingesting/searching unstructured/sitecontent/staticontent data.

**{**

**"description" : "Extract attachment information",**

**"processors" : [**

**{**

**"attachment" : {**

**"field" : "data",**

**"indexed\_chars\_field" : "max\_size",**

**"properties": [ "content", "title", "keywords", "content\_type", "content\_length" ]**

**}**

**}**

**]**

**}**

1. **custom-crawl-SiteContentIndexDatabaseConnectorPipe-Attachment :** This process group are being used for crawling the content pages from website, save into file structure then save in directory, read file content from directory, encode into base64 and ingest file content into the elastic search.

This process group are having multiple processors to ingest the files into elastic search.

The two following process group are added in this nifi flow.

* 1. **Crawl Content:** This process group are being used for the crawling the content pages for the storied passed, it will also put the content in files and save into directory.

Graphical user interface, Excel

Description automatically generated

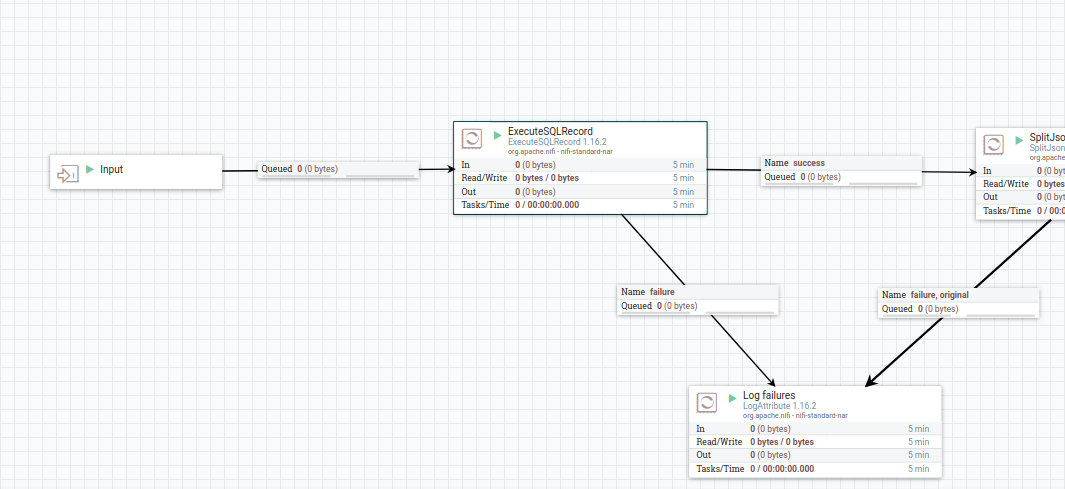
Inside crawl content, we first query the database to fetch the content pages data along with the hostname of the website. Below is the process group(ExecuteSQLRecord) that queries the db

**Note :**  Run the below queries in db before doing this

INSERT INTO storeconf VALUES (1,'wc.search.indexcrawl.hostname','http://{hostname}/wcs/shop/en/auroraesite',0);

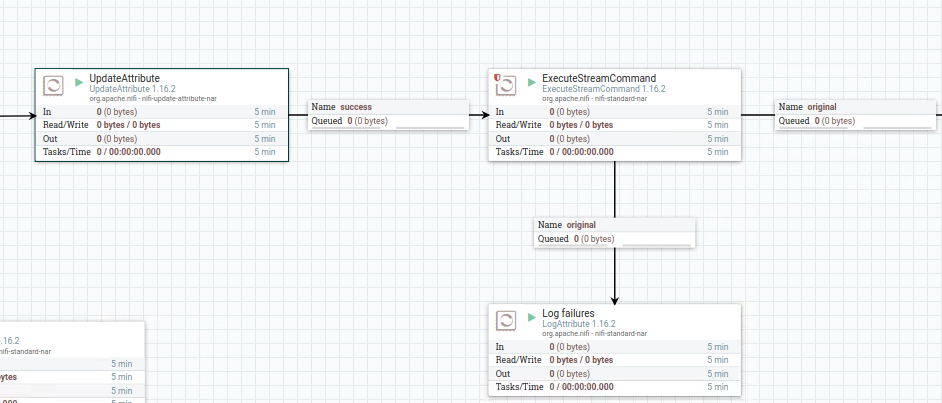
INSERT INTO storeconf VALUES (11,'wc.search.indexcrawl.hostname','http://{hostname}/Emerald/',0);

This data is required for the query in the ExecuteSQLRecord process group



We fetch the data from the db and split the data and join urlhost and urlpath to form the url required for crawling

We then pass the url to “ExecuteStreamCommand” process group



Graphical user interface, text, application, email

Description automatically generated

**Note :** Please update the command Arguments to below value

/opt/nifi/nifi-crawl/crawl\_script.js;${url};/opt/nifi/extDocs/

This triggers the script that we have created by passing inputs “url” and “path to store the crawled data”

* 1. **Fetch URL and Invoke Search:** This process group are being used to get all saved files from local directories then encode in base64, then ingest in Elastic search.

Diagram, schematic

Description automatically generated

Diagram

Description automatically generated

**Step 6:** Start all the four process groups, then navigate to inside each process group and right click on the Nifi Flow then select the ‘Enable Transmission’.

Graphical user interface

Description automatically generated

**Steps 8:** After starting the process group run the following url from postman

POST-

https://localhost:5443/wcs/resources/admin/index/dataImport/build?connectorId=auth.sitecontent&storeId=11

For checking status:

GET- https://localhost:5443/wcs/resources/admin/index/dataImport/status?jobStatusId={jobid}

**Steps 9:**  Now, we can verify indexed data, we can pass keyword as what we have given during setting of the attachment:

POST - localhost:30200/auth.11.sitecontent/\_search

Body: With Content available in file or name

{ "query": { "bool": {

"must": [

{

"query\_string": {

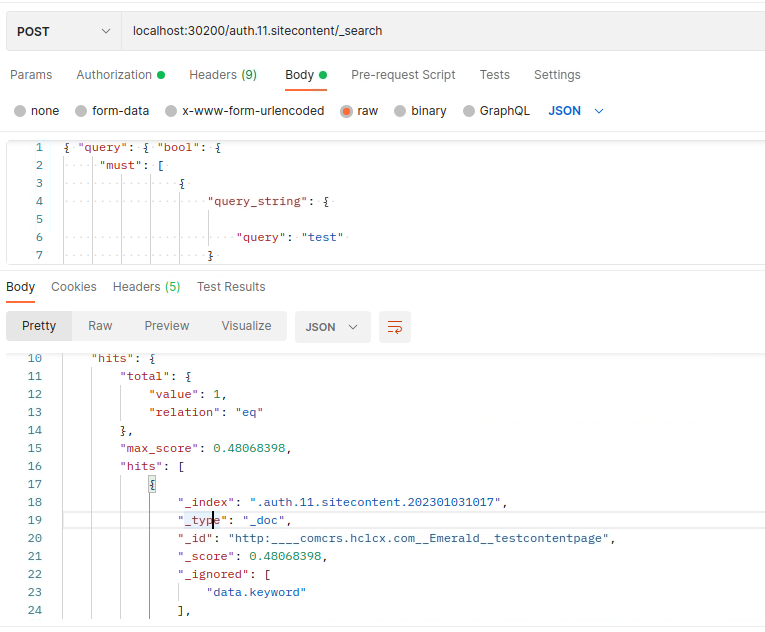
"query": "test"

}

}

]

} } }



**Step 12:**  Now, we will create a new search API to access the unstructured index data

1. Create a new rest resource “UnstructuredContentResource”.



1. Create a new resource property file “**unstructured-resources.properties**”



1. Create the json file for the profile “**HCL\_findSiteContentsBySearchTerm**” under “**profiles.unstructured**”



1. Create a new custom postprocessor “**SearchWebContentPostprocessor**”



1. Select all these files and export them as jar.
2. Patch the jar in **search-query-app** and **search-query-appdata** containers

Copy the jar to the directory **/patches/** in both the containers.

Go to **/SETUP/bin/**

Run **./patch.sh ../../patches/{jarname}**

1. Create a new field mapping “**UnstructuredContentResponseFieldMapping**”. Pass the below json to the below api.

**POST - {{baseUrl}}/api/v2/configuration?nodeName=component&envType=auth**



1. Restart both **search-query-app** and **search-query-appdata** containers.
2. Hit the new custom API to validate the data in response. This API has to be integrated in frontend.

**http://localhost:3737/search/resources/store/1/unstructuredcontent/siteContentsBySearchTerm/{searchTerm}**

