Operational Cache Technical Preview Documentation

This document will guide you how to use Operational Cache (OC) Technical Preview.
The Operational Cache Technical Preview setup is a stack of docker containers consisting of:

- 3 x SOLR Cloud
- 3 x Zookeeper
- 1 x Cassandra
- 1 x Derby Database
- 1 x Operational Cache API

The underlying storage backend for Janus Graph is Cassandra. SOLR is being leveraged as an indexing back end to allow speedy searches on record data (represented as vertices in the graph).

Pre-Installation Requirements

Operating System: CentOS 7.1+/Ubuntu 18.x OR RHEL 7.1+

Docker: Docker CE 18.06+ for CentOS/Ubuntu or Docker EE 17.06+ for Redhat.

Docker-Compose: Docker Compose 1.23.x or higher.

CPU: 8 CPU or more

Memory: 16GB or more.

Disk Space: 100 GB or more.

NOTE: The amount of CPU, Memory and disk space required will be dependent on how much data you plan to load and test with. Depending on the docker storage driver you use

you may need to increase the docker image size settings prior to starting up the container.

Installation

The `operational-cache-installer.bin` file will be used to download:
- The Operational Cache Docker Image
- A compose script that can be used to provision the stack
- A sample performance testing script
- **NOTE**: This script must be run on the node hosting the docker containers. It also requires access to run docker commands.

The system in which the `bin` file is run on must have internet access.

To run the bin make sure to give the file executable permissions.
`chmod 775 operational-cache-installer.bin`

The `bin` is a self extracting bundle that will execute a script that prompts you for information (including licensing) and guides you through the installation process.

Docker Compose & Docker Swarm

Docker compose and swarm is used to stand up the stack of containers necessary for OC to function. Configuration parameters have been tuned based upon the compose configuration.

NOTE: We leverage the network overlay settings provided by Docker Swarm but the Technical Preview does NOT support provisioning of a stack on a multi node Docker Swarm setup via the `docker stack deploy` command. This is due to certain containers having the requirement to be run in privileged mode which is currently not supported with `docker stack deploy`.

Initialize Docker Swarm

Before we can stand up the OC environment using compose we need to initialize docker swarm, to do so issue the following command:

```
docker swarm init
```

NOTE: Be sure to view the output of the above command to ensure it completed successfully. If your host has multiple NICs then it will not know which to bind to and you will need to rerun the command with an additional flag to provide that information.

Modifying values of the compose configuration

When using the technical preview with a small data set and minimum hardware (8 CPU, 16 GB) the compose configuration will work without modification, however on large servers the configuration needs to be tuned to avoid over/under allocation of memory/cpu. For example this issue was observed on our 32 CPU, 128GB MEM, 1tb SSD environments. The issue will result in data load/processing failures when too much memory is allocated and there isn't enough to go around. In order to resolve these issues it's recommended to add the following extra configuration to the `oc` service in the `docker-compose.yaml` file:

```
cpu_count: 8
mem_limit: 4096M
```

These settings will set the maximum amount of cpu and memory which can be used for each instance of OC. The above values may need to be tweaked to match the hardware you're running on.

Starting the stack using compose

To start the stack in the foreground and get a running log from the output of each container issue the following command:
NOTE: To start the containers in the background without keeping the terminal open include the `-d` flag.

```
docker-compose -f <path_to_compose_file> up --scale oc=<N>
```

Be sure to replace `<path_to_compose_file>` which should have been reported in the output of the bin file. Also be sure to replace `<N>` with the number of instances you would like for the OC container.

NOTE: For the Technical Preview, only 1 Operational Cache API container is supported.

Stopping the stack using compose

If you started the stack in the foreground you can use key combination `ctrl-c` once to stop the stack gracefully. Using `ctrl-c` a second time will force kill the stack. If you started the stack in the background you will need to use the `docker stop <container>` command to stop each container.

Using Operational Cache

Exposed Web Applications

Operational Cache (OC) REST API

The OC REST API is available at `https://<DOCKER_HOST_NAME>:<PORT>/api` and the Swagger Reference Doc is available at `https://<DOCKER_HOST_NAME>:<PORT>/api/explorer`
The default credentials are:

```
username: test
password: test
```

Operational Cache admin credentials (used for advanced tasks like using Gremlin Templates):

```
username: admin
password: admin
```

The `<PORT>` value will differ depending on how many instances of OC REST API you have provisioned and the port assigned/mapped to 9443. This can be checked by `docker ps` with a sample output like:

```
docker ps
CONTAINER ID        IMAGE
          COMMAND                  CREATED             STATUS
      PORTS
      NAMES
9a4a69d0993f        registry.ng.bluemix.net/mdmexpress/operational-cache:1
.0.0.0   "/usr/sbin/init"         42 minutes ago      Up 3 minutes
      0.0.0.0:32769->9443/tcp
```

This indicates that the `<PORT>` value is 32769

SOLR Consoles

The SOLR Console UIs are available at:

- `https://<DOCKER_HOST_NAME>:8983/solr/`
- `https://<DOCKER_HOST_NAME>:8984/solr/`
- `https://<DOCKER_HOST_NAME>:8985/solr/`

Accessing the Docker Container and Starting/Stopping the Docker Container.

Accessing the Container:

In the host machine, type `docker exec -it <container_name> bash` to access the container. The `<container_name>` will vary depending on the instance you wish to access.

Stopping individual containers:

In the host machine, type:
- `docker ps -aq` to get the ID of the running container
- `docker container stop -t 600 <containerID>` to stop the container.

Starting individual containers:

In the host machine, type:
- `docker ps -aq` to get the ID of the stopped container
- `docker container start <containerID>` to start the container.

NOTE: Starting time of containers varies. The OC container can take a few minutes to start.

Significant Directories (inside the container)

Root Directories

- `/usr/ibmpacks/current/bigmatch` : Contains all the product binaries
  - This directory also includes the OC specific binaries

Configuration Directories

- `/usr/ibmpacks/current/bigmatch/conf` : EI configuration files

Logging Directories

- `/var/log/bigmatch` : Log files of the IBM WebSphere Liberty Instance (all REST API logging would be in this log)

Other Directories

- `/usr/ibmpacks/current/bigmatch/bin` : Contains scripts useful for managing OC

Control Scripts

The OC API container uses systemd to manage running processes. If inside the container, Use `systemctl` to start, stop, or get status of the service.

The service name is:
- `oc.service` : Will manage the Operational Cache service

For example, to restart Operational Cache from inside the container you would run the following commands:

```
sudo systemctl stop oc.service
sudo systemctl start oc.service
```

Troubleshooting

SystemD service failures

If the platform or entity-insight services fail you can use the `journalctl` utility to get information about what was run at boot time. For example if the `platform.service` failed I would want to run `journalctl -u <service-name>.service -b` to determine what might of went wrong with the loading of that service during the last boot up sequence.

It is highly recommended that if inside the container and not operating with docker stop/start commands then restarting all services is done through the `systemctl` commands. If IBM WebSphere Liberty or any other component is restarted outside of the `systemctl` commands, services may not function as expected. To resolve this issue, stop all services that were restarted externally, and then restart all services using the recommended commands.

Loading Sample Data

You can use the IBM MDM Publisher to Bulk load MDM AE data into the Operational Cache but if needed, also included in the OC API container is a shell script that loads sample MDM data including Person and Contract data, to the application. This process extends the logical model to include support for consent and contract data, before running several processes that load the records and relationships into the application.
The data loading process may take about an hour or more depending on resources allocated as approximately 180k records are loaded along with their relationships.

To load the sample data, execute the following commands:

```
sudo su - bigmatch
/usr/ibmpacks/current/bigmatch/bin/loadMDMSampleDataWithContract.sh
```

Searching using APIs

Searches can be performed using the OC REST APIs. The HTTP function used for running searches is as follows:

POST /api/search

```
{
  "searchType": "RECORD",
  "query": {
    "expressions": [
      {
        "property": "recordSource",
        "value": "MDM"
      }
    ]
  },
  "filters": [
    {
      "type": "RECORD",
      "values": [
        "Contract"
      ]
    }
  ]
}
```

This search includes a request for records with Record Source as "MDM", while a record type filter is applied to the search. The result is a list of records of type Contract.

```
{
  "searchType": "RECORD",

  "query":
  {
    "operation": "AND",
    "expressions":
    [
      {
        "property": "LegalName.LastName",
```

```
        "value": "MORALES"
      },
      {
        "property": "LegalName.GivenName",
        "value": "TRINA"
      }
    ]
  }
}
```

This search includes a request for records which have a `LegalName.LastName` of `MORALES` and `LegalName.GivenName` of `TRINA`.

Domain Object Transformation

This feature allows for the ability to do the following:
- Searching using custom properties assuming the transformation code includes customizations to map the custom properties to attributes in the application model.
- Searching using a custom content type other than JSON such as (XML, YAML, etc).
- Modifying the results of a search such as returning custom content type other than JSON (XML, YAML, etc) and returning custom properties, assuming the transformation code includes customizations to map the custom properties to attributes in the application model.

Examples for both of these use cases are included in the installation.
Please see the `Sample Transforms` section below for more information on where to find them.

Below is an example of a custom search request with custom properties 'customFirstName' and 'customAddress1':

```
{
   "searchType":"RECORD",
   "query":{
      "expressions":[
         {
            "property":"customFirstName",
            "condition":"EQUAL",
```

```
            "value":"John"
         },
         {
            "property":"customAddress1",
            "condition":"EQUAL",
            "value":"101 Main Street"
         }
      ],
      "operation":"AND"
   }
}
```

The following is an example of a transformed search request from the above example showing the custom properties changing to 'LegalName.GivenName' and 'PrimaryResidence.Address.AddressLine1' respectively:

```
{
   "searchType":"RECORD",
   "query":{
      "expressions":[
         {
            "property":"LegalName.GivenName",
            "condition":"EQUAL",
            "value":"John"
         },
         {
            "property":"PrimaryResidence.Address.AddressLine1",
            "condition":"EQUAL",
            "value":"101 Main Street"
         }
      ],
      "operation":"AND"
   }
}
```

The following API endpoint functions were enhanced to support Domain Object Transformation:

POST /api/search

POST /api/search/graph/query

Creating custom transforms

Creating a custom transform gives the client the ability the transform custom search requests into content that is accepted by the application. It also allows the application to transform the response back into the custom format accepted by the client.

In order to create a custom transform, the custom transform class must be annotated with the `@CustomXFormer` annotation and implement the following interface:

`com.ibm.entity.analytics.extension.transform.Transform`

This interface is located in the following jar file:
`/usr/ibmpacks/current/bigmatch/analytics/lib/analytics-graph-extension-transform-<version>.jar`

This interface contains two methods in its contract which do the following:

- toSearchCriteria(String searchCriteria) – Transforms a custom search request string into a Criteria object which is accepted by the application.
- fromSearchResult(SearchResult searchResult) – Transforms a SearchResult object returned from the application into a custom search response.

The custom transform must be packaged in a jar file and must contain a beans.xml file inside its META-INF directory, and must be placed in the following directory:

`/usr/ibmpacks/current/bigmatch/analytics/lib`

After the transform has been placed in the specified directory, you must restart all services, as described earlier in this document, for the changes to take effect.

The application uses a default implementation which is bundled in the same jar file as the interface. However, if both the default and custom implementations are present, the application will use the custom implementation.

Sample Transforms

A jar containing sample transforms, including the source code is located in the following file:
`/usr/ibmpacks/current/bigmatch/etc/transform-sample/analytics-graph-extension-transform-custom-<version>.jar`

The following examples are included in the jar file:
- CustomJSONPropertyKeyTransformer.java – An example custom transformer that transforms custom search property keys into application model property keys when running

a search. The following custom properties are mapped to the application model properties respectively:

`customAddress1` : PrimaryResidence.Address.AddressLine1
`customAddress2` : PrimaryResidence.Address.AddressLine2
`customAddress3` : PrimaryResidence.Address.AddressLine3
`customAddressCity` : PrimaryResidence.Address.City
`customAddressState` : PrimaryResidence.Address.ProvinceStateValue
`customAddressZip` : PrimaryResidence.Address.ZipPostalCode
`customAddressCountry` : PrimaryResidence.Address.CountryValue
`customFirstName` : LegalName.GivenName
`customMiddleName` : LegalName.MiddleName
`customLastName` : LegalName.LastName

The result of a search is also transformed so that the model property keys are transformed back into the custom property keys.

- CustomXMLTransformer.java – An example custom transformer that transforms a custom XML search request into a Criteria object which is accepted by the application. The result of a search is also transformed from a SearchResult object back into the custom XML format.

To use the sample transforms, the jar containing the sample transforms must be placed in the `/usr/ibmpacks/current/bigmatch/analytics/lib` directory and all services must be restarted.

Due to the fact that the CustomXMLTransformer is annotated with the `@alternative` annotation, the CustomJSONPropertyKeyTransformer will be used by default. To use the CustomXMLTransformer, you must modify the `WEB-INF/beans.xml` file in the analytics-graph-api-rest.war by adding the following:

```
<beans>
    <alternatives>
        <class>com.ibm.entity.analytics.graph.extension.transform.CustomXM
LTransformer</class>
    </alternatives>
</beans>
```

Note: You must restart all services after modifying the `WEB-INF/beans.xml`

Additional Notes

Due to the fact that the application does not restrict the content type generated by the transformer, the `Accept` header of an incoming search request will be ignored.

Gremlin Query Searches

There is functionality to traverse the graph database to perform searches via traversal filters. The traversal language of choice in JanusGraph is Gremlin.

Defining Gremlin Templates

A user with the role of Entity Insight Administrator is able to create and delete templates for generic Gremlin queries, which can then be executed by any Entity Insight user. This feature also includes support for transforming the results of the graph traversal when the Domain Object Transformation feature is enabled.

The following API endpoint functions were introduced to enable an Entity Insight administrator to manage query templates:

POST /api/search/graph/template

The endpoint function enables the Entity Insight admin to create a new query template by specifying a name for the template, a brief description, the query to be executed, and the list of placeholders contained in the query. Templates are validated before creation, and pose the following restrictions:
- Placeholders may not contain any of the reserved words: g, vertex, element, edge, property, label, key.
- The query should contain equality comparisons alone. Range and wildcard queries will show poor performance due to the nature of the database schema.
Upon validation, the template is created and visible by any Entity Insight user.

An example of an input to the endpoint is as follows:

```
{
  "name": "ContractComponent Links by Source",
  "description": "This template can be used to determine which contract co
```

```
mponents are linked to contracts based on the specified edge source value.
",
  "query": "g.V().has('recordType', 'Contract').outE('ContractComponentLin
k').has('recordSource', src).inV().has('recordType', 'ContractComponent')"
,
  "placeholders": [
    "src"
  ]
}
```

The result of the above example would be the defined template with an identifier assigned:

```
{
  "id": <template_id>,
  "name": "ContractComponent Links by Source",
  "description": "This template can be used to determine which contract co
mponents are linked to contracts based on the specified edge source value.
",
  "query": "g.V().has('recordType', 'Contract').outE('ContractComponentLin
k').has('recordSource', src).inV().has('recordType', 'ContractComponent')"
,
  "placeholders": [
    "src"
  ]
}
```

DELETE /api/search/graph/template/{id}

This endpoint is used to delete a previously defined query template by specifying the numeric identifier of the template in the request.

Viewing Defined Templates

An Entity Insight user or administrator may view a list of defined templates, or a single template given its identifier, using the endpoint functions respectively:
- `GET /api/search/graph/template`
- `GET /api/search/graph/template/{id}`

Running Gremlin Searches

Entity Insight users have the ability to execute these queries, with parameter mappings specified for any placeholders in the template. The user may optionally request a specific

page of results using query parameters to the endpoint function, otherwise the default paging configurations for the application are used to retrieve a subset of the results.

The following example illustrates how an Entity Insight user might perform a gremlin query using a defined template:

`POST /api/graph/query`

```
{
  "template": {
    "id": <template>id>
  },
  "parameters": {
    "src": "MDM"
  }
}
```

The results is then returned or transformed depending on whether or not Domain Object Transformation is enabled.

Performance

Executing a gremlin traversal requires JanusGraph to iterate over a vertex stream and apply the specified filters, before returning the results as a stream. As a result, the performance of the search is non-deterministic. While case-by-case situations may show different results, the following performance observations apply in the general case:
- Requesting higher page numbers will show higher latency in search performance.
- Searches against a sparse result set may show higher latency since the process could spend more time between found results.
- Edge-centric queries will show poor performance. JanusGraph traversals are vertex-centric.
- Subsequent executions of the same query may show improved performance depending on the caching mechanism on the graph.

Recommendations to Improve Performance

- IMPORTANT: The property keys and edge labels being used in the query should be indexed in JanusGraph to reduce latency, especially on databases containing large data sets. Vertex-centric searches on indexed vertices show extremely low latency, and typically return within a few seconds even in the presence of large data sets. Searching on non-indexed properties can take minutes depending on the conditions of the search

and the nature of the data.

  - Information on how to create composite indexes can be found here: https://docs.janusgraph.org/latest/indexes.html. See Section 11.1.1: Composite Index. The indexes should be created before data is loaded to avoid the need for reindexing the graph, which is an expensive process.
  - If queries rely on specific edge labels, those labels should be indexed using vertex-centric indexes. See the JanusGraph documentation https://docs.janusgraph.org/latest/indexes.html at Section 11.2: Vertex-centric Indexes.
- JanusGraph documentation recommends the use of the configuration property `query.force-index` to require the use of indexes in searches. This prevents execution of queries that do not utilize indexes in JanusGraph, and thus could potentially take unreasonable amounts of time. More information can be found in the official JanusGraph documentation.