IBM MDM Bulk Publisher for Operational Cache Technical Preview Documentation

This document will guide you how to use the MDM Bulk Publisher for Operational Cache (OC) Technical Preview. The MDM Bulk Publisher is comprised of a single self container docker container with all the backend services and front end user interfaces.

NOTE: Please follow this guide after following Operational Cache installation guide, as Publisher setup will require information about OC setup.

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: 50 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.

Docker Compose & Docker Swarm

Docker compose and swarm is used to stand up the container necessary for Publisher to function. Configuration parameters have been hard coded 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.

Setup

Note that it is recommended to setup the MDM Publisher and Operational Cache on different physical hosts. If you decide to install Publisher on the same machine as OC, make sure that there is sufficient memory available for both based on requirements listed in the OC document as well as in the Publisher document.

The `mdm-publisher.bin` file will be used to download MDM Publisher image.
Note: 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. 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.

NOTE: This script must be run on the node hosting the docker containers. It also requires access to run docker commands. Lastly, machine hosting OC containers should be live and reachable from the machine where MDM Publisher is being installed.

Note that there are two more configuration scripts under `mdm-publisher` that are not run automatically and should be run using the information below.
Please make sure to use `chmod +x` to make them executable prior to running them.

- `configure_oc_connection.sh` - configures Publisher compose file to point to the Operational Cache instance. Please make sure to have OC instance information handy in order to complete the prompts of this script. This script will have to be executed every time OC is deployed on a different instance.
- `configure_certificates.sh` - configures certificate on Publisher WLP instance to ensure secure communication to OC. Note that during this script you will be prompted for OC information as well. This script starts up Publisher compose, and expects that OC instance you're connecting to is also live and functional. The requirement of running and reachable OC is there as the script will extract OC WLP certificate and import it into Publisher as trusted. Once this script is complete, please allow Publisher container to fully initialize, delaying any action by approximately 10 minutes. Note that you will need to run this script every time OC instance is updated with a new certificate, as the certificate trust between instances will need to be re-established.

Once the above actions have been completed, Publisher instance is operational and ready to publish data into OC.

NOTE The default performance configuration of Publisher may not be suitable for large datasets and may have to be tweaked in accordance with the Performance guidelines below.

Modifying values of the compose configuration

If Publisher does not exclusively occupy the machine, you may want to customize the service definition inside of the compose file to limit Publisher to a certain memory and cpu allocation, avoiding starving other processes:

```
cpu_count: 8
mem_limit: 16384M
```

However it is recommended to deploy Publisher to a machine that is exclusive to it.

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

on.

Starting the stack using compose

You can stop and start Publisher container using the following compose commands.

```
# to start
docker-compose start
```

```
# to stop
docker-compose stop
```

You can also tear down the publisher container completely.

```
# to tear down publisher
docker-compose down
```

```
# to set it up from scratch
docker-compose up
```

NOTE: If you tear down the Publisher and recreate it from scratch - make sure to rerun `configure_oc_connection.sh` as previous WLP certificate config will be wiped out. If that is not done, you will experience SSL certificate errors, preventing Publisher from functioning correctly.

Configuring Oracle connectivity (optional)

The MDM Publisher can connect to a MDM AE DB2 or Oracle database but in order to connect to a Oracle database you will need to provide the Oracle JDBC driver JAR `ojdbc8.jar` and copy it into following locations within the docker container.

1. Rename `ojdbc8.jar` to `publisher-jdbc-jar-ojdbc8.jar`:

   ```
   mv ./ojdbc8.jar ./publisher-jdbc-jar-ojdbc8.jar
   ```

2. Copy the JAR into Publisher library location inside the container:

```
docker cp ./publisher-jdbc-jar-ojdbc8.jar compose_publisher_1:/usr/ibmp
acks/bigmatch/curre t/publisher/lib/publisher-jdbc-jar-ojdbc8.jar
```

3. Copy the JAR into Sqoop library location inside the container:

```
docker cp publisher-jdbc-jar-ojdbc8.jar compose_publisher_1:/usr/hadoop
-platform/current/sqoop/lib/publisher-jdbc-jar-ojdbc8.jar
```

4. Copy the JAR into Sqoop library HDFS location:

```
docker exec -it compose_publisher_1 hadoop fs -put /usr/hadoop-platform
/current/sqoop/lib/publisher-jdbc-jar-ojdbc8.jar /bigmatch/oozie/shared
Libraries/publisher/
```

5. Restart compose:

```
docker-compose -f <docker-compose.yml location> stop
```

```
docker-compose -f <docker-compose.yml location> start
```

After completing the above steps you should be able to connect to Oracle database through Publisher UI and execute export operations.

NOTE: If you tear down the Publisher container and recreate it from scratch, you will need to repeat the above steps to reenable Oracle export capabilities.

Using MDM Publisher

The MDM Publisher Docker Image Technical Preview has the same functionality as the IBM

MDM Publisher available with Entity Insight so most of the documentation referred here https://www.ibm.com/support/knowledgecenter/en/SSWSR9_11.6.0/com.ibm.swg.im.mdmhs.publisher.ui.doc/topics/pubui_pubui.html also applies to this Technical Preview with a few exceptions like:
- This Technical Preview is geared towards publishing data to the Operational Cache
- Command line/advanced usage is not supported in the technical preview.

Once the Publisher container is up and running you should be able to access Publisher UI at `https://hostname:9443/publisher`. You will be prompted to log in and the default credentials are `mdmadmin:mdmadmin`.

Preparing a publish job

After logging in, click on `Go to Job Catalog` in the top right corner of the page. This will take you to the list of currently running jobs, which should be empty if this is your first time accessing the UI. You should be able to sort the jobs by selecting various columns in the table.

In order to create a new job, click on `New Job` button in the top right corner of the screen. You will be take to a Job wizard, which will walk you through the configuration process.

At the `Configure` step of the wizard, please provide the database credentials for the MDM database you're trying to export. Once the details are complete, click on `Connect` button to validate the connection. If the button has turned grey and indicates `Connected!`, navigate to the top right corner of the page and click `Next` to proceed. Note that you can also `Save` the data source for future use. If you do so, when creating a new Job, you will be presented with a list of saved datasource, and will only have to fill in the password field to proceed.

WARNING: Since Publisher is running inside of the Docker container, any entries to the `/etc/hosts` file that exist on the physical host will need to be applied to the container if those entries are required for OC or datasource connectivity. You can modify `/etc/hosts` file directly in the container, but that configuration will be reset at container restart. To create configuration that survives restart, modify `docker-compose.yml` file of Publisher and add desired mappings under `extra-hosts` object. That will translate into `/etc/hosts` file inside of the container.

In the next screen of the Job configuration wizard, you will be asked to select data subsets that will apply to the publish job. By selecting specific entities - you are including them into the job. To select an entity or a sub-entity, click on it and then click `Add all` depending on the context. If entity is selected, all of its sub-entities and their respective fields will be

added. If sub-entity is selected, all of its fields will be added. Note that you can view details of entities and sub-entities by simply clicking on them, the corresponding details will show up on the right page, describing what fields are included.

Once you complete your selection of data subsets, click `Next` to proceed.

In the following screen you will be presented with the review of all of the data subsets you have selected as well as total counts. Please verify your selection. If a mistake is spotted, click `Previous` in the top right corner of the screen to go back to the dataset selection screen. Note that at any point during the use of the configuration wizard you can click `Cancel` in the top right corner of the page to discard any configuration done so far.

Once the review is completed, click `Next` in the top right corner of the screen to proceed.

The last screen of the configuration of the wizard is the summary page. Here you will be asked to name the publish job and presented with an overview. If you are ready to start the job, click `Publish` in the top right corner of the screen. The job will be submitted, and you will be take to the Job Catalog page, where you can observe the progress of the publish job and its details. You will also be able to pause a running job by clicking `Pause` button, and rerun a failed job by clicking `Rerun`.

Note that you can retrieve the details of the job by using Oozie, Hadoop or Spark history consoles as outlined in the Troubleshooting section.

Once the job is completed you can navigate over to Operational Cache APIs and verify that the data has been loaded successfully.

Troubleshooting

Error 500 in the Publisher UI

If you experience a `500` error while using Publisher UI, follow the steps below to look at the logs:

1. Get into the Publisher container by running `docker exec –it compose_publisher_1 bash`.
2. Once inside the container inspect WLP log for errors. It is located at `/var/log/bigmatch/`.

One of the common reasons you may see an error is the lack of certificate trust setup between Operational Cache and MDM Publisher. If you see a missing certificate error,

please run the `configure_oc_connection.sh` mentioned in the installation section to properly configure publisher.

Unexpected errors

To avoid any unexpected errors from Publisher, please do not start or stop any services manually from within the container. Instead use the `docker-compose` commands provided above.

Monitoring Publish jobs

Note that Publisher container exposes Oozie, Yarn resource Manager UI and Spark ports that should help you monitor ongoing jobs and identify potential issues. You should be able to reach them at the following ports:
- `11000` - Oozie
- `8088` - Yarn Resource Manager
- `18080` - Spark
- `19888` - Job History

Working with Larger Data Sets

Location of configuration files

The following files are used to change configuration based on the hardware allocated and the size/complexity of the data set:

```
graphload_edge-template.properties
graphload_vertex-template.properties
publish-config.xml
```

These files are located under `/usr/ibmpacks/current/bigmatch/conf/publisher` inside the container.

Note: To Get into the Publisher container, run command `docker exec -it compose_publisher_1 bash`.

Examples

Following are some of the examples we have tested internally along with the details of the hardware used and the configuration parameters.

10K Person Data Set ----------

Machine hardware:

- VCPU count: 8
- Memory: 16GB
- Disk space: 1TB

Publisher configuration used:

This is the default configuration that ships with the Publisher docker image.

graphload_edge-template.properties:

```
graphBatchCommitSize=100
```

graphload_vertex-template.properties:

```
graphBatchCommitSize=100
```

publish-config.xml:

```
<property>
    <name>publisher.sqoop-import.args.m</name>
    <value>4</value>
</property>
<property>
    <name>publisher.subflow.prop.executorMemory</name>
    <value>1G</value>
</property>
<property>
    <name>publisher.subflow.prop.executorCores</name>
    <value>1</value>
</property>
```

```
<property>
    <name>publisher.subflow.prop.numExecutors</name>
    <value>4</value>
</property>
<property>
    <name>publisher.subflow.prop.executorMemoryOverhead</name>
    <value>256</value>
</property>
<property>
    <name>publisher.subflow.prop.driverMemory</name>
    <value>512M</value>
</property>
</property>
<property>
    <name>publisher.spark-join.opts.executor-memory</name>
    <value>1G</value>
</property>
<property>
    <name>publisher.spark-join.opts.num-executors</name>
    <value>4</value>
</property>
<property>
    <name>publisher.spark-join.optconf.spark.sql.shuffle.partitions</name>
    <value>200</value>
</property>
```

Total time taken to load data: 58 Minutes

NOTE: A certain overhead exists for the services that are spun up and the processes that are started for each job. As the size of the data grows, the relative percentage of time spent on those processes will diminish.

1M Person ----------

Machine hardware:

- VCPU count: 8
- Memory: 16GB
- Disk space: 1TB

Publisher configuration:

graphload_edge-template.properties:

```
graphBatchCommitSize=100
```

graphload_vertex-template.properties:

```
graphBatchCommitSize=100
```

publish-config.xml:

```
<property>
    <name>publisher.sqoop-import.args.m</name>
    <value>4</value>
</property>
<property>
    <name>publisher.subflow.prop.executorMemory</name>
    <value>1G</value>
</property>
<property>
    <name>publisher.subflow.prop.executorCores</name>
    <value>1</value>
</property>
<property>
    <name>publisher.subflow.prop.numExecutors</name>
    <value>4</value>
</property>
<property>
    <name>publisher.subflow.prop.executorMemoryOverhead</name>
    <value>256</value>
</property>
<property>
    <name>publisher.subflow.prop.driverMemory</name>
    <value>512M</value>
</property>
</property>
<property>
    <name>publisher.spark-join.opts.executor-memory</name>
    <value>1500M</value>
</property>
<property>
    <name>publisher.spark-join.opts.num-executors</name>
    <value>4</value>
</property>
<property>
    <name>publisher.spark-join.optconf.spark.sql.shuffle.partitions</name>
```

```
    <value>200</value>
</property>
```

Total time taken to load data: 70 minutes

10M Person ----------

Machine resources:

- VCPU count: 32
- Memory: 128GB
- Disk space: 1TB

Publisher configuration:

graphload_edge-template.properties:

```
graphBatchCommitSize=100
```

graphload_vertex-template.properties:

```
graphBatchCommitSize=100
```

publish-config.xml:

```
<property>
    <name>publisher.sqoop-import.args.m</name>
    <value>30</value>
</property>
<property>
    <name>publisher.subflow.prop.executorMemory</name>
    <value>1G</value>
</property>
<property>
    <name>publisher.subflow.prop.executorCores</name>
    <value>1</value>
</property>
<property>
    <name>publisher.subflow.prop.numExecutors</name>
    <value>30</value>
```

```
</property>
<property>
    <name>publisher.subflow.prop.executorMemoryOverhead</name>
    <value>256</value>
</property>
<property>
    <name>publisher.subflow.prop.driverMemory</name>
    <value>512M</value>
</property>
</property>
<property>
    <name>publisher.spark-join.opts.executor-memory</name>
    <value>1500M</value>
</property>
<property>
    <name>publisher.spark-join.opts.num-executors</name>
    <value>30</value>
</property>
<property>
    <name>publisher.spark-join.optconf.spark.sql.shuffle.partitions</name>
    <value>200</value>
</property>
```

Total time taken to load data: 105 minutes

General Performance considerations:

- The overall time taken to load a data set is dependent on a combination of the hardware being used, the configuration properties and the characteristics of a data set. (e.g. loading a large number of relationships will take longer as compared to a person only data set)
- When using a larger dataset, you may have to adjust `publisher.spark-join.optconf.spark.sql.shuffle.partitions` to a number greater than `200`. Alternatively you can increase the amount memory per spark executor `publisher.spark-join.opts.executor-memory` to be able to handle fewer partitions that are greater in size. The ideal combination of these two parameters will vary depending on your dataset size and hardware.