Deploying the DuraCloud Mill

The following instructions describe how to run the Mill for development purposes. If you are deploying the Mill in a production environment, see the documentation here: https://github.com/duracloud/deployment-docs/blob/master/mill-setup.md.

This article describes the necessary steps for configuring and running the DuraCloud Mill. The DuraCloud Mill is a collection of applications that work in concert with DuraCloud to perform a variety of bulk processing tasks including audit log writing, manifest maintenance, duplication and bit integrity checks. While the Mill has been designed to run in an auto-scalable environment such as AWS EC2, it can also be run as a suite of stand-alone applications on any machine with sufficient computing resources. This article only describes the various applications and how to configure them; it does not cover approaches to leveraging AWS autoscaling and supporting dev opts tools such as Puppet.

If you are not yet familiar with the DuraCloud Mill please refer to the DuraCloud Architecture document, which describes the purpose and primary components of the Mill.

Requirements

1. Java: Version 8 required
    a. The Oracle JDK is recommended for building DuraCloud, as this is the JDK used for DuraCloud testing.
2. Maven 3.x
3. Git
4. MySQL database
5. AWS account and credentials

Download, Build, Configure

1. To start, clone and build the latest release of the mill

```
git clone https://github.com/duracloud/mill.git
cd mill
git checkout release-2.1.0
mvn clean install
```

2. Create database
    a. Create the empty mill database
    b. Add database credentials
    c. Create the schema using <mill project root>/resources/mill.schema.sql
    d. Run schema updates in ascending order: <mill project root>/resources/schema-update-*.sql
3. Create a configuration file.
    a. Now that you've built the mill and created the database, we need to set up a configuration file that can be used my the various components of the system. A template of this configuration file can be found in the base line at mill/resources/mill-config-sample.properties
        i. Copy and rename the file to mill-config.properties
        ii. Configure the database connections to the mill database as well as the management console database:

```
###################
# Mill Database
###################
# Config for mill database.
mill.db.host=[fill in]
mill.db.port=[fill in]
mill.db.name=[fill in]
# User must have read/write permission
mill.db.user=[fill in]
mill.db.pass=[fill in]

###################
# Account Management Database
###################
# Config for the management console database - used to retrieve accounts and storage
provider credentials
db.host=[fill in]
db.port=[fill in]
db.name=[fill in]
# User must have read permission
db.user=[fill in]
db.pass=[fill in]
```

Configuring and Running Workman

Workman is responsible for reading tasks from a set of queues, delegating them to task processors, and removing them once they have reached a completed state. In the case of failures, tasks are retried three times before they are sent to the dead letter queue. A single instance of Workman can run multiple tasks in parallel. How many tasks depends on the **max-workers** setting in the mill-config.properties file. It is also safe to run multiple instances of workman a single machine as well as multiple. We recommend running a single instance of workman on each machine instance, setting the max-workers setting in accordance with the available resources.

1. Queue Names refer to the AWS SQS queue names defined in your account. You must create and configure the following queues as defined in the queue section by replacing the brackets ([]) with names.

```
#########
# Queues
#########
queue.name.audit=[]
queue.name.bit-integrity=[]
queue.name.dup-high-priority=[]
queue.name.dup-low-priority=[]
queue.name.bit-error=[]
queue.name.bit-report=[]
queue.name.storagestats=[]
queue.name.dead-letter=[]
```

2. For a given instance of workman you must specify which queues will be consumed and the order in which they will be read. In other words, a given instance of workman can focus on specific kinds of tasks. It can also decide which tasks have a higher priority. In this way, instances of workman can be configured to work on hardware configurations that are suitable to the load and kinds of tasks they will need to bear. Make sure you use the above defined keys rather than the queue names themselves.

```
## A comma-separated prioritized list of task queue keys (ie do not use the
## concrete aws queue names - use  queue.name.* keys) where the first is highest priority.
## The first items in the list have highest priority; the last the lowest.
queue.task.ordered=[]
```

3. As we mentioned before, max-workers sets the number of task processing threads that can run simultaneously.

```
# The max number of worker threads that can run at a time. The default value is 5.
max-workers=[]
```

4. The duplication policy manager writes policies to an S3 bucket. Both the loopingduptaskproducer and workman use those policies for making decisions about duplication.

```
# The last portion of the name of the S3 bucket where duplication policies can be found.
duplication-policy.bucket-suffix=duplication-policy-repo
# The frequency in milliseconds between refreshes of duplication policies.
duplication-policy.refresh-frequency=[]
```

5. You can also set the workdir which defines where temp data will be written as well as notification.recipients.

```
# Directory that will be used to temporarily store files as they are being processed.
workdir=[]
# A comma-separated list of email addresses
notification.recipients=[]
```

6. Once these settings are in place you can run workman by simply invoking the following java command:

```
 java -Dlog.level=INFO -jar workman-{mill version here}.jar -c /path/to/mill-config.properties
```

Configuring and Running Duplication

Once you have an instance of workman running you can perform an explicit duplication run. The spaces that have been configured with duplication policies (see the Mill Overview for details) will generate duplication events when the audit tasks associated with them are processed. If you add a new duplication policy to a new space that already has content items, you'll need to perform a duplication run to ensure that those new items get duplicated. The loopingduptaskproducer fulfills this function. Based on the set of duplication policies, it will generate duplication tasks for all matching spaces. It will keep track of which accounts, spaces and items have been processed in a given run so it does not need to run in daemon mode. It will run until it has reached the max number of allowable items on the queue and then it will exit. The next time it is run, it will pick up where it left off. You may want to dial down the max queue size in the event that you have so many items and so little computing power to process them with that you may exceed the maximum life of an SQS message (which happens to be 14 days). It should also be noted here that items are added roughly one thousand at a time for each space in a round-robin fashion to ensure that all spaces are processed in a timely way.   This strategy ensures that small spaces that are flanked by large spaces are processed quickly.   **It is also important that only one instance of loopingduptaskproducer is running at any moment in time.**   Two settings to be concerned with when it comes to the looping dup task producer:

```
#############################
# LOOPING DUP TASK PRODUCER
#############################
# The frequency for a complete run through all store policies. Specify in hours (e.g. 3h), days (e.g. 3d), or
months (e.g. 3m). Default is 1m - i.e. one month
looping.dup.frequency=[]
# Indicates how large the task queue should be allowed to grow before the Looping Task Producer exits.
looping.dup.max-task-queue-size=[]
```

The program can be run using hte following command:

```
java -Dlog.level=INFO -jar loopingduptaskproducer-{mill version here}.jar -c /path/to/mill-config.properties
```

Configuring and Running Bit Integrity

Bit integrity runs works similarly to the duplication runs. **loopingbittaskproducer** has similar settings as those mentioned above as well as two others, looping.bit.inclusion-list-file and looping.bit.exclusion-list-file. These two config files let you be more surgical in what you decide to include and exclude from your bit integrity run and function similarly to the duplication policies. The important thing to note here is that if there are not entries in the inclusion list all accounts, stores, and spaces are included.   **It is also important that only one instance of loopingbittaskproducer is running at any moment in time.**

```
#############################
# LOOPING BIT TASK PRODUCER
#############################
# The frequency for a complete run through all store policies. Specify in hours (e.g. 3h), days (e.g. 3d), or
months (e.g. 3m). Default is 1m - i.e. one month
looping.bit.frequency=[]
# Indicates how large the task queue should be allowed to grow before the Looping Task Producer quits.
looping.bit.max-task-queue-size=[]
# A file containing inclusions. Expressions will be matched against the following path: /{account}/{storeId}/
{spaceId} and should have the same format. You can use an asterix (*) to indicate all.
# For example,  to indicate all spaces name "space-a"  in the "test" account across all providers you would add
a line like this:
#    /test/*/space-a
# You may also comment lines by using a hash (#) at the beginning of the line.
looping.bit.inclusion-list-file=[]
# A file containing exclusions as regular expressions using the same format as specified for inclusions.
looping.bit.exclusion-list-file=[]
```

```
java -Dlog.level=INFO -jar loopingbittaskproducer-{mill version here}.jar -c /path/to/mill-config.properties
```

Configuring and Running Storage Statistics

Storage statistics looping task producer runs works similarly to the duplication runs. **looping-storagestats-taskproducer** has similar settings as those mentioned above as well as two others, looping.storagestats.inclusion-list-file and looping.storagestats.exclusion-list-file. These two config files let you be more surgical in what you decide to include and exclude from your storage stats run and functions similarly to the duplication policies. The important thing to note here is that if there are no entries in the inclusion list then all accounts, stores, and spaces are included.

```
#############################
# LOOPING STORAGE STATS TASK PRODUCER
#############################
# The frequency for a complete run through all store policies. Specify in hours (e.g. 3h), days (e.g. 3d), or
months (e.g. 3m). Default is 1m - i.e. one month
looping.storagestats.frequency=1d
# Indicates how large the task queue should be allowed to grow before the Looping Task Producer quits.
looping.storagestats.max-task-queue-size=[]
# A file containing inclusions. Expressions will be matched against the following path: /{account}/{storeId}/
{spaceId} and should have the same format. You can use an asterix (*) to indicate all.
# For example,  to indicate all spaces name "space-a"  in the "test" account across all providers you would add
a line like this:
#    /test/*/space-a
# You may also comment lines by using a hash (#) at the beginning of the line.
looping.storagestats.inclusion-list-file=[]
# A file containing exclusions as regular expressions using the same format as specified for inclusions.
looping.storagestats.exclusion-list-file=[]
```

The program can be run using the following command:

```
 java -Dlog.level=INFO -jar looping-storagestats-taskproducer-{mill version here}.jar -c /path/to/mill-config.
properties
```

Configuring the Audit Log Generator

Audit logs are initially written to the Mill database, then transitioned to a single AWS bucket for safekeeping. DuraStore (on the DuraCloud side) needs to know about this bucket in order to present these logs to API and DurAdmin users. The auditloggenerator application is responsible for reading from the database and writing these log files. The audit_log_item table in the database holds the audit events until the auditloggenerator writes and securely stores theses files in S3. Once written, the auditloggenerator will flag audit events as written as well as delete anything over thirty days old. Once the audit log generator has written and purged everything there is to be written and purged, it will exit. It is recommended to run this process on a cron job to ensure that audit logs are processed in a timely way. **It is also important that only one instance of auditloggenerator is running at any moment in time.**

```
 # The global repository for duracloud audit logs
audit-log-generator.audit-log-space-id=[]
```

The program can be run using the following command:

```
 java -Dlog.level=INFO -jar auditloggenerator-{mill version here}.jar -c /path/to/mill-config.properties
```

Configuring the Manifest Cleaner

Finally the manifest-cleaner program is responsible for clearing out deleted manifest items from the database.   When items are deleted from the manifest, initially they are flagged as deleted rather than being deleted from the database.   We do it this way to ensure that we can correctly handle add and delete messages that are delivered out of order (since SQS does not guarantee FIFO). However, eventually we do want to purge deleted records to keep our database from growing too large.   So the manifest cleaner simply checks for "expired" deleted items and flushes them out of the db.   It has just one setting (in a addition to the database configuration information we mentioned above). We don't recommend specifying any time less than 5 minutes.

```
 ###################
# MANIFEST CLEANER
###################
# Time in seconds, minutes, hours, or days after which items deleted items should be purged.
# Expected format: [number: 0-n][timeunit:s,m,h,d]. For example 2 hours would be represented as 2h
manifest.expiration-time=[]
```

The program can be run using hte following command:

```
java -Dlog.level=INFO -jar manifest-cleaner-{mill version here}.jar -c /path/to/mill-config.properties
```