Dovecot Migration Framework Scheduler Technical Documentation for
1.0.0

2021-05-06

Copyright notice

Contents

1 General Information

1.1 Warnings

⚠️ **Warning**
This preview delivery is not for productive usage and not affected by service-level agreements.

⚠️ **Warning**
Images included in following pages have been attached as a generic visual reference for the theme and should not be considered as the final aspect when installed on production environment. Actual aspect will change based on components/plugins enabled and their configuration.

⚠️ **Warning**
Custom configuration or template files are potentially not updated automatically. After the update, please always check for files with a **.dpkg-new** or **.rpmnew** suffix and merge the changes manually. Configuration file changes are listed in their own respective section below but don't include changes to template files. For details about all the configuration files and templates shipped as part of this delivery, please read the relevant section of each package.

1.2 Delivery Comment

This delivery was requested with following comment:

> *DMF Scheduler 1.0.0 Feature Delivery*

1.3 Install Package Repository

This delivery is part of a restricted preview software repository:

```
https://software.open-xchange.com/components/dmf-scheduler/preview/1.0.0/RHEL7
https://software.open-xchange.com/components/dmf-scheduler/preview/1.0.0/DebianStretch
https://software.open-xchange.com/components/dmf-scheduler/preview/1.0.0/DebianBuster
```

1.4 Build Dependencies

This delivery was build with following dependencies:

```
RedHat:RHEL-7,Debian:Stretch,Debian:Buster
```

2 Overview

The Dovecot Migration Framework (DMF for short and henceforth referred to as) consists of multiple components that are installed and operate in various parts of the overall architecture of a migration from one Source IMAP installation to a Target Dovecot installation.

2.1 Migration Phases

While DMF leaves the operations of the migration phases up to the operator, it still defines two phases.

2.1.1 Pre-Sync

The purpose of the "pre-sync" (aka sync) phase of the migration is to copy over as much data as possible without locking users out of the Source system in order to minimize the time it will require to perform the "cutover" phase.

2.1.2 Cutover

The purpose of the "cutover" phase of the migration is to copy all data from the Source system to the Target while the user is locked out of the Source. It is important to note that DMF does not explicitly lock users on the Source, but can be configured to do so by defining pre/post/failure sync commands.

In summary, within DMF, there is not a strict difference between pre-sync and cutover. However, it provides the ability to separate the different phases to allow for executing specific operations during each phase.

2.2 Migration Components

2.2.1 The Source

The Source system is an IMAP installation that is running productively, either within the same data center or remotely. The Source IMAP server does not have to be Dovecot.

2.2.2 The Target

The Target system is a Dovecot installation that run the production environment in which the users from the Source are migrated into.

2.2.3 The Database

The Migration Database holds all DMF migration component data, and user migration data.

2.2.4 The Scheduler

The Scheduler runs in the target environment and is a server that provides APIs to

- schedule migrations during which data is being synced from the Source to the Target (by workers).
- query information about ongoing and finished migrations.
- manage Workers.

There may be multiple instances of the Scheduler for failover purposes. It is a standalone Java process that does not require any particular framework, and available through the package `open-xchange-dmf-scheduler` or Docker images.

2.2.5 The Workers

The Workers run in the target environment on Dovecot backends. Each Dovecot backend can have one DMF Worker. It is a standalone Java process that does not require any particular framework, and available through the package `open-xchange-dmf-worker`. They take care of the actual migration by processing migration jobs found in the Migration Database.
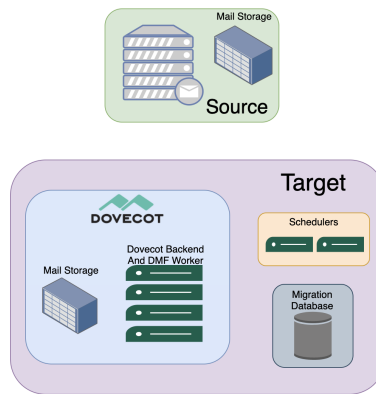
Figure 1: components

2.3   Communication

With a standard DMF deployment, there are at least three forms of communication:

- HTTPS between the DMF Client and DMF Scheduler
- JDBC between:
    - The DMF Scheduler and Migration Database
    - The DMF Worker and Migration Database
- IMAP between the Target Dovecot and Source IMAP Server

Some features of DMF may utilize additional forms of communication. For instance, using the doveadm protocol between the Target Dovecot and Source Dovecot server, or HTTP communication between the Target Dovecot node and Target Director node. Those kinds of communication information are specified in the documentation for each designated feature.
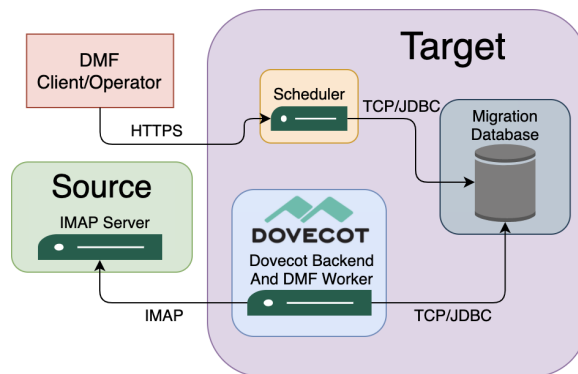
Figure 2: communication

2.4   Flow

2.4.1   Submitting the Migration

An operator submits a migration using the authenticated REST API directly, e.g.

```
1  curl -X 'POST' \
2    'https://dmf-scheduler:8443/dmf/api/v2/users/migrations' \
3    -H 'accept: application/json' \
4    -H 'X-API-KEY: b270b24a-711d-47ee-8bdc-f59ad5c1abf2' \
5    -H 'Content-Type: application/json' \
6    -d '{"sourceUid": "user1",
7        "targetUid": "user1",
```

6

```
 8          "phase": "PRE_SYNC",
 9          "priority": true,
10          "target": "default",
11          "sourcehost": "source-1",
12          "sourceport": 143
13        }'
```

Referencing the below UML:

- The Operator submits a new migration to the Scheduler using the REST API
- The Scheduler creates a new migration job based on the data submitted and stores it in the database "queue"
- Meanwhile, the Worker is polling the database for new jobs. It finds the newly created job
- The Worker processes the migration job
- The Worker updates the job's status in the database during and after the migration
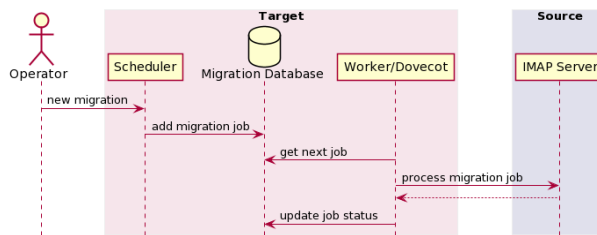
Figure 3: flow

3 Operations Guide

This guide provides the necessary information to deploy a DMF platform and run a migration. It does not provide the technical details of how DMF works, and you must read DMF Overview as a prerequisite to this.

3.1 Requirements

3.1.1 Database

The DMF Migration Database supports any recent version of MariaDB Server.

3.1.2 Java

The Worker and Scheduler require JRE 8.

3.2 Packages

3.2.1 1.0.0

Package	Component	Supported Distribution
open-xchange-dmf-scheduler	Scheduler	RHEL7, DebianBuster, DebianStretch
open-xchange-dmf-worker	Worker	RHEL7, DebianBuster, DebianStretch

3.3 Components

3.3.1 Migration Database

The Migration database is a MariaDB database used by both the Scheduler and Worker and requires read/write access to a database called `migration`.

For example:

```
1  CREATE USER 'scheduler'@'192.168.1.167' IDENTIFIED BY 'secret';
2  GRANT ALL PRIVILEGES ON migration.* TO 'scheduler'@'192.168.1.167' IDENTIFIED BY 'secret';
3  CREATE USER 'worker'@'192.168.1.167' IDENTIFIED BY 'secret';
4  GRANT ALL PRIVILEGES ON migration.* TO 'worker'@'192.168.1.167' IDENTIFIED BY 'secret';
```

By defaultBy default, the Scheduler is configured to create the database when it does not already exist. The Scheduler also manages the database using Liquibase.

3.3.2 DMF Scheduler

See the DMF Scheduler Install documentation.

3.3.3 DMF Worker

See the DMF Worker Install documentation.

3.4 Logging

Both the Scheduler and Worker use the Micronaut logging integration which is Logback.

The default configuration for the Scheduler is:

```
1  <configuration>
2      <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
3          <withJansi>true</withJansi>
4          <!-- encoders are assigned the type
5               ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
6          <encoder>
7              <pattern>%cyan(%d{HH:mm:ss.SSS}) %gray([%thread]) %highlight(%-5level) %
                   magenta(%logger{36}) - %msg%n</pattern>
8          </encoder>
9      </appender>
10     <root level="info">
11         <appender-ref ref="STDOUT" />
12     </root>
13 </configuration>
```

The default configuration for the Worker is:

```
1  <configuration>
2      <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
3          <withJansi>true</withJansi>
4          <!-- encoders are assigned the type
5               ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
6          <encoder>
7              <pattern>%cyan(%d{HH:mm:ss.SSS}) %gray([%thread]) %highlight(%-5level) %
                   magenta(%logger{36}) - %yellow(%mdc{job}) %msg%n</pattern>
8          </encoder>
9      </appender>
10     <root level="info">
11         <appender-ref ref="STDOUT"/>
12     </root>
13 </configuration>
```

Notice that there is Mapped Diagnostic Context (MDC) called "job" which will produce a value of `[JobID: <jobid>]` for all logging produced during a migration job.

All DMF loggers start with `dmf.worker`.

There are two ways that the log level can be customized:

3.4.1 Via Configuration

Micronaut documentation. This option will persist the log level as long as the property is set.

Add loggers with their desired log level under the "logger.levels" property.

Example:

application.yml

```
1  logger:
2    levels:
3      dmf: DEBUG
```

3.4.2 Via HTTP

Micronaut documentation. This option will persist only as long as the application is running.

Use GET to get the information about all loggers or a specific logger. Authorization is based on the configured basic auth.

```
1  curl -u admin:password https://scheduler:8443/loggers/dmf
```

Sample output:

```
1  {"configuredLevel":"NOT_SPECIFIED","effectiveLevel":"INFO"}
```

Finally, update the log level using POST:

```
1  curl -u admin:password -X POST -H "Content-Type: application/json" -d '{ "configuredLevel
      ": "DEBUG" }' https://scheduler:8443/loggers/dmf
```

3.5 Monitoring

3.5.1 Prometheus

The Scheduler and Worker both make use of the Micronaut Micrometer integration to export metrics for consumption by Prometheus.

See the individual component's Metrics sections for more details about what metrics each provide.

There are a lot of very useful metrics that can be gathered and shared with Grafana (or similar).

Prometheus can be configured like so:

prometheus.yml

```
1   global:
2     scrape_interval: 15s
3     evaluation_interval: 15s
4   rule_files:
5   scrape_configs:
6     - job_name: 'prometheus'
7       static_configs:
8       - targets: ['127.0.0.1:9090']
9     - job_name: 'scheduler'
10      metrics_path: '/prometheus'
11      scrape_interval: 5s
12      basic_auth:
13        username: admin
14        password: password
```

```
15      static_configs:
16      - targets: ['scheduler:8443']
17    - job_name: 'worker'
18      metrics_path: '/prometheus'
19      scrape_interval: 5s
20      basic_auth:
21        username: admin
22        password: password
23      static_configs:
24      - targets: ['worker:8443']
```

3.5.2 Grafana

It is up to you how you would like to utilize Grafana, however, future releases will provide additional metrics that will be helpful when monitoring progress of a migration.

3.6 Running a Migration

After installing all components, the following can be done to run a migration. See the DMF REST API section of the documentation for information on how to use the API.

3.6.1 1. Create a Source

- The `name` can be any identifier that represents the Source platform you are migrating from. It must match the regex `[a-zA-Z0-9_]+`.
- The `apiKey` will be used by you or your customer to submit and review migrations.

3.6.1.1 Example Source

```
1  curl -X 'POST' \
2    'https://scheduler:8443/dmf/admin/api/v2/sources' \
3    -H 'accept: application/json' \
4    -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
5    -H 'Content-Type: application/json' \
6    -d '{
7    "name": "CustomerX",
8    "apiKey": "b270b24a-711d-47ee-8bdc-f59ad5c1abf2"
9  }'
```

3.6.2 2. Create a Sourcehost

- The `sourceName` should be the name of the Source you just created.
- The Sourcehost `name` should be the host name of one of the IMAP servers that users of this Source will be migrated from.
- The `maxConnections` should be the maximum number of concurrent migrations that this host will support.
- For the following "Command" arguments, see the DMF Command Guide to learn how to construct them:
  - The `migrationCommand1` should be the migration command that will be executed during the pre-sync migration phase. If it is configured in the Worker, then simply specify a random command name (ex: `ignore`) and no other command components.
  - The `migrationCommand2` should be the migration command that will be executed during the cutover migration phase. If it is the same command that is executed for pre-sync, or it is configured in the Worker, then leave this empty.
  - The `statusCommand` should be the command that will be executed when collecting mailbox count and size from the user's source mailbox. If it will not be used, it is configured in the Worker, or the default should be used, then leave this empty.

3.6.2.1 Example Sourcehost

```
1   curl -X 'POST' \
2     'https://scheduler:8443/dmf/admin/api/v2/sources/CustomerX/sourcehosts' \
3     -H 'accept: application/json' \
4     -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
5     -H 'Content-Type: application/json' \
6     -d '{
7     "name": "imap.host.1",
8     "maxConnections": 200,
9     "migrationCommand1": {
10      "name": "doveadm",
11        "options": [
12          {
13            "name": "imapc_user",
14            "value": "%{mdb:ruid}"
15          },
16          {
17            "name": "imapc_password",
18            "value": "%{conf:imapc_master_password}"
19          },
20          {
21            "name": "imapc_host",
22            "value": "imap.host.1"
23          }
24        ],
25        "subCommand": {
26          "name": "backup",
27          "flags": [
28            "-R"
29          ],
30          "options": [
31            {
32              "name": "-u",
33              "value": "%{mdb:uid}"
34            }
35          ],
36          "arguments": [
37            "imapc:"
38          ]
39        }
40      }
41   }'
```

3.6.3 3. Create a Target

If you have already started a DMF Worker, then there is no need to do this. The first Worker to start
for a particular Target will define the Target in DMF based on their config value:

- `dmf.worker.identity.target`

Otherwise, you should name the Target based on the value that you will configure in the Workers
for that Target. If your client/customer is still using the now deprecated API, then it is required to
use a Target named "default".

3.6.3.1 Example Target

```
1   curl -X 'POST' \
2     'https://scheduler:8443/dmf/admin/api/v2/targets' \
3     -H 'accept: application/json' \
4     -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
5     -H 'Content-Type: application/json' \
6     -d '{
7     "name": "default"
8   }'
```

3.6.4   4. Submit a Migration

- It is possible to first "create" a DMF User record without submitting a migration. However, if you add a new migration for a user who does not exist in DMF, then a DMF User record will be implicitly created.
- This is where the API Key we set during the "Create a Source" step comes into play. Use it in the `X-API-KEY` header to authenticate with the API. This will also tell the API which Source to submit this migration in.
- The `sourceUid` should be the user's mail identifier on the Source system.
- The `targetUid` should be the user's new mail identifier on the Target system.
- The `target` should be the name of the Target we just created.
- The `sourcehost` should be the name of the Sourcehost we just created.
- The next time you submit a migration for this user, you will only need to specify the `sourceUid` and `phase`.

3.6.4.1   Example Submit

```
1   curl -X 'POST' \
2     'https://scheduler:8443/dmf/api/v2/users/migrations' \
3     -H 'accept: application/json' \
4     -H 'X-API-KEY: b270b24a-711d-47ee-8bdc-f59ad5c1abf2' \
5     -H 'Content-Type: application/json' \
6     -d '{
7       "sourceUid": "source_user1",
8       "phase": "PRE_SYNC",
9       "priority": true,
10      "targetUid": "target_user1",
11      "target": "default",
12      "sourcehost": "imap.host.1"
13   }'
```

3.6.5   5. Review the Migration

At any point, you can get the details of the latest migration record for any user you have submitted a migration for.

3.6.5.1   Example Review

```
1   curl -X 'GET' \
2     'https://localhost:7443/dmf/api/v2/users/source_user1/migrations/latest' \
3     -H 'accept: application/json' \
4     -H 'X-API-KEY: b270b24a-711d-47ee-8bdc-f59ad5c1abf2'
```

3.6.5.2   Example Response

```
1   {
2     "id": 1,
3     "sourceUid": "user1",
4     "created": "2021-04-16T12:50:08.99Z",
5     "status": "SYNC",
6     "phase": "PRE_SYNC",
7     "priority": false,
8     "started": "2021-04-16T12:50:09.00Z",
9     "backend": "default/dmf-worker",
10    "modified": "2021-04-16T12:50:10.16Z"
11   }
```

Notice the `status` of the record says `SYNC`. This means that the migration is still ongoing. We get a lot of information about the migration already though. We know when we submitted (created) it, which Worker is processing it, when the Worker started processing it, and the last time the Worker updated the record.

When the migration has finally completed (status SUCCESS, FAILED, ABORTED), if we pull the migration record again, we might see:

```
1  {
2    "id": 1,
3    "sourceUid": "source_user1",
4    "created": "2021-04-16T12:50:08.99Z",
5    "status": "FAILURE",
6    "phase": "PRE_SYNC",
7    "priority": false,
8    "started": "2021-04-16T12:50:09.00Z",
9    "stopped": "2021-04-16T12:51:20.51Z",
10   "details": [
11     {
12       "count local mailbox pre sync": {
13         "success": true,
14         "command": "/usr/bin/doveadm -f tab mailbox status -u target_user1 \"messages
                vsize\" INBOX/* INBOX *",
15         "exitCode": 0
16       }
17     },
18     {
19       "dsync": {
20         "command": "/usr/bin/doveadm -o imapc_user=source_user1 -o imapc_host=imap.host.1
                -o imapc_password=<hidden> backup -R -u target_user1 imapc:",
21         "exitCode": 75,
22         "success": false,
23         "attempts": 5,
24         "errors": [
25           "Dsync failed with return code 75 after 5 attempts. Reason: Temporary failure.
                Hint: Re-running the migration at a later time will usually resolve this",
26           "dsync(target_user1): Error: Failed to initialize user: imapc: Login to imap.
                host.1 failed: Authentication failed: [AUTHENTICATIONFAILED] Authentication
                failed."
27         ]
28       }
29     }
30   ],
31   "backend": "default/dmf-worker",
32   "modified": "2021-04-16T12:51:20.58Z",
33   "expungedMessageCount": 0,
34   "syncMessageCount": 0,
35   "syncSpeed": 69958
36 }
```

This time we found that the migration FAILED. There is another JSON key this time called details which provides details about what operations were done during the migration. That JSON blob lists the operation blocks in order. We can see that the count local mailbox pre sync operation succeeded and what command was executed. However, the main sync command dsync failed after 5 attempts (our configuration says to retry up to 5 times with 15s sleep in between failures with error code 75 - hence the 69s syncSpeed).

Finally, the errors block gives a hint about what went wrong as well as the last error message from the doveadm logging. This error is self explanatory, we used a user that does not exist on our Source-host imap.host.1. However, if we did not know what the issue was there are several ways we can debug it.

3.6.5.2.1   Review the Command   The migration details metadata gives us the command (minus password) that was constructed and executed by the Worker. With this, we could inspect it for errors, or manually run the command and review the output.

3.6.5.2.2   Review the Log   If the command logging is enabled. Example:

```
1  dmf:
2    worker:
```

```
3    command:
4      logging:
5        type: file
6        file:
7          format: "%(source)-%(user)-user-migration.log"
8          location: /app
```

Then we can review the logging of the migration command in the file: `/app/CustomerX-target_uid-user-migration.log`.

After we have reviewed the issue and found that the source user does not exist, we can try submiting a new migration with a real user and find a better migration result:

```
1  {
2    "id": 3,
3    "sourceUid": "3@234",
4    "created": "2021-04-16T13:10:40.03Z",
5    "status": "SUCCESS",
6    "phase": "PRE_SYNC",
7    "priority": true,
8    "started": "2021-04-16T13:10:42.00Z",
9    "stopped": "2021-04-16T13:10:44.20Z",
10   "details": [
11     {
12       "count local mailbox pre sync": {
13         "success": true,
14         "command": "/usr/bin/doveadm -f tab mailbox status -u 3@600 \"messages vsize\"
               INBOX/* INBOX *",
15         "exitCode": 0
16       }
17     },
18     {
19       "dsync": {
20         "command": "/usr/bin/doveadm -o imapc_user=3@234 -o imapc_host=imap.host.1 -o
               imapc_password=<hidden> backup -R -u 3@600 imapc:",
21         "exitCode": 0,
22         "success": true,
23         "attempts": 1,
24         "saved": {
25           "INBOX": 150
26         }
27       }
28     },
29     {
30       "count local mailbox post sync": {
31         "success": true,
32         "command": "/usr/bin/doveadm -f tab mailbox status -u 3@600 \"messages vsize\"
               INBOX/* INBOX *",
33         "exitCode": 0
34       }
35     },
36     {
37       "count remote mailbox": {
38         "success": true,
39         "command": "/usr/bin/doveadm -f tab -o mail=imapc: -o imapc_user=3@234 -o
               imapc_ssl=no -o imapc_host=imap.host.1 -o imapc_port=143 -o imapc_password=<
               hidden> mailbox status -u 3@234 \"messages vsize\" INBOX/* INBOX *",
40         "exitCode": 0
41       }
42     }
43   ],
44   "backend": "default/dmf-worker",
45   "modified": "2021-04-16T13:10:44.21Z",
46   "targetMessageCount": 150,
47   "targetMailboxSize": 142255,
48   "sourceMessageCount": 150,
49   "sourceMailboxSize": 142255,
50   "expungedMessageCount": 0,
51   "syncMessageCount": 150,
52   "syncSpeed": 1244
```

```
53   }
```

Now with a SUCCESS we get significantly more information back. Looking at the dsync block again, this time it is success: true still with the same information as last time, except we also get a new saved block which tells us which mailboxes and how many mails were saved during the sync. If there were deleted mails then we would also see an expunged block.

There is also now several more operations that were executed that are listed after the dsync block. Finally, we get summarized numbers for source and target message count and mailbox size, as well as how many mails were added or removed during the sync. We also see that the sync took only 1244ms to complete. This number is not the entire migration, but just the time to sync mailboxes.

3.6.6   Handling Lost Users

A lost user is one that has been pushed IN_PROGRESS, but never completes. There are two, hopefully rare, reasons this may happen.

3.6.6.1   Worker Crashes   If the Worker crashes while it is processing migration jobs, then it will never know if the migrations it started ever completed. With that, the sourcehost connection is also still reserved since the database never heard back from the Worker.

3.6.6.2   Worker Cannot Update   The Worker will do its best to update the status of a migration job, however, when things beyond its control such as database failure occur, it can only eventually give up and log the incident: Failure to send job response ....

In both scenarios, a user migration will appear to be in progress, but in reality it has been lost. With that, the sourcehost connection is also still reserved since the database never heard back from the Worker.

To resolve this issue, you should first verify that the migration is **not** running on the Worker which can be done by reviewing the processes on that server and verifying that there is a not a process with the migration command containing that user's source or target uid.

Once that has been verified, you can manually close the user record with the Scheduler REST API:

```
1   curl -X 'PUT' \
2     'https://localhost:7443/dmf/api/v2/users/source_uid/migrations/close/latest/as/ABORTED'
          \
3     -H 'accept: */*' \
4     -H 'X-API-KEY: b270b24a-711d-47ee-8bdc-f59ad5c1abf2' \
5     -H 'Content-Type: application/json'
```

This will close the migration as ABORTED and release the sourcehost connection.

In the future, we will provide resources to easily identify and verify the incident and in some cases self-correct it.

3.6.7   Migration Production Test

By default, DMF does not make any changes to the Source system so it is safe to execute dry runs as long as no configured command makes a change to the Source or proxy systems.

For instance, if you configure a pre cutover command to lock the Source user's account, then this command should be disabled while testing production customer data.

4 Scheduler Install

The DMF Scheduler is a stateless micro service. As such, you can deploy any number of Schedulers based on your needs.

Currently, the Scheduler is only used by the client/operator to submit and review migrations, and manage the DMF components.

4.1 Install the Package

The Scheduler can be installed with package `open-xchange-dmf-scheduler`. You will find that the package requires JRE8.

Example:

```
1   apt-get install open-xchange-dmf-scheduler
```

This package registers a systemd service script called `dmf-scheduler`.

You will find all related application files under `/opt/open-xchange/dmf/scheduler` and `/opt/open-xchange/sbin`.

4.2 Configure the Application

Once installed, you can find the configuration file at: `/opt/open-xchange/dmf/scheduler/etc/dmf-scheduler.yml`. All properties can also be set as environment variables.

For instance, `http.admin.username` would be `HTTP_ADMIN_USERNAME`, while it would be configured as follows in `dmf-scheduler.yml`:

```
1   http:
2     admin:
3       username: admin
```

Environment variables have precedence over configuration file settings.

4.2.1 Configure HTTPS

Review the Micronaut HTTPS documentation and examples to configure TLS.

Use keys under `micronaut.ssl` to configure the server. The default configuration expects a private key and the corresponding certificate in `/opt/open-xchange/dmf/certs/keystore.p12`

This file can be easily generated by running the following:

```
1   /opt/open-xchange/sbin/dmf-scheduler-gen-certs -d /opt/open-xchange/dmf/certs
```

The script `dmf-scheduler-gen-certs` is installed as part of the `open-xchange-dmf-scheduler` package. In addition to `keystore.p12` for the Scheduler, the script also generates `scheduler.p12` in the same directory. This file contains the self-signed certificate, and can be used by clients to verify the identity of the Scheduler.

For distributed setups, when there are multiple Scheduler nodes, the certificate should be generated only once, and then the generated files distributed to all corresponding nodes.

As a side-effect, the script also generates `scheduler.pem`, which is the same self-signed certificate in a more popular format. It can be used by browsers and other clients, but is not necessary for DMF operation.

If the Scheduler operates behind a web server or any other proxy which performs the actual TLS termination, and also uses a self-signed certificate, then its certificate can be converted to the right

format manually, using Java's `keytool`. See the last step in the `dmf-scheduler-gen-certs` script for an example.

An example configuration:

```
1  micronaut:
2    ssl:
3      enabled: true
4      key-store:
5        path: file:/opt/open-xchange/dmf/certs/keystore.p12
6        type: PKCS12
7        password: verysecretpassword
8      port: 8443
```

4.2.2 Configure Authentication

Basic authentication is used to authenticate HTTP clients that use the Admin API. This can be configured like so:

```
1  http:
2    admin:
3      username: admin
4      password: verysecretpassword
```

API documentation via redoc, rapidoc, and swagger UI's are available:

4.2.2.1 API Links

```
1  https://<hostname>/rapidoc
2  https://<hostname>/redoc
3  https://<hostname>/swagger-ui
```

By default, they can be accessed anonymously, but you can restrict them via basic auth by changing the access level of those paths:

```
1   micronaut:
2     security:
3       intercept-url-map:
4         - pattern: /swagger/**
5           access:
6             - isAuthenticated()
7         - pattern: /swagger-ui/**
8           access:
9             - isAuthenticated()
10        - pattern: /rapidoc/**
11          access:
12            - isAuthenticated()
13        - pattern: /redoc/**
14          access:
15            - isAuthenticated()
```

The API static resources can be removed by removing their references in the configuration under the `micronaut.router.static-resources` configuration.

4.2.2.2 Endpoints All built-in Micronaut Endpoints, or custom endpoints, are restricted by default, but any can be configured to be accessed anonymously:

```
1  endpoints:
2    info:
3      sensitive: false
```

4.2.2.3 Security   Restricting access to HTTP resources is enabled using the property:

- `micronaut.security.enabled`

You can also restrict clients by IP by using the `micronaut.security.ip-patterns` property.

```
1  micronaut:
2    security:
3      enabled: true
4      ip-patterns:
5        - 127.0.0.1
6        - 192.168.1.*
```

4.2.3   Configure Data Source

The Scheduler must talk to the Migration Database and this is the only data source you need to configure. Aside from basic connection properties, the data source is highly configurable using any of the JDBC Hikari properties.

ℹ️ **Info**
If `createDatabaseIfNotExist=true` is not used in the JDBC URL, you must create a database name `migration` before starting the Scheduler.

ℹ️ **Info**
The configured database user must have `create` and `drop` table permissions.

Example configuration:

```
1  datasources:
2    default:
3      url: jdbc:mysql://dmf-db:3306/migration?createDatabaseIfNotExist=true
4      username: scheduler
5      password: verysecretpassword
6      dialect: MYSQL
7      driverClassName: org.mariadb.jdbc.Driver
```

4.3   Manage the Application

The application can be started/stopped/restarted using the systemd script `dmf-scheduler`.

Start example:

```
1  systemctl start dmf-scheduler
```

Stop example:

```
1  systemctl stop dmf-scheduler
```

5   Command Guide

This guide lays out the structure of a command, as well as how to define commands within DMF.

DMF's concept of a `Command` equates to that of a shell command. There are several places within DMF where shell commands can be executed, and this includes the main migration command.

⚠️ **Warning**
DMF will not obfuscate passwords in commands that are recorded in logs or migration records unless otherwise stated. For instance, with the doveadm commands. See the Doveadm Password section.

5.1 Structure

The structure of a Command is pretty simple.

Shell commands can be summarized as:

- `command [options] [arguments] [sub command]`

Similarly, DMF commands can be summarized as:

- `name [flags] [options] [arguments] [sub command]`

While naming is somewhat different, they are actually the same structure.

5.1.1 Name

The command name is just that. For instance, when you use the `doveadm` command, the command name is `doveadm`.

5.1.2 Success Code

The success code is the code that a command returns when it has completed successfully. In most cases this is 0, but can be changed to any signed integer value.

5.1.3 Flags

Flags are command `options` that do not use a parameter. For instance, `-l`, `-r`, `-t` are all flags.

5.1.4 Options

Options take an option name and a value: `name value`

Typically, the option name begins with a hyphen. For instance, `-M INBOX` is an option.

5.1.5 Arguments

Arguments are single structure values. For instance, `"messages size"` is an argument.

5.1.6 Sub Command

The structure of a sub command is the structure of a command. For instance, in the command `doveadm mailbox status`, the command `doveadm` has a subcommand `mailbox` which has a subcommand `status`.

5.2 Format

5.2.1 JSON

The command `doveadm -D -o imapc_user=user1 -o imapc_password=secret -o imapc_host=host backup -R -u user1 imapc:` can be structured as the following in JSON:

```
1  {
2      "name": "doveadm",
3      "successCode": 0,
4      "flags": [
5        "-D"
6      ],
7      "options": [
8        {
9          "name": "-o",
10         "value": "imapc_user=user1"
11       },
12       {
```

```
13          "name": "-o",
14          "value": "imapc_password=secret"
15        },
16        {
17          "name": "-o",
18          "value": "imapc_host=host"
19        }
20      ],
21      "subCommand": {
22        "name": "backup",
23        "flags": [
24          "-R"
25        ],
26        "options": [
27          {
28            "name": "-u",
29            "value": "user2"
30          }
31        ],
32        "arguments": [
33          "imapc:"
34        ],
35      }
36    }
```

5.2.2   YAML

The command `doveadm -D -o imapc_user=user1 -o imapc_password=secret -o imapc_host=host` `backup -R -u user1 imapc:` can be structured as the following in YAML:

```
1   name: doveadm
2   flags: -D
3   options:
4     - name: -o
5       value: imapc_user=user1
6     - name: -o
7       value: imapc_password=secret
8     - name: -o
9       value: imapc_host=host
10  sub-command:
11    name: backup
12    flags: -R
13    arguments: "imapc:"
14    options:
15      - name: -u
16        value: user2
```

5.3   Property Injection

During migration job processing, DMF provides the ability to inject pre defined values into your commands.

5.3.1   Standard Properties

You can use the formatter `%{mdb:X}` where X is one of the following:

- `uid`: the user's targetUid
- `ruid`: the user's sourceUid
- `sourcehost`: the user's Sourcehost
- `2chrruid`: the first two characters of the user's sourceUid
- `container`: see the DMF Doveadm Features documentation
- `md5path`: the uid converted to md5 hex then formed into `<first2chars>/<next2chars>`
- `source`: the user's DMF Source
- `sourcepasswd`: the user's source password
- `imapcoptions`: value of imapcOptions

- `email`: the user's email
- `sourceport`: the user's sourcehost port
- `imapc_ssl`: the value of useImapcSsl
- `exclude`: the list of folders to exclude

For example:

```
1  {
2    "name": "echo",
3    "arguments": ["%{mdb:uid}"]
4  }
```

would result in a command `echo user1` for a migration where the targetUid is `user1`.

5.3.2   Doveadm Properties

When using the doveadm Worker, you can use the formatter `%{conf:Y}` where Y is one of the following:

- `imapc_host`: the value of config property

  `dmf.doveadm.source.<source>.imapc.host`

- `imapc_master_password`: the value of config property

  `dmf.doveadm.source.<source>.imapc.master-password`

- `imapc_master_user`: the value of config property

  `dmf.doveadm.source.<source>.imapc.master-user`

- `imapc_prefix`: the value of config property

  `dmf.doveadm.source.<source>.imapc.prefix`

- `imapc_port`: the value of config property

  `dmf.doveadm.source.<source>.imapc.port`

Additionally, you can define any other properties via the following config property to inject using the formatter `%{conf:Z}` where Z is a key under property `dmf.doveadm.source.<source>.command.inject`

For instance, if you define:

```
1  dmf:
2    doveadm:
3      source:
4        default:
5          command:
6            inject:
7              mykey: myvalue
```

with the command:

```
1  {
2    "name": "echo",
3    "arguments": ["%{conf:myvalue}"]
4  }
```

then you will end up with the command: `echo myvalue`

You can also override any of the imapc or mdb properties by using the inject config strategy.

5.4   Doveadm

When the doveadm Worker is used, any time a doveadm command is specified, there are four special features.

5.4.1 Passwords

Commands in DMF could be logged or stored in the database at any point, thus it's important to understand how passwords in commands are stored to ensure that unencrypted passwords are not leaked into places they should not be.

When you use any of the following keys within the migration or status commands as **options**, their value will be replaced with "`<hidden>`" when logged or stored in the migration database:

- imapc_password
- imapc_master_password
- pop3c_password
- pop3c_master_password
- doveadm_password

For instance, if you use the following migration command:

```
1  {
2    "name": "doveadm",
3    "options": [
4      {
5        "name": "-o",
6        "value": "imapc_password=%{conf:imapc_master_password}"
7      }
8    ],
9    ...
10 }
```

then the command would be logged like: `doveadm -o imapc_password=<hidden>`

⚠️ **Warning**

These passwords are only obfuscated when those keys are used as command options (not flags, names, arguments).

⚠️ **Warning**

Do not ever assume that other commands, such as the post sync command, will obfuscate passwords. All commands should first be tested with non sensitive passwords to ensure that they are not leaked.

5.4.2 Options Format

There are two ways to specify setting overrides for the doveadm base command.

5.4.2.1 1. Using `-o`

```
1  "options": [
2    {
3      "name": "-o",
4      "value": "imapc_user=user1"
5    }
6  ]
```

- The doveadm Worker will take care to correctly format the setting overrides.

5.4.2.2 2. Not using `-o`

```
1  "options": [
2    {
3      "name": "imapc_user",
4      "value": "user1"
5    }
6  ]
```

- Since the only other possible option is `-f`, the Worker is able to correctly format all other options as setting overrides.
- This option for a doveadm command would result in: `-o imapc_user=user1`.

5.4.3 Command Syntax

The Worker will work to ensure that any doveadm command that you use is syntactically correct, however, it does not know if it is semantically correct.

For instance, if your migration command is:

```
1  {
2    "name": "doveadm",
3    "arguments": "this shouldn't be here",
4    ...
5  }
```

then DMF will fail the migration because doveadm does not take any arguments, but if you were to describe a doveadm command that uses options that don't make sense together, then DMF will not complain and that issue would be found at command execution time.

5.4.4 Exclude Folders

When defining a user in DMF, you have the option to provide a list of folders in a field called excludeFolders. When using the property injection formatter `{mdb:exclude}`, DMF will convert it to the DoveAdmSync folder exclusion format.

For example, if you specify for a user:

```
1  {
2    "excludeFolders": ["INBOX", "INBOX2"]
3  }
```

with the following migration command:

```
1  {
2    ...
3    "subCommand": {
4      "name": "backup",
5      "arguments": ["{%mdb:exclude}"],
6      ...
7    }
8  }
```

then DMF will format the command like: `doveadm ... backup ... -x INBOX -x INBOX2`

6 REST API

The DMF REST API is self documented through a generated openapi yaml file. By default, this file is exposed through the Scheduler application via three different apps:

- Swagger UI: `https://worker:8443/swagger-ui`
- Rapidoc: `https://worker:8443/rapidoc`
- Redoc: `https://worker:8443/redoc`

Both swagger-ui and rapidoc can be used to interact with the Scheduler API, while redoc just provides the API specification.

6.1 APIs

There are currently three different types of APIs.

6.1.1 Admin

The purpose of the Admin API is to provide a single administrator the ability to manage the DMF platform. This API uses a single pair of basic auth credentials.

The relationship of the resources that the Admin API exposes is represented by the following image:

Figure 4: adminapi

6.1.1.1 Sources   Sources are the Source platforms that you migrate from. When creating a Source, you specify an apiKey that will be used for the Customer/Migration API.

6.1.1.2 Sourcehosts   Sourcehosts are the mail servers of a Source that you migrate from. When adding a Sourcehost, you will specify the following:

- `maxConnection`: the max number of migration connections available. This means that Workers who can migrate users from this Sourcehost will never concurrently migrate more users than this.
- `migrationCommand1`: this command is executed during the pre-sync phase, or during the cutover phase if migrationCommand2 is not provided. See the DMF Command Guide on how to format the command.
- `migrationCommand2`: this command is executed during the cutover phase if specified.
- `statusCommand`: this command is executed when DMF counts the user's Source mailbox size and messages. When using the doveadm Worker this would be the `doveadm mailbox status` command. It is optional and DMF will create the command based on configuration if configured to do so.

All "commands" can also be specified in the Worker configuration instead.

6.1.1.3 Targets   Targets are the Target platforms that you migrate into. The concept of a Target allows you to manage Workers of a target, as well as to enforce migration constrains across Sources.

6.1.1.4 Backends   Backend is synonymous for Worker. This API allows you to review the status of Workers, as well as update their properties.

You can start, pause, and stop a Worker from here, as well as change their initial state and max threads.

6.1.2 Customer/Migration

The Migration (aka Customer) API is the API used to define users within DMF and submit migrations. The reason for "customer" is because this API can be exposed to customers who can be provided the API Key for their Source. This would allow your customer to control the migration schedule while leaving the management of the platform to you.

The Migration API is about managing user migrations whose relationship with the other resources can be represented by the following image:
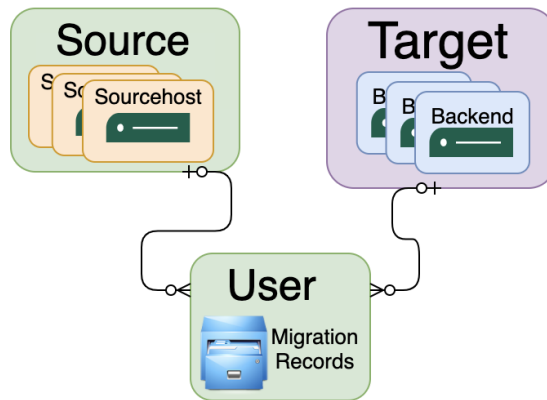
Figure 5: userapi

6.1.3 Users

A user is a record containing the properties of a user account from a Source which will be migrated to a Target.

When creating a new user, there are a number of optional properties that can be specified and injected into the migration or other commands.

It is optional to first create the user in DMF, and you can instead simply submit a migration using the same properties. This will implicitly create a user record.

Once a user has been created, or a first migration has completed, you can submit a migration with just the fields `sourceUid` and `phase`. Setting the priority field to true will force Workers to process that migration before others in the queue without priority. This is useful when running cutovers alongside pre-syncs, to ensure that new cutovers are processed before pre-syncs.

6.1.4 Deprecated

Most of the old DMF HTTP API has been included with this version of DMF to ensure backward compatibility with clients, however, it will soon be removed in a future version.

7 Migration Database

The Migration Database is used by both the Scheduler and Worker to manage migration components and migration jobs.

At first look, you may notice interesting relationships between tables. This is because of DMF's custom data sharding.

Whenever a new Source is created, the two tables `users` and `journal` are duplicated into `usersSourceName` and `journalSourceName`. This is to reduce the amount of migration data between Sources and introduce a small layer of data separation.

That means that the base `users` and `journal` tables will never have any data in them, and instead the data will go into the source-based tables.

7.1 Table Descriptions

7.1.1 Sources

The `sources` tables holds DMF Sources.

7.1.2 Sourcehosts

The `sourcehosts` table holds all Sourcehosts of a Source.

7.1.3 Targets

The `targets` table holds all DMF Targets.

7.1.4 Backends

The `backends` table holds all Worker state data.

7.1.5 Target UIDs

The `target_uids` table is used to enforce Target mail uid uniqueness across Sources. Since each Source has their own user data tables, it's not possible to enforce a unique target uid without this.

7.1.6 Users

The `users` table is used to hold all static user data. This table is constructed for every Source using table name `usersSourceName`.

7.1.7 Journal

The `journal` table is used to hold all migration records of a user. This table is constructed for every Source using table name `journalSourceName`.
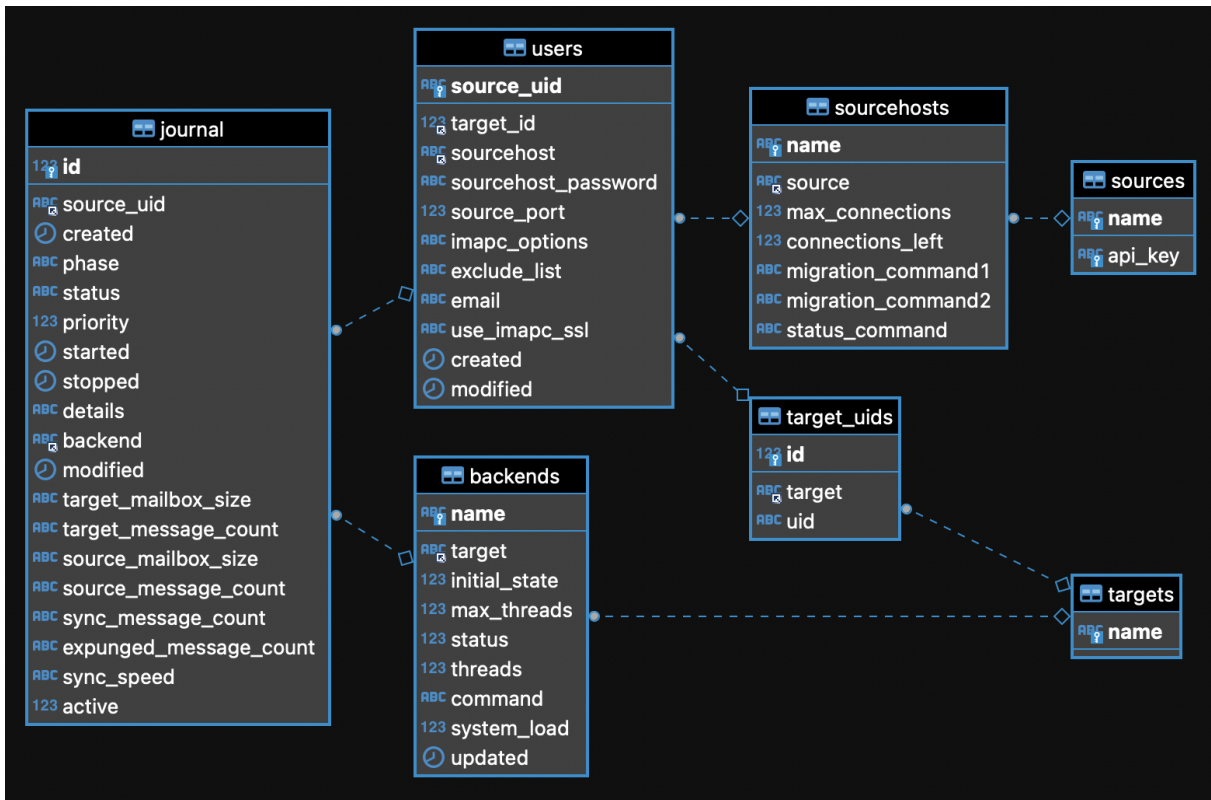
7.2 Entity Relationship Diagram

Figure 6: db-erd

8 Scheduler High Availability

The DMF Scheduler is the component that serves the following purposes:

- provides an API to manage Sources, Sourcehosts, Targets, Backends(Workers)
- provides an API to schedule new migrations
- provides an API to fetch information about ongoing and finished migrations
- creates and manages the migration database

As such, it is a critical component of the migration process which should be running at all times or at least with little downtime.

When the Scheduler is not running:

- it is not possible to submit migrations
- it is not possible to query information about the migration

This would be an issue for automtated client that use DMF for submitting and monitoring progress of migrations.

It does not, however, prevent previously submitted migrations from being processed by Worker nodes.

Operating multiple instances of the Scheduler in an Active/Active scenario is easy:

- the Scheduler only caches uncritical information, such as for metrics and health monitoring, everything else being persisted and fetched live from the Migration database

Furthermore, a monitoring HA load-balancer (e.g. ha-proxy) which performs active/passive proxying of HTTP(S) traffic to the multiple Scheduler nodes would be helpful in order to provide stable URIs for DMF clients.

Scheduler health checking can be set up over HTTP(S) even without authentication, by querying its `/health` URL, as documented in DMF Scheduler Health. When doing so, one can also control which health monitors should be included or disabled.

9 Scheduler Health

As part of the Micronaut framework, each Scheduler node monitors several components and reports a health check, which is reachable under the path `/health`.

It's possible to configure the endpoint to be reachable without authentication and provide a simple status output, and then all other details when authenticated.

9.1 Without Authentication

```
1   curl https://scheduler:8443/health
```

Sample output:

```
1   {
2     "status" : "UP"
3   }
```

9.2 With Authentication and Full Details

```
1   curl -u admin:secret https://scheduler:8443/health
```

Sample output:

```
1   {
2     "name": "api",
3     "status": "UP",
4     "details": {
5       "jdbc": {
6         "name": "api",
7         "status": "UP",
8         "details": {
9           "jdbc:mysql://dmf-db:3306/migration?createDatabaseIfNotExist=true": {
10            "name": "api",
11            "status": "UP",
12            "details": {
13              "database": "MariaDB",
14              "version": "10.5.4-MariaDB-1:10.5.4+maria~focal"
15            }
16          }
17        }
18      },
19      "compositeDiscoveryClient()": {
20        "name": "api",
21        "status": "UP",
22        "details": {
23          "services": {}
24        }
25      },
26      "service": {
27        "name": "api",
28        "status": "UP",
29        "details": null
30      },
31      "diskSpace": {
32        "name": "api",
33        "status": "UP",
34        "details": {
35          "total": 126557421568,
36          "free": 71800705024,
37          "threshold": 10485760
38        }
39      }
40    }
41  }
```

9.3 Configuration

Individual health indicators can be turned off with configuration settings, which can be specified through modifying the `dmf-scheduler.yml` configuration file or through environment variables.

Configuration Property	Indicator Tree	Description
`endpoints.health.disk-space.enabled`	diskSpace	Monitors the available disk space of a configurable path and threshold:`endpoints.health.disk-space.path` (defaults to ".")`endpoints.health.disk-space.threshold` (in bytes, defaults to 10 MB)
`endpoints.health.jdbc.enabled`	jdbc	Monitors databases.

10 Scheduler Metrics

Each Scheduler node exports a number of metrics, currently all being provided by the Micronaut framework. Its metrics API provides JSON data and also offers a Prometheus API.

Note that authentication **is** required to query metrics and their values by default.

28

To change that behavior and not require authentication, set the `endpoints.metrics.sensitive` configuration property to `false`, either in the configuration file `dmf-scheduler.yml` or in as an environment variable.

The whole metrics API can also be disabled altogether by setting `endpoints.metrics.enabled` to `false`.

10.1 List of Metrics

A list of metric names can be queried using

```
curl -u admin:secret https://scheduler:8443/metrics
```

Sample output:

```
{
  "names": [
    "executor",
    "executor.active",
    "executor.completed",
    "executor.pool.core",
    "executor.pool.max",
    "executor.pool.size",
    "executor.queue.remaining",
    "executor.queued",
    "hikaricp.connections",
    "hikaricp.connections.acquire",
    "hikaricp.connections.active",
    "hikaricp.connections.creation",
    "hikaricp.connections.idle",
    "hikaricp.connections.max",
    "hikaricp.connections.min",
    "hikaricp.connections.pending",
    "hikaricp.connections.timeout",
    "hikaricp.connections.usage",
    "http.server.requests",
    "jvm.buffer.count",
    "jvm.buffer.memory.used",
    "jvm.buffer.total.capacity",
    "jvm.classes.loaded",
    "jvm.classes.unloaded",
    "jvm.gc.live.data.size",
    "jvm.gc.max.data.size",
    "jvm.gc.memory.allocated",
    "jvm.gc.memory.promoted",
    "jvm.gc.pause",
    "jvm.memory.committed",
    "jvm.memory.max",
    "jvm.memory.used",
    "jvm.threads.daemon",
    "jvm.threads.live",
    "jvm.threads.peak",
    "jvm.threads.states",
    "logback.events",
    "process.cpu.usage",
    "process.files.max",
    "process.files.open",
    "process.start.time",
    "process.uptime",
    "system.cpu.count",
    "system.cpu.usage",
    "system.load.average.1m"
  ]
}
```

10.2   Query a Metric

Querying a specific metric can be achieved as follows:

```
1  curl -u admin:secret https://scheduler:8443/metrics/process.uptime
```

Sample output:

```
1   {
2     "name": "process.uptime",
3     "measurements": [
4       {
5         "statistic": "VALUE",
6         "value": 168503.498
7       }
8     ],
9     "availableTags": [],
10    "description": "The uptime of the Java virtual machine",
11    "baseUnit": null
12  }
```

10.3   Prometheus Metrics API

The values of all metrics can be fetched in Prometheus' format using the /prometheus endoint:

```
1  curl -u admin:secret https://scheduler:8443/prometheus
```

A portion of the sample output:

```
1   # HELP hikaricp_connections_active Active connections
2   # TYPE hikaricp_connections_active gauge
3   hikaricp_connections_active{pool="HikariPool-1",} 0.0
4   # HELP jvm_buffer_memory_used_bytes An estimate of the memory that the Java virtual
        machine is using for this buffer pool
5   # TYPE jvm_buffer_memory_used_bytes gauge
6   jvm_buffer_memory_used_bytes{id="direct",} 3.35544376E8
7   jvm_buffer_memory_used_bytes{id="mapped",} 0.0
8   # HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in
        this pool
9   # TYPE jvm_buffer_total_capacity_bytes gauge
10  jvm_buffer_total_capacity_bytes{id="direct",} 3.35544375E8
11  jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
```

11   Shipped Packages and Version

11.1   Package open-xchange-dmf-scheduler

DMF Scheduler Dovecot Migration Framework Scheduler.
Version: 1.0.0-6
Type: Other

11.1.1   Installation

Install on nodes with package installer **apt-get** or **yum**:

```
<package installer> install open-xchange-dmf-scheduler
```

11.1.2 Configuration

For details, please see appendix
/opt/open-xchange/dmf/scheduler/etc/dmf-scheduler.yml (page)

A Configuration Files

File 1 /opt/open-xchange/dmf/scheduler/etc/dmf-scheduler.yml

```
1   # Required for legacy API which should have
2   # all keys in the json response
3   jackson:
4     serializationInclusion: ALWAYS
5   ---
6   http:
7     admin:
8       username:
9       password:
10    headers:
11      xapikey: X-API-KEY
12  ---
13  micronaut:
14    # SSL configuration
15    # Required for production environments.
16    # See https://docs.micronaut.io/latest/guide/index.html#https for details.
17    ssl:
18      enabled: true
19      key-store:
20        path: file:/opt/open-xchange/dmf/certs/keystore.p12
21        type: PKCS12
22        password:
23      port: 8443
24    router:
25      versioning:
26        enabled: true
27        default-version: v1
28        parameter:
29          enabled: false
30          names: 'v,api-version'
31        header:
32          enabled: true
33          names:
34            - 'X-API-VERSION'
35            - 'Accept-Version'
36      # Allows the openapi views to be seen
37      static-resources:
38        swagger:
39          paths: classpath:META-INF/swagger
40          mapping: /swagger/**
41        redoc:
42          paths: classpath:META-INF/swagger/views/redoc
43          mapping: /redoc/**
44        rapidoc:
45          paths: classpath:META-INF/swagger/views/rapidoc
46          mapping: /rapidoc/**
47        swagger-ui:
48          paths: classpath:META-INF/swagger/views/swagger-ui
49          mapping: /swagger-ui/**
50    security:
51      enabled: true
52      # Change the security of the open api views to anonymous so that they can be viewed
           without credentials
53      intercept-url-map:
54        - pattern: /swagger/**
55          access:
56            - isAnonymous()
57        - pattern: /swagger-ui/**
58          access:
```

```
 59              - isAnonymous()
 60          - pattern: /rapidoc/**
 61            access:
 62              - isAnonymous()
 63          - pattern: /redoc/**
 64            access:
 65              - isAnonymous()
 66    application:
 67      name: api
 68    metrics:
 69      enabled: true
 70      export:
 71        prometheus:
 72          enabled: true
 73          step: PT1M
 74          descriptions: true
 75  ---
 76  # This way of defining the datasource properties means that we can externalize the
          configuration,
 77  # for example for production environment, and also provide a default value for development
          .
 78  # If the environment variables are not defined Micronaut will use the default values. Also
           keep in
 79  # mind that it is necessary to escape the : in the connection url using back ticks `.
 80  datasources:
 81    default:
 82      # url should use createDatabaseIfNotExist=true if the database will not
 83      # already exist: https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-
          configuration-properties.html
 84      url: jdbc:mysql://localhost:3306/migration?createDatabaseIfNotExist=true
 85      username:
 86      password:
 87      dialect: MYSQL
 88      driverClassName: org.mariadb.jdbc.Driver
 89  ---
 90  liquibase:
 91    datasources:
 92      default:
 93        change-log: 'classpath:liquibase/scheduler/liquibase-changelog.xml'
 94  ---
 95  endpoints:
 96    all:
 97      enabled: true
 98      sensitive: true
 99    liquibase:
100      # fails with missing transition, might be fixed in later Micronaut releases
101      enabled: false
```