Dovecot Migration Framework Worker Technical Documentation for 1.2.0-rev10

2023-12-07

Copyright notice

Contents

1 General Information

1.1 Warnings

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 Worker 1.2.0 Maintenance Delivery 10*

1.3 Install Package Repository

This delivery is part of a restricted software repository:

https://software.open-xchange.com/components/dmf-worker/stable/1.2.0/RHEL8
https://software.open-xchange.com/components/dmf-worker/stable/1.2.0/RHEL7
https://software.open-xchange.com/components/dmf-worker/stable/1.2.0/DebianBullseye
https://software.open-xchange.com/components/dmf-worker/stable/1.2.0/DebianBuster

1.4 Build Dependencies

This delivery was build and tested with following dependencies:

```
RedHat:rhel-8,RedHat:rhel-7,Debian:Buster,
Debian:Bullseye
```

2 Worker Install

The DMF Worker is a stateful service which processes the migration jobs that are added to the Migration Database queue by the DMF Scheduler. How it processes those jobs is configurable. This guide will discuss the instation of the Worker as well as all configuration options.

2.1 Install the Package

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

Example:

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

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

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

Where you install the Worker(s) is completely based upon how you plan to use it. While DMF stands for *Dovecot* Migration Framework, in reality, it is more like a *Mail* Migration Framework because nothing restricts you from using it for just Dovecot migrations. In fact, bare bones, it is just a job

processing framework that allows you to plug in any Job Worker to process your job. However, this guide will assume that you are using DMF for a Dovecot target migration. If you want to know more about using it outside of Dovecot, see the DMF Beyond Dovecot documentation.

For this purpose, you will install one DMF Worker on every Dovecot backend that you plan to migrate users into. DMF must have execute permission on the Dovecot `doveadm` shell utility.

2.2 Configure the Application

Once installed, you can find the configuration file at: `/opt/open-xchange/dmf/worker/etc/dmf-worker.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-worker.yml`:

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

Environment variables have precedence over configuration file settings.

2.2.1 Configure HTTPS

The Worker does not expose a custom API, however, it exposes all built in Micronaut Endpoints as well as a metrics endpoint for prometheus at `https://worker:8443/prometheus`.

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-worker-gen-certs -d /opt/open-xchange/dmf/certs
```

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

As a side-effect, the script also generates `worker.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 Worker 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-worker-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
```

2.2.2 Configure Authentication

Basic authentication is used to authenticate HTTP clients. This can be configured like so:

```
http:
  admin:
    username: admin
    password: verysecretpassword
```

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

```
endpoints:
  info:
    sensitive: false
```

2.2.2.2 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.

```
micronaut:
  security:
    enabled: true
    ip-patterns:
      - 127.0.0.1
      - 192.168.1.*
```

2.2.3 Configure Data Source

The Worker 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.

Info
The configured database user must have `read` and `update` permissions on the `migration` database tables.

Example configuration:

```
datasources:
  default:
    url: jdbc:mysql://dmf-db:3306/migration
    username: worker
    password: verysecretpassword
    dialect: MYSQL
    driverClassName: org.mariadb.jdbc.Driver
```

2.2.4 Configure Executor Pools

The DMF Worker makes use of Java executor pools to run migration jobs in parallel. You can find all configuration properties for these pools at Micronaut Thread Pools.

In the Worker configuration, there are two pools that you should consider configuring. The default and recommended executor type is `cached`, because threads are already limited by the max worker jobs setting.

```
micronaut:
```

```
  executors:
    worker-executor:
      name: worker-executor
      type: cached
    command-executor:
      name: command-executor
      type: cached
```

2.2.5 Configure HTTP Services

Currently, the only potential http service is for `doveadm` which you find under `micronaut.http.services.doveadm`. If you are using this, to make calls to the doveadm http api, which is not typically the case, then you can set things like ssl settings, timeouts, etc.

```
micronaut:
  http:
    services:
      ####
      # Configure the HTTP connection properties for the doveadm HTTP APIs.
      # This configuration is shared for all defined doveadm HTTP configurations.
      # This is where SSL can be enabled and configured.
      ####
      doveadm:
        ssl: {}
  #           enabled: true
  #           trust-store:
  #             path: file:/opt/open-xchange/dmf/certs/doveadm.p12
  #             password:
  #             type: PKCS12
```

2.2.6 Configure Identity

Each Worker has an identity so that you can identitfy them when managing their settings, but also so that you can trace where a migration job was processed. There are two parts to a Worker identity: target and memberid.

The Worker target should represent the Target platform. For instance, if you have a platform called “cloud”, you might name the Target “cloud” and then set this as the target property of every Worker that will migrate users to this platform. It is important to use the same Target name for all Workers in the same platform to prevent multiple Workers from migrating to the same target mailbox at the same time. Therefore, ensuring a 1-1 relationship between user and target mailbox.

Warning
If your customer/client is still using the deprecated legacy API, then you must use a single Target called “default”. This will require you to deploy a separate DMF platform for other Targets, so they should update to the new API ASAP.

The Worker’s memberid must be unique within the Target. No Worker should ever have the same memeberid as another Worker who has the same target identity.

Warning
There is not a mechanism in place to stop a Worker from stealing the identity of another Worker so care should be taken.

```
dmf:
  worker:
    identity:
      target: cloud
      memberid: 1
```

Once the Worker has been started and registered with the Migration Database, you can manage it with the Scheduler Admin Backends REST API with the Target `cloud` and name `cloud/1`:

```
curl -X 'GET' \
  'https://localhost:7443/dmf/admin/api/v2/targets/cloud/backends/cloud%2F1' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ='
```

2.2.7 Configure Sources

The Worker has the ability to service any number of DMF Sources. Remember that a DMF Source is created using the Admin HTTP API and represents a Source platform. For this purpose, we will have two Sources: `["POD_1", "POD_2"]`. Workers will get jobs for all Sources based on priority, then time of submission.

Configure the Worker to service both Sources:

```
dmf:
  worker:
    sources:
      - "POD_1"
      - "POD_2"
```

An important note is that the Sources can now be configured via the DMF REST API and so this setting will only be used the first time the Worker is started.

2.2.8 Configure State

By default, the Worker is configured to start polling for migration jobs when it is started. If you want to disable so that you can start the Worker, but not start polling for jobs until a later time, then you can change the initial state to STOPPED.

```
dmf:
  worker:
    state:
      initial: STOPPED
```

Info

Once the Worker has been started and registered with the Migration Database, this can be changed by using the Scheduler Admin REST API. At this point, the database value will override the application configuration.

```
curl -X 'PATCH' \
  'https://worker:8443/dmf/admin/api/v2/targets/cloud/backends' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "cloud/1",
  "initialState": 1
}'
```

Now that we set it to 1 (STARTED), if the Worker is ever restarted, it will start polling for jobs.

2.2.9 Configure Crypto

This section is only relevant if user passwords will be used instead of master password. Otherwise, the crypto section of the configuration can be omitted.

DMF uses a symmetric AES-256 key stored on disk and initialization vector stored in the database to wrap the user passwords that are then stored in the database.

The wrapping is done by the DMF Scheduler, however, the Worker will need to unwrap it when it is time to use the password for migration.

To support the ability to use new or different keys among Schedulers, the Worker can be configured to use any number of storage keys that can be identified based on the name that the Scheduler gave them. When the Scheduler encrypts the user's password, it also stores the name of the key in the database. That key name is used by the Worker to look up the correct key on disk.

Reference to keys can be configured under the *dmf.worker.crypto.storageKeys* property:

```
dmf:
  worker:
    crypto:
      storageKeys:
        key1:
          file: keystore:/opt/open-xchange/dmf/certs/keystore.p12
          secret: password
        key2:
          file: keystore:/opt/open-xchange/dmf/certs/keystore.p12
          secret: password
```

The key name, in this example "key1" and "key2" are the key names, is what DMF will use when finding the correct key to use.

The "file" can either be a plain file with the key as encoded bytes or a Java KeyStore file. If using a keystore, then the prefix "keystore:" must be used like in the example.

An example of creating a keystore with a key called key1:

```
keytool -genseckey -alias 'key1' -keyalg 'AES' -keysize '256' -storetype 'pkcs12' -
    storepass 'password' -keystore keystore.p12
```

2.2.10 Configure Job Settings

2.2.10.1 Job Cache The Worker collects prospective migration jobs in a local cache in order to reduce the number of sorting queries performed since the database does not actually provide a priority queue. If you find that the Worker has seemingly unnecessary high memory usage, then it may be beneficial to reduce the size of the cache, or in the case of high database load reduce the refresh rate.

```
dmf:
  worker:
    jobs:
      cache:
        size: 400
        referesh: 1m
```

2.2.10.2 Max Jobs This is the max number of migration jobs that the Worker will process in parallel. Keep in mind that this value should be less than or equal to the `number-of-thread` configured for the `worker-executor` executor pool. If it is not, then jobs will be queued within the executor and not executed until a thread is freed, thus potentially blocking another Worker from processing it.

```
dmf:
  worker:
    jobs:
      max: 400
```

The max number of jobs you want to execute depends on the sizing of the server that the Worker is running, the Worker features you have enabled, and the migration command you are using. If unknown, it is recommened to start low and scale up. It is easy to increase the max jobs, however, difficult to stop jobs or handle an overloaded server.

Info

Once the Worker has been started and registered with the Migration Database, this can be changed by using the Scheduler Admin REST API. At this point, the database value will override the application configuration.

```
curl -X 'PATCH' \
  'https://worker:8443/dmf/admin/api/v2/targets/cloud/backends' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "cloud/1",
  "maxThreads": 120
}'
```

2.2.11 Configure Command Logging

There are three ways to configure command execution output. This is the output whenever a native command is executed, for instance, like the migration command.

1. `file`: Redirects all output to a file. You can specify the location and file naming as well as if the file should be removed upon successful execution of the command.
2. `logger`: Redirects all output to the Worker application logger
3. `off`: silences all output

```
dmf:
  worker:
    command:
      logging:
        type: file
        file:
          # The log file name format. You can use the following specifiers:
          #   * source: the source name
          #   * user: the target uid
          #   * phase: the migration phase (pre-sync, cutover)
          #   * job: the job id
          #   * date: the YYYY-MM-DD date
          format: "%(source)-%(user)-user-migration.log"
          location: /var/log/dmf
          delete-on-success: false
```

2.2.12 Configure Doveadm

This section only discusses how to setup the doveadm configuration, but you should review the DMF Doveadm Features documentation for more information on what each property is responsible for and how you should use it.

Within DMF, we have a concept of a Worker type. For this purpose, we will use the type `doveadm` which uses the `doveadm` command line tool supplied by Dovecot to process migration jobs. The documentation DMF Beyond Dovecot provides information about the other Worker types.

As already seen when configuring the Sources, the Worker supports migrating from multiple Sources to a Target. So, the `doveadm` Worker allows you to configure it per Source.

The configuration looks like:

```
dmf:
  doveadm:
    path: "/usr/bin/doveadm"
    source:
      ...
```

You can apply a special key called `default` and then all doveadm specific properties under that. In which case, the Worker will use this configuration if there is not an override.

To override the default, use the source name as the key:

```
dmf:
  doveadm:
    path: "/usr/bin/doveadm"
    source:
      default:
        ...
      POD_1:
        ...
```

Now, when the Worker gets a migration job for POD_1, it will use the POD_1 doveadm configuration, but if it gets a job for POD_2, then it will use the default doveadm configuration.

2.3 Manage the Application

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

Start example:

```
systemctl start dmf-worker
```

Stop example:

```
systemctl stop dmf-worker
```

Warning
The DMF Worker is very stateful in a number of ways. It is extremly important to only stop the application while it is processing migraiton jobs in critical situations.

The proper way to stop a DMF worker is:

2.3.1 1. Pause the Worker

By using the DMF Scheduler Admin REST API.

```
curl -X 'PATCH' \
  'https://scheduler:8443/dmf/admin/api/v2/targets/default/backends' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "default/worker1",
  "command": "PAUSE"
}'
```

2.3.2 2. Check and Confirm on no Running Jobs

Once you have confirmed that the Worker has no running jobs, you can stop it. This can be confirmed by using the Scheduler API to get the current status of the Worker. If the Worker is paused and has 0 threads, then it is not running any migration jobs.

```
curl -X 'GET' \
  'https://localhost:7443/dmf/admin/api/v2/targets/default/backends/default%2Fwoker1' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ='
```

```
{
  "name": "default/worker1",
  "target": "default",
  "initialState": 1,
  "maxThreads": 200,
  "status": 0,  <-- 0 is paused
  "threads": 0, <-- 0 threads means 0 jobs running
  "command": null,
  "updated": "2021-04-12T21:34:56.00Z"
}
```

2.3.3 3. Stop the worker either using the Admin API or service.

```
curl -X 'PATCH' \
  'https://scheduler:8443/dmf/admin/api/v2/targets/default/backends' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "default/worker1",
  "command": "STOP"
}'
```

The Worker will stop polling for jobs and the application will close.

3 Doveadm Features

By default, the DMF Worker will use the Doveadm Worker type to process migration jobs. This section lists all features, what they do, and how to configure them.

The below configuration examples assume that the key is under `dmf.doveadm.source.<sourceName>`. For instance, if the required configuration is to set `my-property` to `true`, then the following are equivalent:

```
...:
  my-property: true
---
dmf:
  doveadm:
    source:
      mySource:
        my-property: true
```

Info
A migration job will be successful unless something is misconfigured, a fatal unknown exception occurs, or the migration command fails after the max retries.

3.1 Analyze Log

The Worker will create a doveadm log analyzer which can be used for finding errors (some which could be automatically resolved) and mailbox statistics.

By setting to false, the following functionality will be lost:

- Find and resolve duplicate UIDs
- Find and resolve failed save due to timeout
- Collect errors for the job response
- Collect sync mail statistics

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        analyze-log: true
```

3.2 Move Duplicates

This feature requires analyze-log.

While rare, it is possible that the source mailbox contains multiple messages with the same UID causing a duplicate UID situation. This is detected by finding log messages containing "Expunged message reappeared in session". Any UIDs with this issue will be extracted and an attempt to fix them will be made. This error causes the migration command to fail. If there is a retry configured, the duplicate fix will be performed prior to the command retry.

Step:

1. The Worker will connect to the source host through an imap connection
2. Create a new mailbox named `lost+found-<originalFolder>` under the folder that the UID is within.
3. Copy the message from the original folder to the lost and found folder
4. Expunge the original mail

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        move-duplicates: true
```

If a duplicate message is found at any point during the migration (even if it is resolved), it will be included in the Migration details field in the `dsync` operation metadata as `invalid`.

Example:

```
{
  "dsync": {
    ...,
    "invalid": {
      "INBOX": [
      "uid1",
      "uid2"
      ]
    }
  }
}
```

3.3 IMAPC Inbox

Currently, this feature executes shell commands `cp` and `chown` on a users home mail path as well as delete folders that it creates through the Java API. Unfortunately, this requires DMF to be executed with a user with this permission level.

A hack to speed up delta syncs with pop3 uidl. It copies the local user's INBOX mailbox to:

- `<userHome>/imapc/.INBOX/.INBOX`

This is done after the pre migration command but before the migration command.

Steps:

1. Get the user's home path with: `doveadm user -f home <userUid>`

2. Get the user's INBOX path with: `doveadm mailbox path -u <userUid> INBOX`
3. Copy the INBOX to homePath/.INBOX.INBOX: `/usr/bin/cp -a homePath/.INBOX.INBOX`
4. Change the home path owner to vmail: `/bin/chown -R vmail:vmail homePath`

If configured to remove the created INBOX path, this is done after the migration command, regardless of outcome, but before the post/failure migration command.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        copy-imapc-inbox: true
        remove-imapc-path: true
```

3.4 Metacache

Executes metacache flush and/or metacache clean for the target user.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        ####
        # Metacache clean/flush settings. Replaces the legacy "flush-metacache" and "clean
            -metacache" settings,
        # however, if the legacy settings are set as 'true' then they will be used
        ####
        metacache:
          ####
          # Executes metacache flush for the local user.
          # This is executed before the post migration scripts.
          ####
          flush:
            on-success: false
            on-failure: false
          ####
          # Executes metacache clean for the local user.
          # This is executed before the post migration scripts.
          ####
          clean:
            on-success: false
            on-failure: false
```

It's also possible to call the HTTP API to flush metacache, for the Source user for instance. It can be configured for sync and cutover.

This requires that the director property is enabled and the http configuration is complete.

Configuration:

```
dmf:
  doveadm:
    source:
      <yoursource>:
        director:
          enabled: true
          flush-metacache:
            sync: true
            cutover: true
        # Notice that this is not under the director key
        http:
          url: "https://doveadmhttpapi/doveadm/v1"
          username: admin
```

```
14            password: verysecretpassword
```

3.5 Kick User

This requires that the director property is enabled and the http configuration is complete.

Makes the “directorKick” call to the Director HTTP API for the user. This is the first thing that is done. It can be configured for sync and cutover. Since the API for directorKick is asynchronous, you can make an obox wait request to wait for session termination. This is only possible if you are using obox.

If you are using the director-kick failure flag, but want to accept failures from the wait, then you can use the wait-accept-codes property. Provide a list of exit codes or -1 for all failures.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        director:
          enabled: true
          kick:
            wait: true
            timeout: 60s
            sync: true
            cutover: true
            wait-accept-codes: [65]
        # Notice that this is not under the director key
        http:
          url: "https://doveadmhttpapi/doveadm/v1"
          username: admin
          password: verysecretpassword
```

3.6 Use Director Sourcehost

This requires that the director property is enabled and the http configuration is complete.

During sync or cutover, the Worker will make a call to the Source Dovecot Director for the user to determine their backend. If there is not a backend defined for the user, it will randomly select one of the Director’s defined backends and move the user there. To use the found sourcehost in the migration command, use `%{mdb:directorSourcehost}`. You can also define a sourcehost for each backend and that command will be used instead of the sourcehost defined in DMF for the user.

Steps:

1. Get backend for user
2. If no backend found, get a random backend from the director, move the user there
3. Get the sourcehost definition from the DMF database if it exists

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        director:
          enabled: true
          use-sourcehost: true
        # Notice that this is not under the director key
        http:
          url: "https://doveadmhttpapi/doveadm/v1"
          username: admin
```

```
12            password: verysecretpassword
```

3.7 Move User

This requires that the director property is enabled and the http configuration is complete.

This will make a move user request to the configured doveadm HTTP API for the target user.

Steps:

1. Get Workers IP address
2. Get the list of Dovecot backends from the HTTP API directorStatus command
3. Verify that the Worker's IP is one of the backends
4. Move the target user to this backend with the HTTP API directorMove command

Configuration:

```
1  dmf:
2    doveadm:
3      sources:
4        <sourcename>:
5          director:
6            enabled: true
7            move-user: true
8          http:
9            url: "https://doveadmhttpapi/doveadm/v1"
10           username: admin
11           password: verysecretpassword
```

3.8 Fetch Container

Gets the `userdb_container` value in the configured container file for the value of imapcoptions defined for the user. If imapcoptions is not set for the user, but this is enabled, then it wont be used. The found value can be injected into migration commands with `%{mdb:container}`.

Configuration:

```
1  dmf:
2    doveadm:
3      sources:
4        <sourcename>:
5          fetch-container: true
6          container-file: /etc/dovecot/passwd.container
```

3.9 Parallel Writes Retry

If retries are configured and a failure has occurred with error message containing `failed: PUT .+ failed: Absolute request timeout expired`, then the following setting override will be added to the migration command prior to the retry:

- `-o plugin/obox_max_parallel_writes=1`

3.10 Mail Count

Properties to configure counting of mailbox messages and size.

3.10.1 Pre Mail Sync

Collect local mailbox stats after the pre migration command but before the migration command.

The result is logged as: `destination system statistics before sync. messages: {}, size: {} (bytes)`

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          pre: true
```

3.10.2 Mail Sync

This feature requires analyze-log.

Count the number of saved and expunged mails during the migration command. The results are stored with the job response as the sync saved and expunged message counts.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          sync: true
```

3.10.3 Post Mail Sync

Collect local mailbox stats after the successful migration command but before the post migration command. The results are stored with the job response as the target mailbox size and message count.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          post: true
```

3.10.4 Post Mail Sync Remote

Collect the remote mailbox stats after the successful migration command but before the post migration command. The results are stored with the job response as the origin mailbox size and message count.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          remote: true
```

There are four ways to configure how the remote count will be executed:

3.10.4.1 IMAPC Protocol This feature requires the imapc configuration or all imap connection properties included with the migration job.

If the user's Sourcehost definition does **not** use a `status command`, and the remote-protocol is set as imapc, then a default remote count command will be used.

This command is:

```
doveadm -f tab -o imapc_ssl=<imaps/no> -o imapc_host=<sourcehost> -o imapc_user=<sourceUid
    > -o imapc_password=<password> -o imapc_port=<port> -o mail=imapc: mailbox status -u <
    sourceUid> "messages vsize" INBOX/* INBOX *
```

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          remote-protocol: imapc
        imapc:
          host: host.with.imap
          port: 143
          prefix:
          master-user: admin
          master-password: verysecretpassword
```

3.10.4.2 Doveadm Protocol

This feature requires the doveadm configuration.

If the user's Sourcehost definition does **not** use a `status command`, and the remote-protocol is set as doveadm, then a default remote count command will be used.

This command is: `doveadm -f tab -o doveadm_password=<doveadm.password> mailbox status -u <sourceUid> -S <doveadm.host>:<doveadm.port> "messages vsize" INBOX/* INBOX *`

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          remote-protocol: doveadm
        doveadm:
          host: host.with.doveadm
          port: 24245
          password: verysecretpassword
```

3.10.4.3 HTTP Protocol

This protocol will use the doveadm HTTP API to count mailbox data.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          remote-protocol: http
          remote-http:
            ####
            # The default mailbox mask is ["INBOX", "INBOX/*", "*"]. If you want to
                specify another mask then
            # add each value in a comma delimited list (no spaces).
            ####
            mailbox-mask:
            ####
            # See doveadm http api mailboxStatus command for why this would be used.
            # Default is empty
            ####
            socket-path:
        http:
```

```
            url: "https://doveadmhttpapi/doveadm/v1"
            username: admin
            password: verysecretpassword
```

3.10.4.4 Status Command If the user's Sourcehost definition **does** include a `status command`, then that command will be used. The command must use the `mailbox status` doveadm subcommands as well as a tab formatter.

3.10.4.5 Configuration To override all other options, you can specify the remote count command using configuration.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        count:
          remote-command:
            name: doveadm
            options:
              - name: -f
                value: tab
              - name: -o
                value: "imapc_user=%{mdb:ruid}"
              - name: -o
                value: "imapc_password=%{conf:imapc_master_password}"
              - name: -o
                value: "imapc_host=%{mdb:sourcehost}"
              - name: -o
                value: "mail=imapc:"
            sub-command:
              name: mailbox status
              arguments:
                - "messages vsize"
                - "INBOX/*"
                - "INBOX"
                - "*"
              options:
                - name: -u
                  value: "%{mdb:uid}"
```

3.11 Migration Retry

Properties to configure retrying the migration command after a failure.

3.11.1 Max Retries

Max number of retries for doveadm sync errors which are non fatal. To override any specific error code, use code-max.

Override for max on the error code level.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        retry:
          max: 1
          code-max:
            75: 5
```

3.11.2 Retry Sleep

The amount of time in ms to sleep before retrying the migration command. To override any specific error code use code-sleep.

Override for sleep on the error code level.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        retry:
          sleep: 5000
          code-sleep:
            75: 15000
```

3.12 Commands

The Doveadm DMF Worker has the ability to execute various commands throughout the migration job. This includes the main migration command. The migration command must be a Doveadm-Sync command. If you are looking to do something else, then you likely do not want to use the Doveadm DMF Worker, and should look into the DMF Beyond Dovecot documentation. Any other command constraints will be listed in the following sections.

3.12.1 Property Injection

Commands can have properties injected into them. By default, you can do the following:

- MDB formatters are used like `%{mdb:X}`, where X can be: md5path, 2chrruid, container, uid, ruid, sourcehost, source, sourcepasswd, imapcoptions, email, sourceport, imapc_ssl, exclude, directorSourcehost
- Conf formatters are used like `%{conf:Y}`, where Y can be: imapc_host, imapc_master_password, imapc_master_user, imapc_prefix, imapc_port

You can also define custom properties that can be injected into commands. You will use the same `%{conf:Y}` formatter where Y will be defined under the property:

- `dmf.doveadm.source.<sourceName>.command.inject`

For example, if you define:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        command:
          inject:
            test: value
```

then you can have a command `doveadm -o setting=%{conf:test}` and the command would resolve to `doveadm -o setting=value`

The order of injection is:

1. custom inject properties
2. imapc properties - from config
3. user properties - from job

All commands support property injection.

3.12.2 Pre Mail Sync

This can be any shell command and it is executed prior to the migration command. An applicable example would be to specify a shell script that takes the sourceUid and locks the source mail account.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        command:
          sync:
            pre:
              name: echo
              arguments:
                - pre
                - sync
                - "%{mdb:uid}"
          cutover:
            pre:
              name: echo
              arguments:
                - pre
                - cutover
                - "%{mdb:uid}"
```

3.12.3 Mail Sync

Overrides the migration command. This must be a Doveadm-Sync command.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        command:
          sync:
            sync:
              name: doveadm
              ...
          cutover:
            sync:
              name: doveadm
              ...
```

3.12.4 Post Mail Sync

This can be any shell command and it is executed after a successful migration command. An applicable example would be to specify a shell script that takes the sourceUid or targetUid and changes a proxy status.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        command:
          sync:
            post:
              name: echo
              arguments:
```

```
                - post
                - sync
                - "%{mdb:uid}"
          cutover:
            post:
              name: echo
              arguments:
                - post
                - cutover
                - "%{mdb:uid}"
```

3.12.5 Mail Sync Failure

This can be any shell command and it is executed after a migration command failure. If retries are configured, this is only executed if the last retry is still a failure. An applicable example would be to specify a shell script that takes the sourceUid and unlocks the source mail account.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        command:
          sync:
            failure:
              name: echo
              arguments:
                - failure
                - sync
                - "%{mdb:uid}"
          cutover:
            failure:
              name: echo
              arguments:
                - failure
                - cutover
                - "%{mdb:uid}"
```

3.13 Configurable Failures

By default, the only things that will mark a migration as FAILURE, is a misconfiguration or a failed migration command. However, the other steps in the migration can be configured to mark the migration as FAILURE if they fail.

Note: This will also cause the post migration failure command to be executed. Currently it is only executed if the migration command fails.

This feature is useful when, for instance, you configure a post migration command to do some processing of the mail data, and if that processing fails, then the user should not be marked as migrated.

By default, the doveadm migration command is the only step that will mark a migration as failed if it fails. All other steps by default will not mark a migration as failed if they fail, however, they can be configured to do so.

Configuration:

```
dmf:
  doveadm:
    sources:
      <sourcename>:
        failure-flags:
          cutover:
            post-migration-command: true
```

Supported configuration keys: INCLUDE-SNIPPET-29: markdownInclude/2/snippet-29.txt

3.14 Order of Operations

The migration job order of operations (only if configured):

1. Move User
2. Pre Mail Sync Command
3. Copy IMAPC Inbox
4. Pre Mail Sync Count
5. Mail Sync(Migration) Command + Mail Sync Count
6. Remove IMAPC Inbox

Successful Migration Command:

1. Post Mail Sync Count
2. Post Mail Sync Remote Count
3. Flush Metacache
4. Clean Metacache
5. Post Mail Sync Command

Failed Migration Command:

1. Mail Sync Failure Command

3.15 Job Response Details

Each operation will provide a status in the Migration Job Response details field. This field is a JSON array with the results from the order of operations.

The operation will only be included if the feature supporting that operation is enabled. For instance, if there is not a Pre Mail Sync Command configured, then there will not be a “pre migration command” operation in the details.

Example:

```
[
  {
    "pre migration command": {
      "success": true,
      "command": "echo pre mail sync user1",
      "exitCode": 0,
      "errors": []
    }
  },
  {
    "copy imapc inbox": {
      "success": true,
      "errors": []
    }
  },
  {
    "count local mailbox pre sync": {
      "success": true,
      "command": "doveadm -f tab mailbox status -u user1 \"messages vsize\" INBOX/* INBOX
          *",
      "exitCode": 0,
      "errors": []
    }
  },
  {
    "dsync": {
      "success": true,
      "command": "doveadm -o imapc_host=host -o imapc_user=user1 -o imapc_password=<hidden
          > -o imapc_port=143 backup -R -u user1 imapc:",
      "exitCode": 0,
      "errors": [],
```

```
        "attempts": 1,
        "saved": {
          "INBOX": 50,
          "special": 25
        },
        "expunged": {
          "special": 5
        },
        "invalid": {
          "INBOX": [
            "uid1",
            "uid2"
          ]
        }
      }
    },
    {
      "remove imapc inbox": {
        "success": true,
        "errors": []
      }
    },
    {
      // This should only exist after a dsync success
      "count local mailbox post sync": {
        "success": true,
        "command": "doveadm -f tab mailbox status -u user1 \"messages vsize\" INBOX/* INBOX
            *",
        "exitCode": 0,
        "errors": []
      }
    },
    {
      // This should only exist after a dsync success
      "count remote mailbox": {
        "success": true,
        "command": "doveadm -f tab -o imapc_host=host -o imapc_user=user1 -o imapc_password
            =<hidden> -o imapc_port=143 -o mail=imapc: mailbox status -u user1 \"messages
            vsize\" INBOX/* INBOX *",
        "exitCode": 0,
        "errors": []
      }
    },
    {
      "flush user metacache": {
        "success": true,
        "command": "doveadm metacache flush -u user1",
        "exitCode": 0,
        "errors": []
      }
    },
    {
      "clean user metacache": {
        "success": false,
        "command": "doveadm metacache clean -u user1",
        "exitCode": 75,
        "errors": ["some error message"]
      }
    },
    {
      // This should only exist after a dsync success
      "post migration command": {
        "success": true,
        "command": "echo post mail sync user1",
        "exitCode": 0,
        "errors": []
      }
    },
    {
      // This should only exist after a failure
      "post migration failure command": {
        "success": true,
```

```
        "command": "echo post mail sync failure user1",
        "exitCode": 0,
        "errors": []
      }
    }
]
```

4 Worker Health

As part of the Micronaut framework, each Worker 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.

4.1 Without Authentication

```
curl https://worker:8443/health
```

Sample output:

```
{
  "status" : "UP"
}
```

4.2 With Authentication and Full Details

```
curl -u admin:secret https://worker:8443/health
```

Sample output:

```
{
  "name": "worker",
  "status": "UP",
  "details": {
    "jdbc": {
      "name": "worker",
      "status": "UP",
      "details": {
        "jdbc:mysql://dmf-db:3306/migration?createDatabaseIfNotExist=true": {
          "name": "worker",
          "status": "UP",
          "details": {
            "database": "MariaDB",
            "version": "10.5.4-MariaDB-1:10.5.4+maria~focal"
          }
        }
      }
    },
    "compositeDiscoveryClient()": {
      "name": "worker",
      "status": "UP"
    },
    "diskSpace": {
      "name": "worker",
      "status": "UP",
      "details": {
        "total": 126557421568,
        "free": 71800446976,
        "threshold": 10485760
      }
```

```
    },
    "service": {
      "name": "worker",
      "status": "UP"
    }
  }
}
```

4.3 Configuration

Individual health indicators can be turned off with configuration settings, which can be specified through modifying the `dmf-worker.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.

5 Worker Metrics

Each Worker 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.

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

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

5.1 List of Metrics

A list of metric names can be queried using

```
curl -u admin:secret https://worker: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",
      "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"
    ]
  }
```

5.2 Query a Metric

Querying a specific metric can be achieved as follows:

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

Sample output:

```
{
  "name": "process.uptime",
  "measurements": [
    {
      "statistic": "VALUE",
      "value": 43.324
    }
  ],
  "description": "The uptime of the Java virtual machine"
}
```

5.3 Prometheus Metrics API

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

```
curl -u admin:secret https://worker:8443/prometheus
```

A portion of the sample output:

```
# HELP hikaricp_connections_active Active connections
# TYPE hikaricp_connections_active gauge
hikaricp_connections_active{pool="HikariPool-1",} 0.0
# HELP jvm_buffer_memory_used_bytes An estimate of the memory that the Java virtual
```

```
    machine is using for this buffer pool
# TYPE jvm_buffer_memory_used_bytes gauge
jvm_buffer_memory_used_bytes{id="direct",} 3.35544376E8
jvm_buffer_memory_used_bytes{id="mapped",} 0.0
# HELP jvm_buffer_total_capacity_bytes An estimate of the total capacity of the buffers in
     this pool
# TYPE jvm_buffer_total_capacity_bytes gauge
jvm_buffer_total_capacity_bytes{id="direct",} 3.35544375E8
jvm_buffer_total_capacity_bytes{id="mapped",} 0.0
```

6 Beyond Dovecot

This documentation discusses the uses of DMF outside of Dovecot. It is not necessary to review this information for a standard DMF deployment.

Warning
If you have not read all other documentation, you should go back before proceeding.

As previously noted, while DMF stands for *Dovecot* Migration Framework, in reality, it is more like a *Mail* Migration Framework because nothing actually limits you to Dovecot. In fact, bare bones, it is just a job processing framework that allows you to plug in any Job Worker to process your job.

6.1 Job Workers

The DMF Worker deploys with three Job Workers out of the box.

The type of job worker is configured using the `dmf.worker.type` property.

6.1.1 Doveadm

The `doveadm` worker is the standard DMF job worker and it is explained in detail in the DMF Doveadm Features section.

```
dmf:
  worker:
    type: doveadm
```

6.1.2 Simulator

The `simulator` worker is used for testing both during development and deployment.

This worker does nothing but sleep for a random amount of time between 1 and 10 seconds inclusive - "simulating" the work.

```
dmf:
  worker:
    type: simulator
```

6.1.3 Command

The `command` worker is used to simply execute the migration command. It does nothing more.

The migration command is not converted in the same way that the `doveadm` worker does it. It does support property injection, however, commands will not be converted to doveadm format and passwords will **not** be hidden.

Warning
do not hard code or inject passwords into the migration command.

This worker allows you to execute any kind of command. You could write a shell command which performs the actual mail sync and configure the migration command to use it.

```
dmf:
  worker:
    type: command
```

6.2 Custom Migration

It's also possible to implement a custom Job Worker to perform the migration in a custom way that the existing workers cannot do and then plug it into DMF.

7 Shipped Version

7.1 Package open-xchange-dmf-worker

DMF Worker Dovecot Migration Framework Worker.
Version: 1.2.0-10
Type: Other

7.1.1 Installation

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

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

7.1.2 Configuration

For details, please see appendix A
/opt/open-xchange/dmf/worker/etc/dmf-worker.yml (page 36)

A Configuration Files

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

```
micronaut:
  # The Worker exposes web services for metrics.
  # SSL configuration
  # Required for production environments.
  # See https://docs.micronaut.io/latest/guide/index.html#https for details.
  ssl:
    enabled: true
#    key-store:
#      path: file:/opt/open-xchange/dmf/certs/keystore.p12
#      type: PKCS12
#      password:
    port: 8443
  application:
    name: worker
  metrics:
    enabled: true
    export:
      prometheus:
        enabled: true
        descriptions: true
        step: PT1M
  ####
  # Configure server thread pools.
```

```
  # See Micronaut doc: https://docs.micronaut.io/latest/guide/index.html#threadPools
  ####
  executors:
    ####
    # The pool where workers are executed.
    # The number of threads must be greater than dmf.worker.jobs.max or there
    # will be thread queuing which will cause jobs to wait.
    ####
    worker-executor:
      name: worker-executor
      # No more than the number of threads
      type: fixed
      number-of-threads: 100
    ####
    # The pool where non worker threads are executed. This is mainly related to log
    # processing.
    # The number of threads must be at least equal to worker-executor
    ####
    command-executor:
      name: command-executor
      # No more than the number of threads
      type: fixed
      number-of-threads: 100
  http:
    services:
      ####
      # Configure the HTTP connection properties for the doveadm HTTP APIs.
      # This configuration is shared for all defined doveadm HTTP configurations.
      # This is where SSL can be enabled and configured.
      ####
      doveadm:
        ssl: {}
  #           enabled: true
  #           trust-store:
  #             path: file:/opt/open-xchange/dmf/certs/doveadm.p12
  #             password:
  #             type: PKCS12
  security:
    enabled: true
    #ip-patterns:
    #  - 127.0.0.1
---
endpoints:
  all:
    enabled: true
    sensitive: true
  info:
    sourceCodeOrigin:
      enabled: true
      location: file:/opt/open-xchange/dmf/worker/share/SourceCodeOrigin.txt
---
# Set the basic auth username and password that can be used to reach
# any built in endpoint
http:
  admin:
    username:
    password:
---
# This will connect the worker to the DMF Migration database
# to get/update worker state and get/update migration jobs
datasources:
  default:
    # url should use createDatabaseIfNotExist=true if the database will not
    # already exist: https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-
        configuration-properties.html
    url: jdbc:mysql://localhost:3306/migration?createDatabaseIfNotExist=true
    username:
    password:
    dialect: MYSQL
    driverClassName: org.mariadb.jdbc.Driver
---
dmf:
```

```
  worker:
    crypto:
      ######################
      # Enable user password encryption. If commands will use user passwords then they have
      # to be decrypted which requires use of the private key that they were encrypted with.
      # Provide each key that has been used to encrypt user passwords here. The keyName is the
      # name of the key that was configured in the DMF Scheduler to encrypt a password, and also
      # identifies the key in the KeyStore if used.
      # The file is a fully qualified path to either a plain file that contains the
      # encoded bytes of the symmetric AES-256 key, or a Java KeyStore file. To use a
      # Keystore, prefix the file path with "keystore:". The secret is required if a JKS is used.
      # You can list any number of storage keys here.
      ######################
      storageKeys:
      #  keyName:
      #    file:
      #    secret:
    sources: default
    state:
      # The workers initial state. When using the database state type, if the worker has been
      # previously started then it will use the initial state in the database to determine how to start.
      # Current options:
      #    1. STARTED
      #    2. STOPPED
      initial: STARTED
      # The way the worker will collect and store its state.
      # Options:
      #    1. database - uses the DMF migration database
      #    2. config - uses this config to get the initial state and max jobs
      type: database
    jobs:
      cache:
        size: 1000
        refresh: 5s
      # Max number of jobs to queue at a time. When using the database state type, if the Worker has previously
      # been started, then this value will be ignored and the maxThreads defined for the worker will be used.
      #
      # Keep in mind that if the executor pool is smaller than this number then
      # some jobs will be waiting in the executor pool.
      max: 10
      termination:
        # The max time to wait in ms for jobs to complete on application termination before willfully shutting down.
        # Shutting down the application while jobs are running can leave DMF in a corrupt state if jobs do not complete.
        # While jobs should shutdown quickly when abort is called, this should be set to a high value.
        timeout: 30000
      poller:
        # The job poller type. Current options:
        #    1. database - collects jobs from the DMF migration database
        #    2. simulator - creates simulated jobs meant for testing other parts of DMF
        type: database
        simulator:
          max-jobs: 5
          max-wait-ms: 0
    identity:
      # Identifies a group of DMF workers. This should be the same for all DMF workers that service a particular Dovecot platform.
      # This should match a "target" that has been registered with the DMF REST API.
      target: default
      # Identifies this worker within a group of DMF workers. This should be unique within a target.
```

```
      memberid:
    # The worker type. Current options:
    #    1. doveadm - uses doveadm to complete the migration job
    #    2. command - executes the migration command as is
    #    3. simulator - does not actually run any commands meant for testing the Worker roundtrip
    type: doveadm
    command:
      logging:
        # Where to redirect standard out and error when running commands
        # Options:
        #   1. file: logs to a file (see file properties)
        #   2. logger: logs to the logger
        #   3. off: silences logging
        type: file
        file:
          # The log file name format. You can use the following specifiers:
          #   * source: the source name
          #   * user: the target uid
          #   * phase: the migration phase (pre-sync, cutover)
          #   * job: the job id
          #   * date: the YYYY-MM-DD date
          format: "%(source)-%(user)-user-migration.log"
          location: /app
          delete-on-success: true
  # Define doveadm properties
  doveadm:
    # The path to the doveadm command
    path: "/usr/bin/doveadm"
    # DMF Source specific properties should go under the source name key
    source:
      # If a source is not defined then it will use the default if it exists
      default:
        ####
        # The worker will create a log analyzer which can be used for finding errors (some which
        # could be automatically resolved) and mailbox statistics.
        # By setting to false, the following functionality will be lost:
        #   - Find and resolve duplicate UIDs
        #   - Find and resolve failed save due to timeout
        #   - Collect errors for the job response
        #   - Collect sync mail statistics
        ####
        analyze-log: true
        ####
        # If the migration command fails due to mails with duplicate UIDs, then it will attempt
        # to connect to the source IMAP to move those duplicate mails into a folder called
        # lost+found-<originalFolderName>. If retry is enabled then the command will be rerun.
        # Duplicates can only be found if analyze-log is enabled.
        ####
        move-duplicates: false
        ####
        # A hack to speed up delta syncs with pop3 uild. It copies the local user's INBOX
        # mailbox to <userHome>/imapc/.INBOX/.INBOX.
        # This is done after the pre migration script but before the migration command
        ####
        copy-imapc-inbox: false
        ####
        # If using copy-imapc-inbox, this will remove the created imapc folder for the local user.
        # This is done after the migration command, regardless of outcome, but before
        # any post migration script
        ####
        remove-imapc-path: false
        ####
        # Metacache clean/flush settings. Replaces the legacy "flush-metacache" and "clean-metacache" settings,
        # however, if the legacy settings are set as 'true' then they will be used
        ####
        metacache:
```

```
    ####
    # Executes metacache flush for the local user.
    # This is executed before the post migration scripts.
    ####
    flush:
      on-success: false
      on-failure: false
    ####
    # Executes metacache clean for the local user.
    # This is executed before the post migration scripts.
    ####
    clean:
      on-success: false
      on-failure: false
####
# Gets the userdb_container value in the file /etc/dovecot/passwd.container for the value of
# imapcoptions defined for the user. If imapcoptions is not set for the user, but this is enabled
# then it wont be used. The found value can be injected into migration commands with %{mdb:container}
####
fetch-container: false
####
# The dovecot passwd.container file path that is used when fetch-container is enabled.
####
container-file: /etc/dovecot/passwd.container
####
# Utilize the DoveAdm HTTP API
# The name "director" is a legacy term and was not renamed for ease of change.
# This configuration set can be used for both director and cluster based architectures.
####
director:
  ####
  # When enabled, DMF will get the Dovecot backend on the Source from the
  # doveadm HTTP API.
  #
  # For director architecture: DMF first attempts to get the
  # user's assigned backend, but if the response does not include a backend, then
  # it will randomly select one of the director's backends and execute a
  # directorMove command for the user with that randomly selected backend.
  #
  # DMF then checks the sourcehosts storage to check if there are commands
  # configured for the new sourcehost. If there are, it will use them, if not
  # then it will use the sourcehost that was originally tied to the user.
  # It may be required to define each backend as a sourcehost though as the
  # sourcehost name provided by the api may not be known, thus you can
  # define the actual sourcehost name in the migration command. Alternatively, if
  # you can use the sourcehost name returned by the api, then you can use the
  # mdb formatter "directorSourcehost" in the migration command to use this sourcehost
  # instead of the one defined for the user.
  ####
  use-sourcehost: false
  ####
  # Use the doveadm HTTP API to kick the user with the directorKick or clusterKick command.
  # Set to true for whichever phase(s) that DMF should perform this.
  # This is the very first operation that DMF performs.
  # This call is asynchronous on the http side, so when obox is used, there is
  # an optional wait call that can be made. The timeout can be configured
  # by using a Duration (ex: 10s, 1m) with a default of 60s.
  # If you are using the director-kick failure flag, but want to accept failures
  # from the wait, then you can use the wait-accept-codes property. Provide
  # a list of exit codes or -1 for all failures
  ####
  kick:
    sync: false
    cutover: false
    wait: false
```

```
284             timeout: 60s
285             wait-accept-codes: []
286           ####
287           # Use the doveadm HTTP API to flush the user's metacache. This is usually
                  important prior to cutover
288           # when there is a chance that the migration connection might not be against the
                  same backend
289           # that the user was last assigned.
290           ####
291           flush-metacache:
292             sync: false
293             cutover: false
294           ####
295           # Inform the director layer via the HTTP API about being managed by this backend
                  .
296           ####
297           move-user: false
298         ####
299         # Properties to configure counting of mailbox messages and size
300         ####
301         count:
302           ####
303           # Collect local mailbox stats after the pre migration script but before the
304           # migration command. The results are simply logged.
305           ####
306           pre: true
307           ####
308           # Count the number of saved and expunged mails during the migration command.
309           # The results are stored with the job response as the sync saved and expunged
310           # message counts.
311           # Sync stats can only be collected if analyze-log is enabled.
312           ####
313           sync: true
314           ####
315           # Collect local mailbox stats after the successful migration command but
316           # before the post migration script. The results are stored with the job
317           # response as the target mailbox size and message count.
318           ####
319           post: true
320           ####
321           # Collect the remote mailbox stats after the successful migration command but
322           # before the post migration script. The results are stored with the job
323           # response as the origin mailbox size and message count.
324           ####
325           remote: true
326           ####
327           # The protocol to use when executing the remote count if the default command
                  will be used.
328           # Options:
329           #   - doveadm: doveadm -o doveadm_password=<doveadm.password> mailbox status -u
                  <ruid>
330           #              -S <doveadm.host>:<doveadm.port> "messages vsize" INBOX/* INBOX *
331           #   - imapc: doveadm -o imapc_ssl=<imaps/no> -o imapc_host=<sourcehost> -o
                  imapc_user=<ruid>
332           #            -o imapc_password=<password> -o imapc_port=<port> -o mail=imapc:
333           #             mailbox status -u <uid> "messages vsize" INBOX/* INBOX *
334           #   - http: uses the configured doveadm http api to execute the mailboxStatus
                  command and collect the messages
335           #           and vsize of each mailbox matching the mailboxMask. This requires
                  the worker.doveadm.source.<name>.http
336           #           configuration. Additionally, you may use the remote-http properties
                  to override the default
337           #           mailboxMask or set a socketPath.
338           ####
339           remote-protocol: imapc
340           ####
341           # Only used for remote-protocol 'http'.
342           ####
343           remote-http:
344             ####
345             # The default mailbox mask is ["INBOX", "INBOX/*", "*"]. If you want to
                    specify another mask then
```

```
      # add each value in a comma delimited list (no spaces).
      ####
      mailbox-mask:
      ####
      # See doveadm http api mailboxStatus command for why this would be used.
      # Default is empty
      ####
      socket-path:
    ####
    # When enabled, the remote mailbox count by default will execute doveadm mailbox status
    # using the doveadm remote protocol. This can be overridden by defining the command
    # with the sourcehost definition. It can also be overridden here which will take precedence.
    # The format here is the common command format. However, there are restrictions to this.
    # The root command "doveadm" and sub command "mailbox status" with fields "messages vsize" will
    # always be used and without debug or verbosity, and will use a tab formatter. Do not change these
    # or it will corrupt the counting. The only important thing to include are setting overrides and
    # mailbox status mailbox patterns.
    # Format:
    #   name: doveadm
    #   options:
    #     -
    #       name: -o
    #       value: "imapc_user=%{mdb:ruid}"
    #     -
    #       name: -o
    #       value: "imapc_password=%{conf:imapc_master_password}"
    #     -
    #       name: -o
    #       value: "imapc_host=%{mdb:sourcehost}"
    #     -
    #       name: -o
    #       value: "mail=imapc:"
    #  sub-command:
    #    name: mailbox status
    #    arguments:
    #      - "messages vsize"
    #      - "INBOX/*"
    #      - "INBOX"
    #      - "*"
    #    options:
    #      -
    #        name: -u
    #        value: "%{mdb:uid}"
    ####
    remote-command:
  ####
  # Properties to configure retrying the migration command after a failure.
  ####
  retry:
    ####
    # Max number of retries for doveadm sync errors which are non fatal.
    # To override any specific error code, use code-max.
    ####
    max: 1
    ####
    # The amount of time in ms to sleep before retrying the migration command.
    # To override any specific error code, use code-sleep.
    ####
    sleep: 5000
    ####
    # Override for max on the error code level. Format is:
    # code-max:
    #   <code>: <num_retries>
    ####
    code-max:
```

```
411             134: 2
412             75: 4
413           ####
414           # Override for sleep on the error code level. Format is:
415           # code-sleep:
416           #   <code>: <time_in_ms>
417           ####
418           code-sleep:
419             75: 15000
420           ####
421           # By default, retries will not run in debug mode.
422           # To enable debug mode for retries, set the flag below to true:
423           #   <debug-mode>: <true>
424           # NOTE: Retry with debug is only used when storage timeout is detected
425           # on previous attempt.
426           ####
427           debug-mode: false
428         ####
429         # Configure a doveadm http api client to the Source
430         ####
431         #http:
432         ## Possible options:
433         ##  * director - uses the director architecture
434         ##  * cluster - uses the cluster architecture
435         #  type: director
436         #  # The full doveadm http api url. Ex: "https://doveadmhttpapi/doveadm/v1"
437         #  url:
438         #  username:
439         #  password:
440         #  # only used with the director type
441         #  # used when use-only-tagged=true
442         #  director-tag:
443         #  # only used with the director type
444         #  # means that the backend status tag must match the tag defined in director-tag
445         #  use-only-tagged:
446         ####
447         ####
448         # Properties for running doveadm commands on the Source using
449         # the doveadm protocol
450         ####
451         #doveadm:
452         #  password:
453         #  host:
454         #  port: 24245
455         ####
456         # IMAP connection properties use to inject into commands or use
457         # for connecting to a users mailbox. You may specify some or all properties.
458         # For IMAP connections, these will override values defined with the job.
459         ####
460         imapc:
461           host:
462           port:
463           prefix:
464           master-user:
465           master-password:
466           ####
467           # Specify java mail api properties here
468           ####
469           session-properties:
470         ####
471         # Properties to configure commands accessible by the worker to be executed
472         # during the specific migration phases.
473         #
474         # pre: executed prior to the migration command and some other configurable
                operations.
475         # sync: Overrides the migration command.
476         # post: executed after a successful migration command and some other configurable
                operations.
477         # failure: executed after a migration command failure. When retries are enabled,
                this is
478         # only executed if the last retry is a failure, otherwise post is executed.
479         #
```

```
480         # Commands can have properties injected into them:
481         #  MDB formatters are used like %{mdb:X}, where X can be: md5path, 2chrruid,
                container,
482         #    uid, ruid, sourcehost, source, sourcepasswd, imapcoptions, email, sourceport,
                 imapc_ssl,
483         #    exclude, directorSourcehost
484         #  Conf formatters are used like %(conf:Y}, where Y can be any value defined in
                the "inject" config
485         #    or: imapc_host, imapc_master_password, imapc_master_user, imapc_prefix,
                imapc_port
486         ####
487         command:
488           ####
489           # Custom properties that can be defined and injected into the migration or
490           # other definable commands. For example, if you define:
491           #   inject:
492           #     test: value
493           # then you can have a command "doveadm backup -o setting=%{conf:test}" and the
                command
494           # would resolve to "doveadm backup -o setting=value"
495           # The order of injection is:
496           #   1. inject properties
497           #   2. user properties - from job
498           #   3. imapc properties - from config
499           ####
500           inject:
501           ####
502           # Overrides the migration command during pre-sync.
503           #
504           # Format + Example:
505           # sync:
506           #   sync:
507           #     name: doveadm
508           #     options:
509           #       -
510           #         name: -o
511           #         value: "imapc_user=%{mdb:ruid}"
512           #       -
513           #         name: -o
514           #         value: "imapc_password=%{conf:imapc_master_password}"
515           #       -
516           #         name: -o
517           #         value: "imapc_host=%{mdb:sourcehost}"
518           #     sub-command:
519           #       name: backup
520           #       flags: -R
521           #       arguments: "imapc:"
522           #       options:
523           #         -
524           #           name: -u
525           #           value: "%{mdb:uid}"
526           ####
527           sync:
528             ####
529             # Example:
530             # pre:
531             #   name: echo
532             #   arguments: "%{mdb:uid}"
533             ####
534             pre:
535             sync:
536             post:
537             failure:
538           ####
539           # Overrides the migration command during cutover. See sync for format.
540           ####
541           cutover:
542             pre:
543             sync:
544             post:
545             failure:
546         ####
```

```
547         # By default, the doveadm migration command is the only step that will mark a
                migration
548         # as failed if it fails. All other steps by default will not mark a migration as
                failed
549         # if they fail, however, they can be configured to do so.
550         # Note: this will cause the post migration failure command to be executed
551         #
552         # Supported steps:
553         #     director-kick-user
554         #     director-flush-metacache
555         #     director-use-sourcehost
556         #     director-move-user
557         #     pre-migration-command
558         #     post-migration-command
559         #     copy-imapc-inbox
560         #     remove-imapc-inbox
561         #     count-local-pre
562         #     count-local-post
563         #     count-remote
564         #     flush-metacache
565         #     clean-metacache
566         #
567         # Use the step name as key and true as value to enable failure
568         # Example that will marek the migration as failed if the pre-migration-command
                step fails:
569         #   failure-flags:
570         #     pre:
571         #       pre-migration-command: true
572         ####
573         failure-flags:
574           ####
575           # Failure flags for pre-sync phase
576           ####
577           pre:
578           ####
579           # Failure flags for cutover phase
580           ####
581           cutover:
```