Dovecot Migration Framework Worker 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**
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 1.0.0 Feature Delivery

1.3 Install Package Repository

This delivery is part of a restricted preview software repository:

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

1.4 Build Dependencies

This delivery was build with following dependencies:

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

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:

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

```
1  endpoints:
2    info:
3      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.

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

```
1  datasources:
2    default:
3      url: jdbc:mysql://dmf-db:3306/migration
4      username: worker
5      password: verysecretpassword
6      dialect: MYSQL
7      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 important thing is the `number-of-threads`. The default is 100. If you plan to allow the Worker to run more than 100 jobs in parallel, then you should adjust this number. For instance, if you plan to allow your Workers to process 400 jobs in parallel, then you should set this value to at least 400 for both the `worker-executor` and `command-executor` executor pools.

```
1   micronaut:
2     executors:
3       worker-executor:
4         name: worker-executor
5         type: fixed
6         number-of-threads: 400
7       command-executor:
8         name: command-executor
9         type: fixed
10        number-of-threads: 400
```

ℹ️ Info

The executor pool max threads is not the property that the Worker uses to determine the max number of jobs it will try to process in parallel. It is only a way to configure compute resources for the DMF application. This means that you should set these values to the max number of jobs that you will ever configure the Worker to process in parallel. It is not possible to change this number while the application is running. However, you can inform the Worker to attempt to process more or less jobs during runtime by using the Scheduler `backends` REST API and modifing the Worker's max threads.

2.2.5   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.

```
1   dmf:
2     worker:
3       identity:
4         target: cloud
5         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`:

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

2.2.6 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:

```
1  dmf:
2    worker:
3      sources:
4        - "POD_1"
5        - "POD_2"
```

2.2.7 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.

```
1  dmf:
2    worker:
3      state:
4        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.

```
1  curl -X 'PATCH' \
2    'https://worker:8443/dmf/admin/api/v2/targets/cloud/backends' \
3    -H 'accept: application/json' \
4    -H 'Authorization: Basic YWRtaW46cGFFzc3dvcmQ=' \
5    -H 'Content-Type: application/json' \
6    -d '{
7    "name": "cloud/1",
8    "initialState": 1
9  }'
```

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

2.2.8 Configure Job Settings

2.2.8.1 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.

```
1  dmf:
2    worker:
3      jobs:
4        cache:
5          size: 400
6          referesh: 1m
```

2.2.8.2 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.

```
1  dmf:
2    worker:
3      jobs:
4        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.

ℹ 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.

```
1  curl -X 'PATCH' \
2    'https://worker:8443/dmf/admin/api/v2/targets/cloud/backends' \
3    -H 'accept: application/json' \
4    -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
5    -H 'Content-Type: application/json' \
6    -d '{
7    "name": "cloud/1",
8    "maxThreads": 120
9  }'
```

2.2.9   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

```
1  dmf:
2    worker:
3      command:
4        logging:
5          type: file
6          file:
7            # The log file name format. You can use the following specifiers:
8            #   * source: the source name
9            #   * user: the target uid
10           #   * phase: the migration phase (pre-sync, cutover)
11           #   * job: the job id
12           #   * date: the YYYY-MM-DD date
13           format: "%(source)-%(user)-user-migration.log"
14           location: /var/log/dmf
15           delete-on-success: false
```

2.2.10   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:

```
1  dmf:
2    doveadm:
3      path: "/usr/bin/doveadm"
4      source:
5        ...
```

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:

```
1  dmf:
2    doveadm:
3      path: "/usr/bin/doveadm"
4      source:
5        default:
6          ...
7        POD_1:
8          ...
```

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:

```
1  systemctl start dmf-worker
```

Stop example:

```
1  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.

```
1  curl -X 'PATCH' \
2    'https://scheduler:8443/dmf/admin/api/v2/targets/default/backends' \
3    -H 'accept: application/json' \
4    -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
5    -H 'Content-Type: application/json' \
6    -d '{
7    "name": "default/worker1",
8    "command": "PAUSE"
9  }'
```

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.

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

```
1  {
2     "name": "default/worker1",
3     "target": "default",
4     "initialState": 1,
5     "maxThreads": 200,
6     "status": 0,  <-- 0 is paused
7     "threads": 0, <-- 0 threads means 0 jobs running
8     "command": null,
9     "updated": "2021-04-12T21:34:56.00Z"
10 }
```

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

```
1  curl -X 'PATCH' \
2     'https://scheduler:8443/dmf/admin/api/v2/targets/default/backends' \
3     -H 'accept: application/json' \
4     -H 'Authorization: Basic YWRtaW46cGFzc3dvcmQ=' \
5     -H 'Content-Type: application/json' \
6     -d '{
7     "name": "default/worker1",
8     "command": "STOP"
9  }'
```

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:

```
1  ...:
2     my-property: true
3  ---
4  dmf:
5     doveadm:
6       source:
7         mySource:
8           my-property: true
```

ℹ️ **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:

```
1  ...:
2    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:

```
1  ...:
2    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:

```
1   {
2     "dsync": {
3       ...,
4       "invalid": {
5         "INBOX": [
6         "uid1",
7         "uid2"
8         ]
9       }
10    }
11  }
```

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:

```
1  ...:
2    copy-imapc-inbox: true
3    remove-imapc-path: true
```

3.4   Metacache

Executes metacache flush and/or metacache clean for the target user. This is only done after a successful migration command but before the post migration command.

Configuration:

```
1  ...:
2    flush-metacache: true
3    clean-metacache: true
```

3.5   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  micronaut:
2    http:
3      services:
4        # Configure the HTTP client connection properties for the doveadm HTTP API
5        # https://docs.micronaut.io/latest/guide/configurationreference.html#io.micronaut.
              http.client.ServiceHttpClientConfiguration
6        doveadm:
7          ssl:
8            enabled: true
9            trust-store:
10             path: file:/opt/open-xchange/dmf/certs/doveadm.p12
11             password: verysecretpassword
12             type: PKCS12
13   ...:
14     director:
15       enabled: true
16       move-user: true
```

```
17    http:
18      url: "https://doveadmhttpapi"
19      username: admin
20      password: verysecretpassword
21      director-tag: aa
22      use-only-tagged: false
```

3.6   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  ...:
2    fetch-container: true
3    container-file: /etc/dovecot/passwd.container
```

3.7   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`

3.8   Mail Count

Properties to configure counting of mailbox messages and size.

3.8.1   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:

```
1  ...:
2    count:
3      pre: true
```

3.8.2   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:

```
1  ...:
2    count:
3      sync: true
```

3.8.3   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:

```
1   ...:
2     post:
3       pre: true
```

3.8.4   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:

```
1   ...:
2     remote:
3       pre: true
```

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

3.8.4.1   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:

```
1   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:

```
1   ...:
2     count:
3       remote-protocol: imapc
4     imapc:
5       host: host.with.imap
6       port: 143
7       prefix:
8       master-user: admin
9       master-password: verysecretpassword
```

3.8.4.2   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:

```
1   ...:
```

```
2    count:
3      remote-protocol: doveadm
4    doveadm:
5      host: host.with.doveadm
6      port: 24245
7      password: verysecretpassword
```

3.8.4.3 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.

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

Configuration:

```
1    ...:
2      count:
3        remote-command:
4          name: doveadm
5          options:
6            - name: -f
7              value: tab
8            - name: -o
9              value: "imapc_user=%{mdb:ruid}"
10           - name: -o
11             value: "imapc_password=%{conf:imapc_master_password}"
12           - name: -o
13             value: "imapc_host=%{mdb:sourcehost}"
14           - name: -o
15             value: "mail=imapc:"
16         sub-command:
17           name: mailbox status
18           arguments:
19             - "messages vsize"
20             - "INBOX/*"
21             - "INBOX"
22             - "*"
23           options:
24             - name: -u
25               value: "%{mdb:uid}"
```

3.9 Migration Retry

Properties to configure retrying the migration command after a failure.

3.9.1 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:

```
1    ...:
2      retry:
3        max: 1
4        code-max:
5          75: 5
```

3.9.2 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:

```
1  ...:
2    retry:
3      sleep: 5000
4      code-sleep:
5        75: 15000
```

3.10 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.

3.10.1 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
- 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:

```
1  ...:
2    command:
3      inject:
4        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.

3.10.2 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:

```
1  ...:
2    sync:
3      pre:
4        name: echo
5        arguments:
6          - pre
7          - sync
8          - "%{mdb:uid}"
9      cutover:
10     pre:
11       name: echo
12       arguments:
13         - pre
14         - cutover
15         - "%{mdb:uid}"
```

3.10.3 Mail Sync

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

Configuration:

```
1  ...:
2    sync:
3      sync:
4        name: doveadm
5        ...
6    cutover:
7      sync:
8        name: doveadm
9        ...
```

3.10.4 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:

```
1  ...:
2    sync:
3      pre:
4        name: echo
5        arguments:
6          - post
7          - sync
8          - "%{mdb:uid}"
9      cutover:
10     pre:
11       name: echo
12       arguments:
13         - post
14         - cutover
15         - "%{mdb:uid}"
```

3.10.5 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:

```
1  ...:
2    sync:
3      pre:
4        name: echo
5        arguments:
6           - failure
7           - sync
8           - "%{mdb:uid}"
9    cutover:
10     pre:
11       name: echo
12       arguments:
13          - failure
14          - cutover
15          - "%{mdb:uid}"
```

3.11   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

3.12   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:

```
1  [
2    {
3      "pre migration command": {
4        "success": true,
5        "command": "echo pre mail sync user1",
6        "exitCode": 0,
7        "errors": []
8      }
9    },
10   {
11     "copy imapc inbox": {
12       "success": true,
13       "errors": []
14     }
```

```
15        },
16        {
17          "count local mailbox pre sync": {
18            "success": true,
19            "command": "doveadm -f tab mailbox status -u user1 \"messages vsize\" INBOX/* INBOX
                  *",
20            "exitCode": 0,
21            "errors": []
22          }
23        },
24        {
25          "dsync": {
26            "success": true,
27            "command": "doveadm -o imapc_host=host -o imapc_user=user1 -o imapc_password=<hidden
                  > -o imapc_port=143 backup -R -u user1 imapc:",
28            "exitCode": 0,
29            "errors": [],
30            "attempts": 1,
31            "saved": {
32              "INBOX": 50,
33              "special": 25
34            },
35            "expunged": {
36              "special": 5
37            },
38            "invalid": {
39              "INBOX": [
40                "uid1",
41                "uid2"
42              ]
43            }
44          }
45        },
46        {
47          "remove imapc inbox": {
48            "success": true,
49            "errors": []
50          }
51        },
52        {
53          // This should only exist after a dsync success
54          "count local mailbox post sync": {
55            "success": true,
56            "command": "doveadm -f tab mailbox status -u user1 \"messages vsize\" INBOX/* INBOX
                  *",
57            "exitCode": 0,
58            "errors": []
59          }
60        },
61        {
62          // This should only exist after a dsync success
63          "count remote mailbox": {
64            "success": true,
65            "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 *",
66            "exitCode": 0,
67            "errors": []
68          }
69        },
70        {
71          // This should only exist after a dsync success
72          "flush user metacache": {
73            "success": true,
74            "command": "doveadm metacache flush -u user1",
75            "exitCode": 0,
76            "errors": []
77          }
78        },
79        {
80          // This should only exist after a dsync success
81          "clean user metacache": {
```

```
82        "success": false,
83        "command": "doveadm metacache clean -u user1",
84        "exitCode": 75,
85        "errors": ["some error message"]
86      }
87    },
88    {
89      // This should only exist after a dsync success
90      "post migration command": {
91        "success": true,
92        "command": "echo post mail sync user1",
93        "exitCode": 0,
94        "errors": []
95      }
96    },
97    {
98      // This should only exist after a failure
99      "post migration failure command": {
100       "success": true,
101       "command": "echo post mail sync failure user1",
102       "exitCode": 0,
103       "errors": []
104     }
105   }
106 ]
```

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

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

Sample output:

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

4.2 With Authentication and Full Details

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

Sample output:

```
1  {
2    "name": "worker",
3    "status": "UP",
4    "details": {
5      "jdbc": {
6        "name": "worker",
7        "status": "UP",
8        "details": {
9          "jdbc:mysql://dmf-db:3306/migration?createDatabaseIfNotExist=true": {
10           "name": "worker",
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": "worker",
21      "status": "UP"
22    },
23    "diskSpace": {
24      "name": "worker",
25      "status": "UP",
26      "details": {
27        "total": 126557421568,
28        "free": 71800446976,
29        "threshold": 10485760
30      }
31    },
32    "service": {
33      "name": "worker",
34      "status": "UP"
35    }
36   }
37 }
```

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

```
1  curl -u admin:secret https://worker:8443/metrics
```

Sample output:

```
1   {
2     "names": [
3       "executor",
4       "executor.active",
5       "executor.completed",
6       "executor.pool.core",
7       "executor.pool.max",
8       "executor.pool.size",
9       "executor.queue.remaining",
10      "executor.queued",
11      "hikaricp.connections",
12      "hikaricp.connections.acquire",
13      "hikaricp.connections.active",
14      "hikaricp.connections.creation",
15      "hikaricp.connections.idle",
16      "hikaricp.connections.max",
17      "hikaricp.connections.min",
18      "hikaricp.connections.pending",
19      "hikaricp.connections.timeout",
20      "hikaricp.connections.usage",
21      "jvm.buffer.count",
22      "jvm.buffer.memory.used",
23      "jvm.buffer.total.capacity",
24      "jvm.classes.loaded",
25      "jvm.classes.unloaded",
26      "jvm.gc.live.data.size",
27      "jvm.gc.max.data.size",
28      "jvm.gc.memory.allocated",
29      "jvm.gc.memory.promoted",
30      "jvm.gc.pause",
31      "jvm.memory.committed",
32      "jvm.memory.max",
33      "jvm.memory.used",
34      "jvm.threads.daemon",
35      "jvm.threads.live",
36      "jvm.threads.peak",
37      "jvm.threads.states",
38      "logback.events",
39      "process.cpu.usage",
40      "process.files.max",
41      "process.files.open",
42      "process.start.time",
43      "process.uptime",
44      "system.cpu.count",
45      "system.cpu.usage",
46      "system.load.average.1m"
47    ]
48  }
```

5.2   Query a Metric

Querying a specific metric can be achieved as follows:

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

Sample output:

```
1   {
2     "name": "process.uptime",
3     "measurements": [
4       {
5         "statistic": "VALUE",
6         "value": 43.324
7       }
8     ],
9     "description": "The uptime of the Java virtual machine"
10  }
```

5.3 Prometheus Metrics API

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

```
1  curl -u admin:secret https://worker: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
```

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.

```
1  dmf:
2    worker:
3      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.

```
1  dmf:
```

```
2    worker:
3      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.

```
1  dmf:
2    worker:
3      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 Packages and Version

7.1 Package open-xchange-dmf-worker

DMF Worker Dovecot Migration Framework Worker.
Version: 1.0.0-5
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 30)

A Configuration Files

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

```
1  micronaut:
2    # The Worker exposes web services for metrics.
3    # SSL configuration
4    # Required for production environments.
5    # See https://docs.micronaut.io/latest/guide/index.html#https for details.
6    ssl:
```

```
 7          enabled: true
 8        key-store:
 9          path: file:/opt/open-xchange/dmf/certs/keystore.p12
10          type: PKCS12
11          password:
12        port: 8443
13      application:
14        name: worker
15      metrics:
16        enabled: true
17        export:
18          prometheus:
19            enabled: true
20            descriptions: true
21            step: PT1M
22      ####
23      # Configure server thread pools.
24      # See Micronaut doc: https://docs.micronaut.io/latest/guide/index.html#threadPools
25      ####
26      executors:
27        ####
28        # The pool where workers are executed.
29        # The number of threads must be greater than dmf.worker.jobs.max or there
30        # will be thread queuing which will cause jobs to wait.
31        ####
32        worker-executor:
33          name: worker-executor
34          # No more than the number of threads
35          type: fixed
36          number-of-threads: 100
37        ####
38        # The pool where non worker threads are executed. This is mainly related to log
39        # processing.
40        # The number of threads must be at least equal to worker-executor
41        ####
42        command-executor:
43          name: command-executor
44          # No more than the number of threads
45          type: fixed
46          number-of-threads: 100
47      http:
48        services:
49          ####
50          # Configure the HTTP connection properties for the doveadm HTTP APIs.
51          # This configuration is shared for all defined doveadm HTTP configurations.
52          # This is where SSL can be enabled and configured.
53          ####
54          doveadm:
55    #         ssl:
56    #           enabled: true
57    #           trust-store:
58    #             path: file:/opt/open-xchange/dmf/certs/doveadm.p12
59    #             password:
60    #             type: PKCS12
61      security:
62        enabled: true
63        #ip-patterns:
64        #  - 127.0.0.1
65  ---
66  endpoints:
67    all:
68      enabled: true
69      sensitive: true
70  ---
71  # Set the basic auth username and password that can be used to reach
72  # any built in endpoint
73  http:
74    admin:
75      username:
76      password:
77  ---
78  # This will connect the worker to the DMF Migration database
```

```
 79   # to get/update worker state and get/update migration jobs
 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   dmf:
 91     worker:
 92       sources: default
 93       state:
 94         # The workers initial state. When using the database state type, if the worker has
                  been
 95         # previously started then it will use the initial state in the database to determine
                  how to start.
 96         # Current options:
 97         #    1. STARTED
 98         #    2. STOPPED
 99         initial: STARTED
100         # The way the worker will collect and store its state.
101         # Options:
102         #    1. database - uses the DMF migration database
103         #    2. config - uses this config to get the initial state and max jobs
104         type: database
105       jobs:
106         cache:
107           size: 1000
108           refresh: 5s
109         # Max number of jobs to queue at a time. When using the database state type, if the
                Worker has previously
110         # been started, then this value will be ignored and the maxThreads defined for the
                worker will be used.
111         #
112         # Keep in mind that if the executor pool is smaller than this number then
113         # some jobs will be waiting in the executor pool.
114         max: 10
115         termination:
116           # The max time to wait in ms for jobs to complete on application termination
                  before willfully shutting down.
117           # Shutting down the application while jobs are running can leave DMF in a corrupt
                  state if jobs do not complete.
118           # While jobs should shutdown quickly when abort is called, this should be set to a
                  high value.
119           timeout: 30000
120         poller:
121           # The job poller type. Current options:
122           #    1. database - collects jobs from the DMF migration database
123           #    2. simulator - creates simulated jobs meant for testing other parts of DMF
124           type: database
125           simulator:
126             max-jobs: 5
127             max-wait-ms: 0
128       identity:
129         # Identifies a group of DMF workers. This should be the same for all DMF workers
                  that service a particular Dovecot platform.
130         # This should match a "target" that has been registered with the DMF REST API.
131         target: default
132         # Identifies this worker within a group of DMF workers. This should be unique within
                  a target.
133         memberid:
134       # The worker type. Current options:
135       #    1. doveadm - uses doveadm to complete the migration job
136       #    2. command - executes the migration command as is
137       #    3. simulator - does not actually run any commands meant for testing the Worker
                roundtrip
138       type: doveadm
139       command:
```

```
140        logging:
141          # Where to redirect standard out and error when running commands
142          # Options:
143          #   1. file: logs to a file (see file properties)
144          #   2. logger: logs to the logger
145          #   3. off: silences logging
146          type: file
147          file:
148            # The log file name format. You can use the following specifiers:
149            #   * source: the source name
150            #   * user: the target uid
151            #   * phase: the migration phase (pre-sync, cutover)
152            #   * job: the job id
153            #   * date: the YYYY-MM-DD date
154            format: "%(source)-%(user)-user-migration.log"
155            location: /app
156            delete-on-success: true
157    # Define doveadm properties
158    doveadm:
159      # The path to the doveadm command
160      path: "/usr/bin/doveadm"
161      # DMF Source specific properties should go under the source name key
162      source:
163        # If a source is not defined then it will use the default if it exists
164        default:
165          ####
166          # The worker will create a log analyzer which can be used for finding errors (some
                  which
167          # could be automatically resolved) and mailbox statistics.
168          # By setting to false, the following functionality will be lost:
169          #   - Find and resolve duplicate UIDs
170          #   - Find and resolve failed save due to timeout
171          #   - Collect errors for the job response
172          #   - Collect sync mail statistics
173          ####
174          analyze-log: true
175          ####
176          # If the migration command fails due to mails with duplicate UIDs, then it will
                  attempt
177          # to connect to the source IMAP to move those duplicate mails into a folder called
178          # lost+found-<originalFolderName>. If retry is enabled then the command will be
                  rerun.
179          # Duplicates can only be found if analyze-log is enabled.
180          ####
181          move-duplicates: false
182          ####
183          # A hack to speed up delta syncs with pop3 uild. It copies the local user's INBOX
184          # mailbox to <userHome>/imapc/.INBOX/.INBOX.
185          # This is done after the pre migration script but before the migration command
186          ####
187          copy-imapc-inbox: false
188          ####
189          # If using copy-imapc-inbox, this will remove the created imapc folder for the
                  local user.
190          # This is done after the migration command, regardless of outcome, but before
191          # any post migration script
192          ####
193          remove-imapc-path: false
194          ####
195          # Executes metacache flush for the local user.
196          # This is only done after a successful migration command but before the post
                  migration script.
197          ####
198          flush-metacache: false
199          ####
200          # Executes metacache clean for the local user.
201          # This is only done after a successful migration command but before the post
                  migration script.
202          ####
203          clean-metacache: false
204          ####
205          # Gets the userdb_container value in the file /etc/dovecot/passwd.container for
```

```
                 the value of
206         # imapcoptions defined for the user. If imapcoptions is not set for the user, but
                 this is enabled
207         # then it wont be used. The found value can be injected into migration commands
                 with %{mdb:container}
208         ####
209         fetch-container: false
210         ####
211         # The dovecot passwd.container file path that is used when fetch-container is
                 enabled.
212         ####
213         container-file: /etc/dovecot/passwd.container
214         ####
215         # Target director config
216         ####
217         director:
218           ####
219           # Informs the worker that it should use its director's HTTP API
220           # to communicate about the migration. This requires the http configuration
221           # and the HTTP API to exist on a director.
222           ####
223           enabled: false
224           ####
225           # Currently not supported
226           ####
227           use-sourcehost: false
228           ####
229           # Inform the director layer via the HTTP API about being managed by this backend
                   .
230           # If use-director property is set then this will use the directors IP.
231           ####
232           move-user: false
233         ####
234         # Properties to configure counting of mailbox messages and size
235         ####
236         count:
237           ####
238           # Collect local mailbox stats after the pre migration script but before the
239           # migration command. The results are simply logged.
240           ####
241           pre: true
242           ####
243           # Count the number of saved and expunged mails during the migration command.
244           # The results are stored with the job response as the sync saved and expunged
245           # message counts.
246           # Sync stats can only be collected if analyze-log is enabled.
247           ####
248           sync: true
249           ####
250           # Collect local mailbox stats after the successful migration command but
251           # before the post migration script. The results are stored with the job
252           # response as the target mailbox size and message count.
253           ####
254           post: true
255           ####
256           # Collect the remote mailbox stats after the successful migration command but
257           # before the post migration script. The results are stored with the job
258           # response as the origin mailbox size and message count.
259           ####
260           remote: true
261           ####
262           # The protocol to use when executing the remote count if the default command
                   will be used.
263           # Options:
264           #   - doveadm: doveadm -o doveadm_password=<doveadm.password> mailbox status -u
                   <ruid>
265           #                -S <doveadm.host>:<doveadm.port> "messages vsize" INBOX/* INBOX *
266           #   - imapc: doveadm -o imapc_ssl=<imaps/no> -o imapc_host=<sourcehost> -o
                   imapc_user=<ruid>
267           #                -o imapc_password=<password> -o imapc_port=<port> -o mail=imapc:
268           #                mailbox status -u <uid> "messages vsize" INBOX/* INBOX *
269           ####
```

```
270         remote-protocol: imapc
271         ####
272         # When enabled, the remote mailbox count by default will execute doveadm mailbox
                   status
273         # using the doveadm remote protocol. This can be overridden by defining the
                   command
274         # with the sourcehost definition. It can also be overridden here which will take
                   precedence.
275         # The format here is the common command format. However, there are restrictions
                   to this.
276         # The root command "doveadm" and sub command "mailbox status" with fields "
                   messages vsize" will
277         # always be used and without debug or verbosity, and will use a tab formatter.
                   Do not change these
278         # or it will corrupt the counting. The only important thing to include are
                   setting overrides and
279         # mailbox status mailbox patterns.
280         # Format:
281         #   name: doveadm
282         #   options:
283         #     -
284         #       name: -o
285         #       value: "imapc_user=%{mdb:ruid}"
286         #     -
287         #       name: -o
288         #       value: "imapc_password=%{conf:imapc_master_password}"
289         #     -
290         #       name: -o
291         #       value: "imapc_host=%{mdb:sourcehost}"
292         #     -
293         #       name: -o
294         #       value: "mail=imapc:"
295         #   sub-command:
296         #     name: mailbox status
297         #     arguments:
298         #       - "messages vsize"
299         #       - "INBOX/*"
300         #       - "INBOX"
301         #       - "*"
302         #     options:
303         #       -
304         #         name: -u
305         #         value: "%{mdb:uid}"
306         ####
307         remote-command:
308       ####
309       # Properties to configure retrying the migration command after a failure.
310       ####
311       retry:
312         ####
313         # Max number of retries for doveadm sync errors which are non fatal.
314         # To override any specific error code, use code-max.
315         ####
316         max: 1
317         ####
318         # The amount of time in ms to sleep before retrying the migration command.
319         # To override any specific error code use code-sleep.
320         ####
321         sleep: 5000
322         ####
323         # Override for max on the error code level. Format is:
324         # code-max:
325         #   <code>: <num_retries>
326         ####
327         code-max:
328           134: 2
329           75: 4
330         ####
331         # Override for sleep on the error code level. Format is:
332         # code-sleep:
333         #   <code>: <time_in_ms>
334         ####
```

```
335            code-sleep:
336              75: 15000
337        ####
338        # Properties to define connection to a local DoveAdm HTTP API.
339        # Currently this is only used when the director is enabled.
340        ####
341        #http:
342        #   url: "https://doveadmhttpapi"
343        #   username:
344        #   password:
345        #   director-tag: aa
346        #   use-only-tagged: false
347        ####
348        # Properties for running doveadm commands on the Source using
349        # the doveadm protocol
350        ####
351        #doveadm:
352        #   password:
353        #   host:
354        #   port: 24245
355        ####
356        # IMAP connection properties use to inject into commands or use
357        # for connecting to a users mailbox. You may specify some or all properties.
358        # For IMAP connections, these will override values defined with the job.
359        ####
360        imapc:
361          host:
362          port:
363          prefix:
364          master-user:
365          master-password:
366          ####
367          # Specify java mail api properties here
368          ####
369          session-properties:
370        ####
371        # Properties to configure commands accessible by the worker to be executed
372        # during the specific migration phases.
373        #
374        # pre: executed prior to the migration command and some other configurable
               operations.
375        # sync: Overrides the migration command.
376        # post: executed after a successful migration command and some other configurable
               operations.
377        # failure: executed after a migration command failure. When retries are enabled,
               this is
378        # only executed if the last retry is a failure, otherwise post is executed.
379        #
380        # Commands can have properties injected into them:
381        #   MDB formatters are used like %{mdb:X}, where X can be: md5path, 2chrruid,
               container,
382        #     uid, ruid, sourcehost, source, sourcepasswd, imapcoptions, email, sourceport,
               imapc_ssl, exclude
383        #   Conf formatters are used like %(conf:Y}, where Y can be any value defined in
               the "inject" config
384        #     or: imapc_host, imapc_master_password, imapc_master_user, imapc_prefix,
               imapc_port
385        ####
386        command:
387          ####
388          # Custom properties that can be defined and injected into the migration or
389          # other definable commands. For example, if you define:
390          #   inject:
391          #     test: value
392          # then you can have a command "doveadm backup -o setting=%{conf:test}" and the
               command
393          # would resolve to "doveadm backup -o setting=value"
394          # The order of injection is:
395          #   1. inject properties
396          #   2. user properties - from job
397          #   3. imapc properties - from config
398          ####
```

```
399        inject:
400        ####
401        # Overrides the migration command during pre-sync.
402        #
403        # Format + Example:
404        # sync:
405        #   sync:
406        #     name: doveadm
407        #     options:
408        #       -
409        #         name: -o
410        #         value: "imapc_user=%{mdb:ruid}"
411        #       -
412        #         name: -o
413        #         value: "imapc_password=%{conf:imapc_master_password}"
414        #       -
415        #         name: -o
416        #         value: "imapc_host=%{mdb:sourcehost}"
417        #     sub-command:
418        #       name: backup
419        #       flags: -R
420        #       arguments: "imapc:"
421        #       options:
422        #         -
423        #           name: -u
424        #           value: "%{mdb:uid}"
425        ####
426        sync:
427          ####
428          # Example:
429          # pre:
430          #   name: echo
431          #   arguments: "%{mdb:uid}"
432          ####
433          pre:
434          sync:
435          post:
436          failure:
437        ####
438        # Overrides the migration command during cutover. See sync for format.
439        ####
440        cutover:
441          pre:
442          sync:
443          post:
444          failure:
```