TechD·vision

Pacemaker

Table of Contents

PACEMAKER

Pacemaker is an extension bundle (1) for Magento 2. It is a new way to organize your background processes inside your shop system. Therefore Pacemaker uses the pipeline pattern, which is familiar to most developers from tools like Gitlab, Jenkins or Travis. Developers are using these tools to manage complex processes like building and deployment of software. With Pacemaker it is possible to use this powerful pattern also in your eCommerce infrastructure.

Copyright and License

Introduction

Pacemaker Context

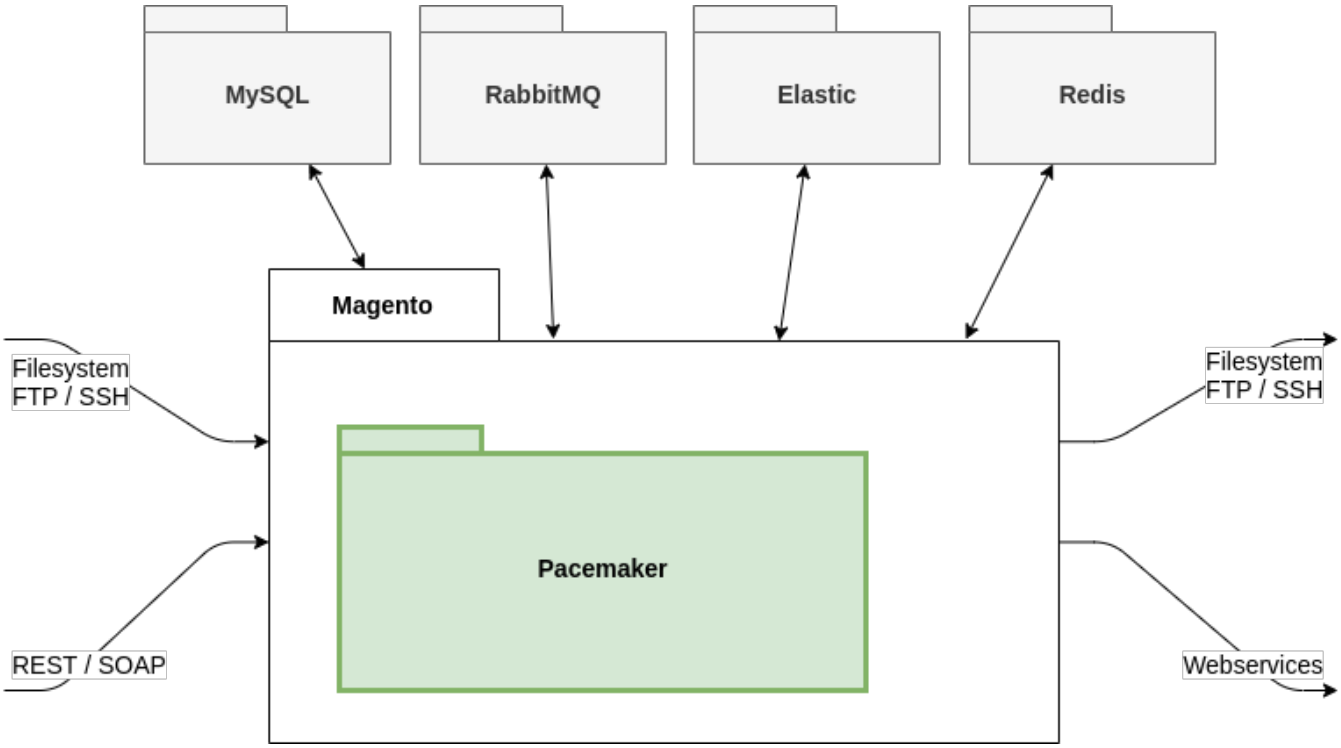


Figure 1. Pacemaker Context View

Pacemaker is a Magento extension, which has access to all existing interfaces of Magento as well as has the possibility to introduce new connections to foreign or local systems.

Pacemaker Components

Pacemaker consists of many modules. In the current version, these modules can be grouped into four components. For a better understanding of Pacemaker, it is necessary to understand that each of these components is a solution for different types of problems. Each component has its own dependencies and whenever you introduce new functionality to Pacemaker, you need to identify the right component where to integrate your code.

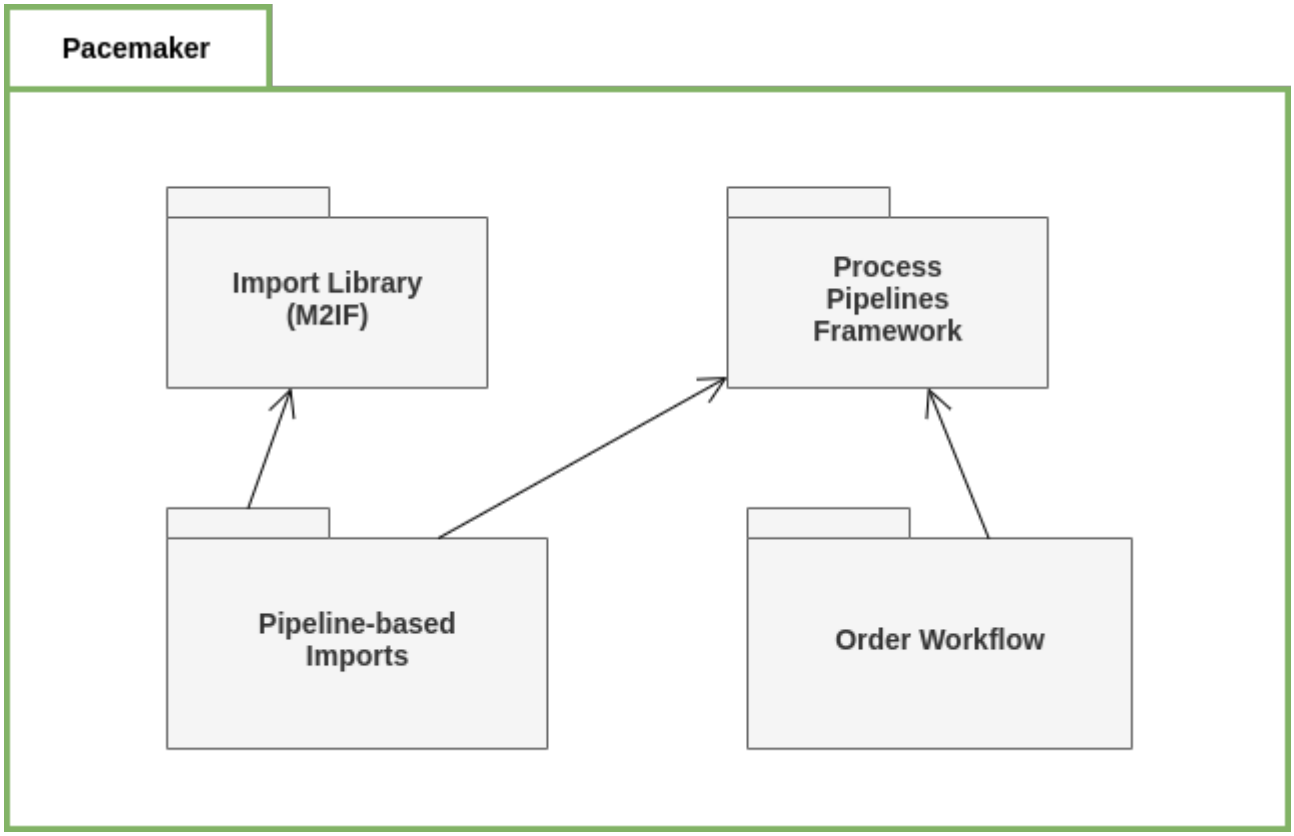


Figure 2. Pacemaker Components

Get started

How to install

Before you begin the install, you need a running Magento 2 instance. Please refer Magento's documentation: Install Magento using Composer. You need also to install RabbitMQ and configure it for your Magento instance. Refer the article from Magento's DevDocs.

Install Pacemaker via Composer

After purchasing a Pacemaker license you will receive a **username** and **password** from our support team. You need to enter these credentials to the auth.json file of your Magento instance. If you didn't use this file before, you'll find an auth.json.sample file, just copy it as auth.json.

Enter our repository (gitlab.met.tdintern.de), your username and password as following into the http-basic section of your auth.json file.

```
{
   "http-basic": {
      "repo.magento.com": {
         "username": "...",
         "password": "..."
      },
      "gitlab.met.tdintern.de": {
         "username": "<YOUR-TOKEN-USER>",
         "password": "<YOUR-TOKEN-PASS>"
      }
   }
}
```

Register the repository

After adding the credentials you need to register the repository as a possible source. Therefore you need to run the following command or add the repository manually into your composer.json file.

Commandline registration

```
composer config repositories.repo.met.tdintern.de composer https://repo.met.tdintern.de/
```

(Optional) Manual entry in composer.json

```
...
"repositories": {
"repo.met.tdintern.de": {
  "type": "composer",
  "url": "https://repo.met.tdintern.de/"
},
```

```
"repo.magento.com": {
  "type": "composer",
  "url": "https://repo.magento.com/"
}
...
```

Install the module

Now you can run composer require techdivision/pacemaker="~1.2.0"to install the module.

Configure Heartbeat Cron

The heartbeat cronjob is one of the major parts of Pacemaker. It is necessary to have this job up and running to use Pacemaker.

The heartbeat executes the condition checks for and initiates pipelines, which are ready to be executed. At the same time, it checks the conditions for steps within existing pipelines and triggers the execution, if a step is ready to run.

Default configuration

By default you should create a new crontab entry within your OS as following:

```
* * * * *  /usr/bin/env php <PATH_TO_MAGENTO_ROOT>/bin/magento pipeline:heartbeat
```

This configuration will run the heartbeat every minute.

Another heartbeat interval

If you need to run the heartbeat less often then every minute you need to define the pulse to avoid time gaps within time-based conditions.

Example for a heartbeat, which is running every two minues:

```
*/2 * * * *  /usr/bin/env php <PATH_TO_MAGENTO_ROOT>/bin/magento pipeline:heartbeat --pulse
2
```

How to configure the runners

Runners are doing the job. They execute the major logic of each step if the step is ready to run. You need to have at least one runner up and running. You should have multiple runners to use the multiprocessing capabilities of Pacemaker.

How to execute a runner

A Pacemaker runner is a message queue consumer. Therefore you need to run the default Magento queue:consumers:start command to start a Pacemaker pipeline runner.

```
bin/magento queue:consumers:start pipelineRunner
```

Avoid using cron_consumers_runner

Magento provides the ability to start MQ consumers via cron. We advise not to use this feature. Supervisor is a much better option to manage your consumers.

In cases, you are using tools like Supervisor to run Pacemaker runners, you MUST update your cron_consumer_runner configuration not to start pipelineRunner. Otherwise, this will lead in runtime issues because you will have different versions of pipelineRunner up and running, probably with different configurations.

Using supervisor for Pacemaker runners

We recommend to use supervisor in order to run multiple runners and ensure, that all runners are up and running all the time. See following supervisor configuration example:

```
[program:pipelineRunner]
stderr_logfile = <MAGENTO_ROOT>/var/pipelineRunner.log
autorestart = 1
autostart = 1
startsecs = 0
startretries = 0
directory = <MAGENTO_ROOT>
numprocs = 4
process_name = %(program_name)s_%(process_num)02d
user = www-data
command = /usr/bin/env php <MAGENTO_ROOT>/bin/magento queue:consumers:start --max-messages 1
pipelineRunner
```

Limit runner messages (--max-messages 1)

We recommend using the --max-messages option with value 1to avoid issues within stateful PHP code. Of course - it depends on how you implement your job executors, but there is still vendor code, which could have a state and this would raise issues, which are easy to fix but hard to detect.

startsecs config option

This option is required, since we have executors, which are running faster than one second. In this case supervisor would stop restarting the runners. Please refer to the supervisor configuration documentation for more details

TIP If you're using MacOS X please refer to how to setup RabbitMQ on MacOS X

Setup RabbitMQ on MacOS X

Assuming you have installed RabbitMQ with brew, you may have to execute the following steps to connect Magento and RabbitMQ.

Create Credentials & Configuration

To fix these issues or to create the connection if not already done, start by creating the RabbitMQ user and virtual host as well as setting the permissions with

```
rabbitmqctl add_user admin admin
rabbitmqctl add_vhost pacemaker
rabbitmqctl set_permissions -p pacemaker admin "." "." ".*"
```

In the commands above, we create a user with the username admin and the password admin. Additionally a virtual host for RabbitMQ with name pacemaker has been created.

Then add the queue node to the Magento configuration under app/etc/env.php. With the username/password and virtual host values from above, this has to look like

```
return [
    'queue' => [
        'amqp' => [
            'host' => 'localhost',
            'port' => '5672',
            'user' => 'admin',
            'password' => 'admin',
            'virtualhost' => 'pacemaker',
            'ssl' => 'false'
        ]
    ],
];
```

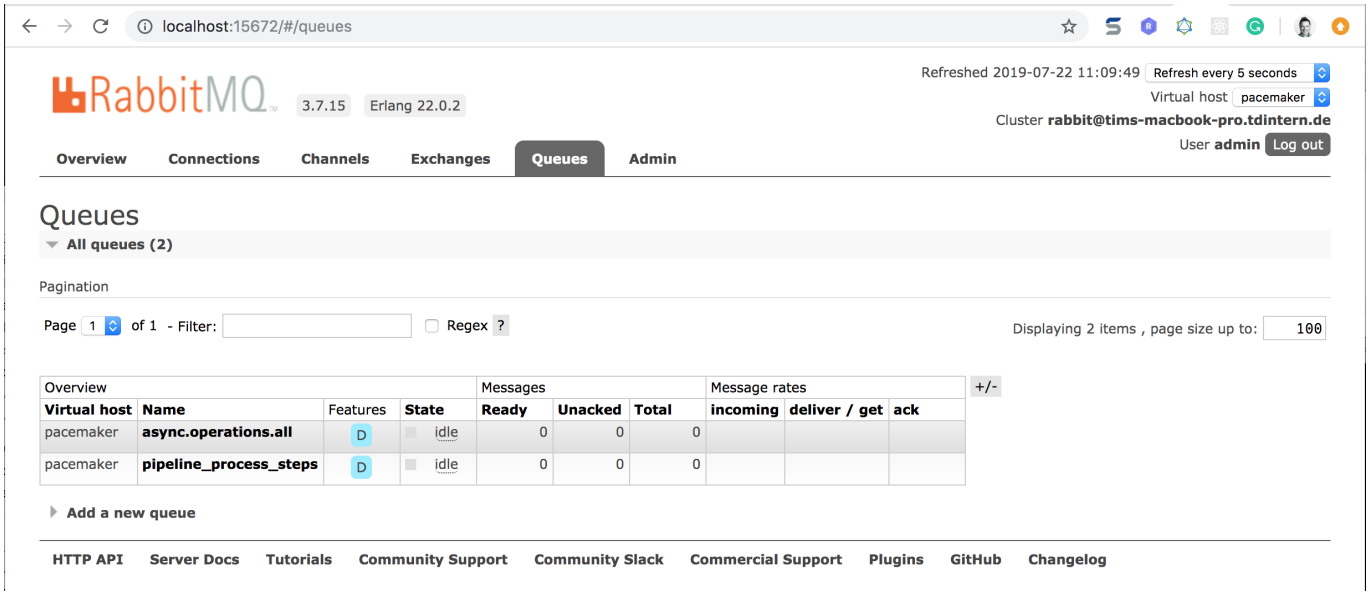
After that, finalize the Magento setup with the following command

```
bin/magento setup:upgrade
```

When this command has been executed successfully, the queues within RabbitMQ has been created and the connection has been established. Finally, to start the runner, enter the following command

```
bin/magento queue:consumers:start pipelineRunner
```

Additionally in the RabbitMQ GUI the queues should now be visible like

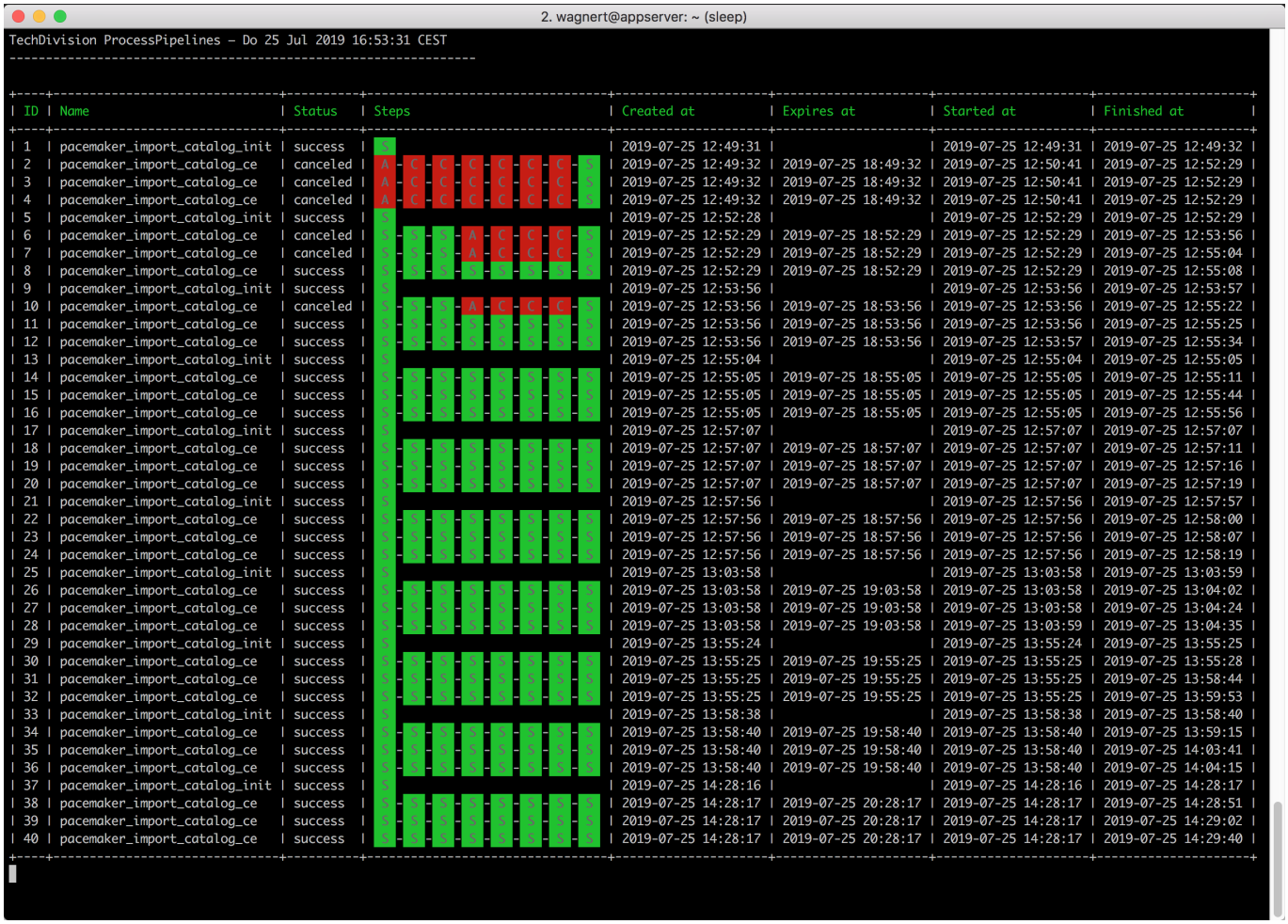


CLI Status Update

To get a brief overview of the running processes (helpful for local development and debugging), it is possible to execute a console command that renders the status of the running pipelins on the console

```
bin/magento pipeline:status -w 2
```

This should render the following output



ID	Name	Status	Steps	Created at	Expires at	Started at	Finished at
1	pacemaker_import_catalog_init	success	S	2019-07-25 12:49:31		2019-07-25 12:49:31	2019-07-25 12:49:32
2	pacemaker_import_catalog_ce	canceled	A-C-C-C-C-C-C-S	2019-07-25 12:49:32	2019-07-25 18:49:32	2019-07-25 12:50:41	2019-07-25 12:52:29
3	pacemaker_import_catalog_ce	canceled	A-C-C-C-C-C-C-S	2019-07-25 12:49:32	2019-07-25 18:49:32	2019-07-25 12:50:41	2019-07-25 12:52:29
4	pacemaker_import_catalog_ce	canceled	A-C-C-C-C-C-C-S	2019-07-25 12:49:32	2019-07-25 18:49:32	2019-07-25 12:50:41	2019-07-25 12:52:29
5	pacemaker_import_catalog_init	success	S	2019-07-25 12:52:28		2019-07-25 12:52:29	2019-07-25 12:52:29
6	pacemaker_import_catalog_ce	canceled	S-S-S-A-C-C-C-S	2019-07-25 12:52:29	2019-07-25 18:52:29	2019-07-25 12:52:29	2019-07-25 12:53:56
7	pacemaker_import_catalog_ce	canceled	S-S-S-A-C-C-C-S	2019-07-25 12:52:29	2019-07-25 18:52:29	2019-07-25 12:52:29	2019-07-25 12:55:04
8	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:52:29	2019-07-25 18:52:29	2019-07-25 12:52:29	2019-07-25 12:55:08
9	pacemaker_import_catalog_init	success	S	2019-07-25 12:53:56		2019-07-25 12:53:56	2019-07-25 12:53:57
10	pacemaker_import_catalog_ce	canceled	S-S-S-A-C-C-C-S	2019-07-25 12:53:56	2019-07-25 18:53:56	2019-07-25 12:53:56	2019-07-25 12:55:22
11	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:53:56	2019-07-25 18:53:56	2019-07-25 12:53:56	2019-07-25 12:55:25
12	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:53:56	2019-07-25 18:53:56	2019-07-25 12:53:57	2019-07-25 12:55:34
13	pacemaker_import_catalog_init	success	S	2019-07-25 12:55:04		2019-07-25 12:55:04	2019-07-25 12:55:05
14	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:55:05	2019-07-25 18:55:05	2019-07-25 12:55:05	2019-07-25 12:55:11
15	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:55:05	2019-07-25 18:55:05	2019-07-25 12:55:05	2019-07-25 12:55:44
16	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:55:05	2019-07-25 18:55:05	2019-07-25 12:55:05	2019-07-25 12:55:56
17	pacemaker_import_catalog_init	success	S	2019-07-25 12:57:07		2019-07-25 12:57:07	2019-07-25 12:57:07
18	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:57:07	2019-07-25 18:57:07	2019-07-25 12:57:07	2019-07-25 12:57:11
19	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:57:07	2019-07-25 18:57:07	2019-07-25 12:57:07	2019-07-25 12:57:16
20	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:57:07	2019-07-25 18:57:07	2019-07-25 12:57:07	2019-07-25 12:57:19
21	pacemaker_import_catalog_init	success	S	2019-07-25 12:57:56		2019-07-25 12:57:56	2019-07-25 12:57:57
22	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:57:56	2019-07-25 18:57:56	2019-07-25 12:57:56	2019-07-25 12:58:00
23	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:57:56	2019-07-25 18:57:56	2019-07-25 12:57:56	2019-07-25 12:58:07
24	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 12:57:56	2019-07-25 18:57:56	2019-07-25 12:57:56	2019-07-25 12:58:19
25	pacemaker_import_catalog_init	success	S	2019-07-25 13:03:58		2019-07-25 13:03:58	2019-07-25 13:03:59
26	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:03:58	2019-07-25 19:03:58	2019-07-25 13:03:58	2019-07-25 13:04:02
27	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:03:58	2019-07-25 19:03:58	2019-07-25 13:03:58	2019-07-25 13:04:24
28	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:03:58	2019-07-25 19:03:58	2019-07-25 13:03:59	2019-07-25 13:04:35
29	pacemaker_import_catalog_init	success	S	2019-07-25 13:55:24		2019-07-25 13:55:24	2019-07-25 13:55:25
30	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:55:25	2019-07-25 19:55:25	2019-07-25 13:55:25	2019-07-25 13:55:28
31	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:55:25	2019-07-25 19:55:25	2019-07-25 13:55:25	2019-07-25 13:58:44
32	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:55:25	2019-07-25 19:55:25	2019-07-25 13:55:25	2019-07-25 13:59:53
33	pacemaker_import_catalog_init	success	S	2019-07-25 13:58:38		2019-07-25 13:58:38	2019-07-25 13:58:40
34	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:58:40	2019-07-25 19:58:40	2019-07-25 13:58:40	2019-07-25 13:59:15
35	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:58:40	2019-07-25 19:58:40	2019-07-25 13:58:40	2019-07-25 14:03:41
36	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 13:58:40	2019-07-25 19:58:40	2019-07-25 13:58:40	2019-07-25 14:04:15
37	pacemaker_import_catalog_init	success	S	2019-07-25 14:28:16		2019-07-25 14:28:16	2019-07-25 14:28:17
38	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 14:28:17	2019-07-25 20:28:17	2019-07-25 14:28:17	2019-07-25 14:28:51
39	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 14:28:17	2019-07-25 20:28:17	2019-07-25 14:28:17	2019-07-25 14:29:02
40	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S-S	2019-07-25 14:28:17	2019-07-25 20:28:17	2019-07-25 14:28:17	2019-07-25 14:29:40

Issues with missing AMQP configuration

If you did not set-up the RabbitMQ connection when installing Magento, you'll probably receive one of the two errors below when you try to start the Pacemaker runner.

The message Unknown connection name amqp signals that the Magento configuration under app/etc/env.php is missing the AMQP connection

```
$ bin/magento queue:consumers:start pipelineRunner

In ConnectionTypeResolver.php line 43:

  Unknown connection name amqp

queue:consumers:start [--max-messages MAX-MESSAGES] [--batch-size BATCH-SIZE] [--area-code
AREA-CODE]
[--pid-file-path PID-FILE-PATH] [--] <consumer>
```

whereas the message ACCESS_REFUSED - Login was refused using authentication mechanism AMQPLAIN. For details see the broker logfile. addresses an issue with the configured username/password in the configuration.

```
$ bin/magento queue:consumers:start pipelineRunner
```

```
In AbstractConnection.php line 689:

  ACCESS_REFUSED - Login was refused using authentication mechanism AMQPLAIN. For details
see the broker logfile.

queue:consumers:start [--max-messages MAX-MESSAGES] [--batch-size BATCH-SIZE] [--area-code
AREA-CODE]
[--pid-file-path PID-FILE-PATH] [--] <consumer>
```

Run your first predefined import jobs

Pacemaker provides predefined import jobs, which can be used out of the box. Therefore there are sample import files (CSV) included in the installed composer packages. By importing this sample files, you'll import Magento's sample data, like they would be created by running the sampledata:deploy command.

Prerequisites

The following tutorial requires an up and running Pacemaker like it is described on the following pages:

- How to install the module / extension
- How to configure the heartbeat (cron)
- How to configure the runner(s)

Copy sample files into observed import directory

You can simply copy all sample files into the import directory and Pacemaker would automatically initialize import pipelines (What is a pipeline?) for each import bunch (a bunch of CSV files). Execute the following command from the root directory of your Magento installation depending on which kind of import you want to execute.

Catalog Import (Attributes, Categories, Products)

Sample data set 1

The first sample data set includes all kinds of imports (attribute sets, attributes, categories, and products), which are bundled in multiple bunches.

```
cp -R vendor/techdivision/pacemaker-import-catalog/sample-data/bunch1/* var/pacemaker/import
```

Sample data set 2

The second sample data set includes only category imports, which are bundled into two bunches.

```
cp -R vendor/techdivision/pacemaker-import-catalog/sample-data/bunch2/* var/pacemaker/import
```

Price Import / Inventory (stock) Import

Sample data for price and inventory import includes only one import file each. But it is also possible to split up these imports into multiple files and optionally cluster them into multiple bunches like it is done for the catalog import.

Use the following command for price import:

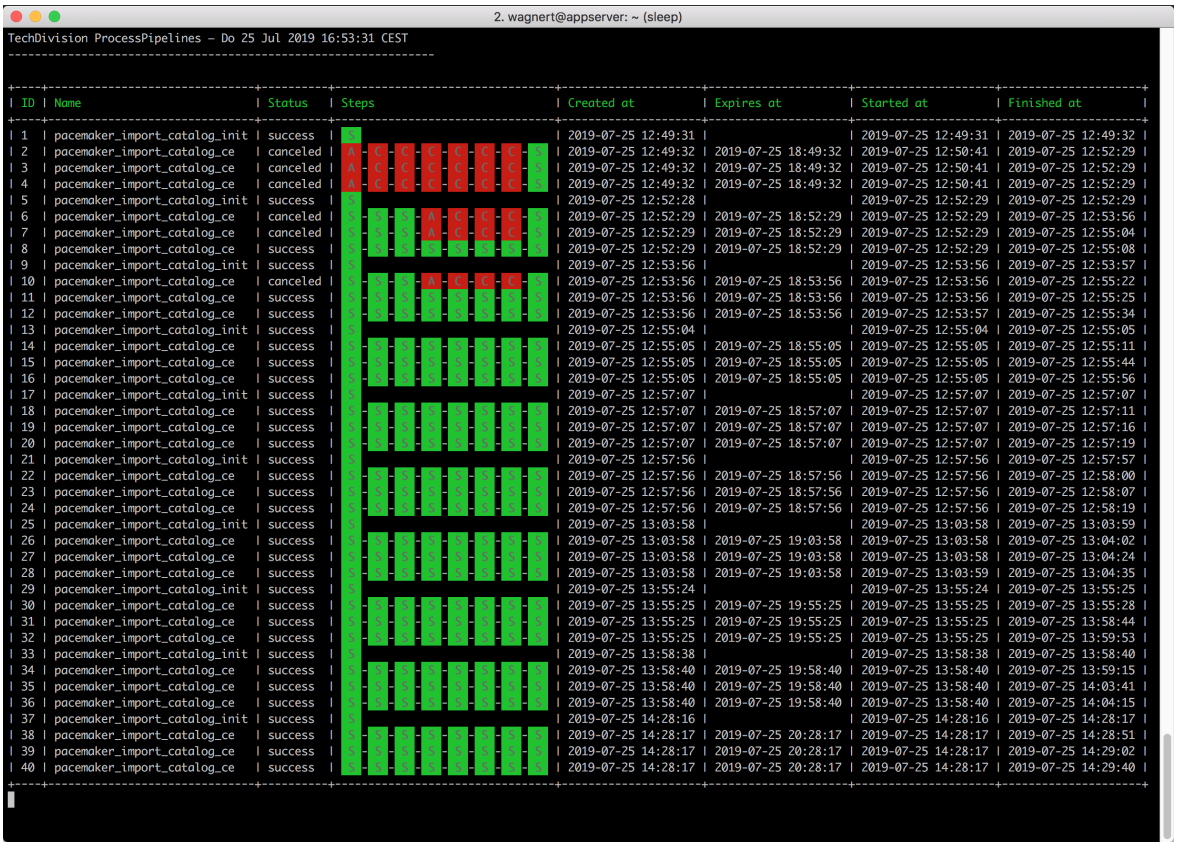
```
cp -R vendor/techdivision/pacemaker-import-price/sample-data/bunch1/* var/pacemaker/import
```

Use the following command for inventory (stock) import:

```
cp -R vendor/techdivision/pacemaker-import-inventory/sample-data/bunch1/*
var/pacemaker/import
```

NOTE

After copying the import files into the target directory you can observe the process by executing the pipeline:status command. Alternatively login into the backend (Magento's admin UI) and open the System > Pacemaker > Pipelines Grid.



ID	Name	Status	Steps	Created at	Expires at	Started at	Finished at
1	pacemaker_import_catalog_init	success	S	2019-07-25 12:49:31		2019-07-25 12:49:31	2019-07-25 12:49:32
2	pacemaker_import_catalog_ce	canceled	A-C-C-C-C-C-S	2019-07-25 12:49:32	2019-07-25 18:49:32	2019-07-25 12:50:41	2019-07-25 12:52:29
3	pacemaker_import_catalog_ce	canceled	A-C-C-C-C-C-S	2019-07-25 12:49:32	2019-07-25 18:49:32	2019-07-25 12:50:41	2019-07-25 12:52:29
4	pacemaker_import_catalog_ce	canceled	A-C-C-C-C-C-S	2019-07-25 12:49:32	2019-07-25 18:49:32	2019-07-25 12:50:41	2019-07-25 12:52:29
5	pacemaker_import_catalog_init	success	S	2019-07-25 12:52:28		2019-07-25 12:52:29	2019-07-25 12:52:29
6	pacemaker_import_catalog_ce	canceled	S-S-S-A-C-C-S	2019-07-25 12:52:29	2019-07-25 18:52:29	2019-07-25 12:52:29	2019-07-25 12:53:56
7	pacemaker_import_catalog_ce	canceled	S-S-S-A-C-C-S	2019-07-25 12:52:29	2019-07-25 18:52:29	2019-07-25 12:52:29	2019-07-25 12:55:04
8	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:52:29	2019-07-25 18:52:29	2019-07-25 12:52:29	2019-07-25 12:55:08
9	pacemaker_import_catalog_init	success	S	2019-07-25 12:53:56		2019-07-25 12:53:56	2019-07-25 12:53:57
10	pacemaker_import_catalog_ce	canceled	S-S-S-A-C-C-S	2019-07-25 12:53:56	2019-07-25 18:53:56	2019-07-25 12:53:56	2019-07-25 12:55:22
11	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:53:56	2019-07-25 18:53:56	2019-07-25 12:53:56	2019-07-25 12:55:25
12	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:53:56	2019-07-25 18:53:56	2019-07-25 12:53:57	2019-07-25 12:55:34
13	pacemaker_import_catalog_init	success	S	2019-07-25 12:55:04		2019-07-25 12:55:04	2019-07-25 12:55:05
14	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:55:05	2019-07-25 18:55:05	2019-07-25 12:55:05	2019-07-25 12:55:11
15	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:55:05	2019-07-25 18:55:05	2019-07-25 12:55:05	2019-07-25 12:55:44
16	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:55:05	2019-07-25 18:55:05	2019-07-25 12:55:05	2019-07-25 12:55:56
17	pacemaker_import_catalog_init	success	S	2019-07-25 12:57:07		2019-07-25 12:57:07	2019-07-25 12:57:07
18	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:57:07	2019-07-25 18:57:07	2019-07-25 12:57:07	2019-07-25 12:57:11
19	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:57:07	2019-07-25 18:57:07	2019-07-25 12:57:07	2019-07-25 12:57:16
20	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:57:07	2019-07-25 18:57:07	2019-07-25 12:57:07	2019-07-25 12:57:19
21	pacemaker_import_catalog_init	success	S	2019-07-25 12:57:56		2019-07-25 12:57:56	2019-07-25 12:57:57
22	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:57:56	2019-07-25 18:57:56	2019-07-25 12:57:56	2019-07-25 12:58:00
23	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:57:56	2019-07-25 18:57:56	2019-07-25 12:57:56	2019-07-25 12:58:07
24	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 12:57:56	2019-07-25 18:57:56	2019-07-25 12:57:56	2019-07-25 12:58:19
25	pacemaker_import_catalog_init	success	S	2019-07-25 13:03:58		2019-07-25 13:03:58	2019-07-25 13:03:59
26	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:03:58	2019-07-25 19:03:58	2019-07-25 13:03:58	2019-07-25 13:04:02
27	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:03:58	2019-07-25 19:03:58	2019-07-25 13:03:58	2019-07-25 13:04:24
28	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:03:58	2019-07-25 19:03:58	2019-07-25 13:03:59	2019-07-25 13:04:35
29	pacemaker_import_catalog_init	success	S	2019-07-25 13:55:24		2019-07-25 13:55:24	2019-07-25 13:55:25
30	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:55:25	2019-07-25 19:55:25	2019-07-25 13:55:25	2019-07-25 13:55:28
31	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:55:25	2019-07-25 19:55:25	2019-07-25 13:55:25	2019-07-25 13:58:44
32	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:55:25	2019-07-25 19:55:25	2019-07-25 13:55:25	2019-07-25 13:59:53
33	pacemaker_import_catalog_init	success	S	2019-07-25 13:58:38		2019-07-25 13:58:38	2019-07-25 13:58:40
34	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:58:40	2019-07-25 19:58:40	2019-07-25 13:58:40	2019-07-25 13:59:15
35	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:58:40	2019-07-25 19:58:40	2019-07-25 13:58:40	2019-07-25 14:03:41
36	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 13:58:40	2019-07-25 19:58:40	2019-07-25 13:58:40	2019-07-25 14:04:15
37	pacemaker_import_catalog_init	success	S	2019-07-25 14:28:16		2019-07-25 14:28:16	2019-07-25 14:28:17
38	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 14:28:17	2019-07-25 20:28:17	2019-07-25 14:28:17	2019-07-25 14:28:51
39	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 14:28:17	2019-07-25 20:28:17	2019-07-25 14:28:17	2019-07-25 14:29:02
40	pacemaker_import_catalog_ce	success	S-S-S-S-S-S-S	2019-07-25 14:28:17	2019-07-25 20:28:17	2019-07-25 14:28:17	2019-07-25 14:29:40

Figure 3. Result output for bin/magento pipeline:status

Features & Configuration

General Settings

Pipeline Settings

Pipelines are the core components for Pacemaker. Please refer to Components & Concepts > Process Pipelines for more details. There are settings, which defines the overall behaviour of pipelines. This settings could be found under following path: Stores > Configuration > Pacemaker > Pipeline Settings

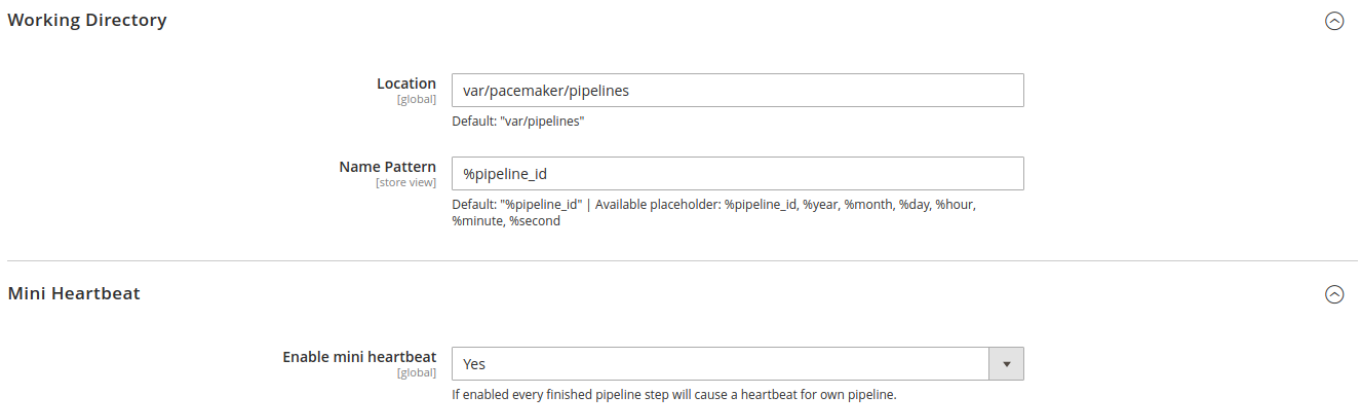


Figure 4. Configuration in Magento Backend

Working directory

In this section you may configure the location of the working directory. The path could be relative to Magento's root directory or absolute. Further you can specify how the working directory should be named. It is possible to use following placeholders:

- %pipeline_id
- %year
- %month
- %day
- %hour
- %minute
- %second

Mini Heartbeat

This feature triggers a heartbeat every time a step was executed. This heartbeat is restricted to the one pipeline, which the concerned step belongs to. With this feature the whole process

Archive Pipeline

This pipeline contains two steps, the first step zip all working directories, which are older than the configured period to a .tar.gz. The second step deletes all working directories, which are older than the configured period.

NOTE

Components & Concept

Please refer to Process Pipelines for technical insights.

Configuration

The configurations for the periods can be done in Magento's admin UI Stores > Configuration > Pacemaker > Pipeline Settings

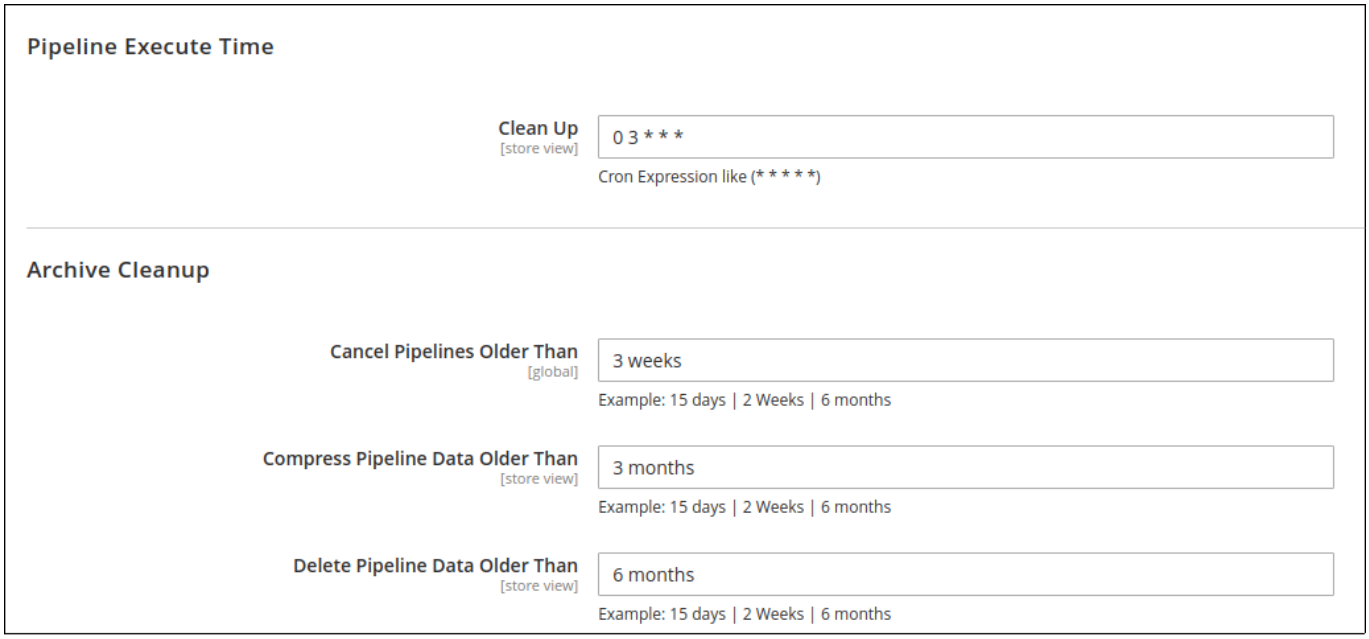


Figure 5. Configuration in Magento Backend

Archive Pipeline

The configuration Pipeline Execute Time > Clean Up contains a cron expression value, which defines when the clean up (archive pipeline) for the working directories should run. By default, this value is set to 3 o'clock every day.

The configuration Archive Cleanup > Compress Pipeline Data Older Than defines the period for the file compression. By default, this value is 3 months.

The configuration Archive Cleanup > Delete Pipeline Data Older Than defines the period for the file compression. By default, this value is 6 months.

Pipeline Cancel

To avoid never-ending pipelines (this depends on the implementation of custom pipelines), there is a cancel routine, which cancels all pipelines older than the configured time in Archive Cleanup > Cancel Pipelines Older Than. This value is 3 weeks by default. This pipeline contains two steps, the first step zip all working directories, which are older than the configured period to a .tar.gz. The second step deletes all working directories, which are older than the configured period.

NOTE

Components & Concept

Please refer to Process Pipelines for technical insights.

Configuration

The configurations for the periods can be done in Magento's admin UI Stores > Configuration > Pacemaker > Pipeline Settings

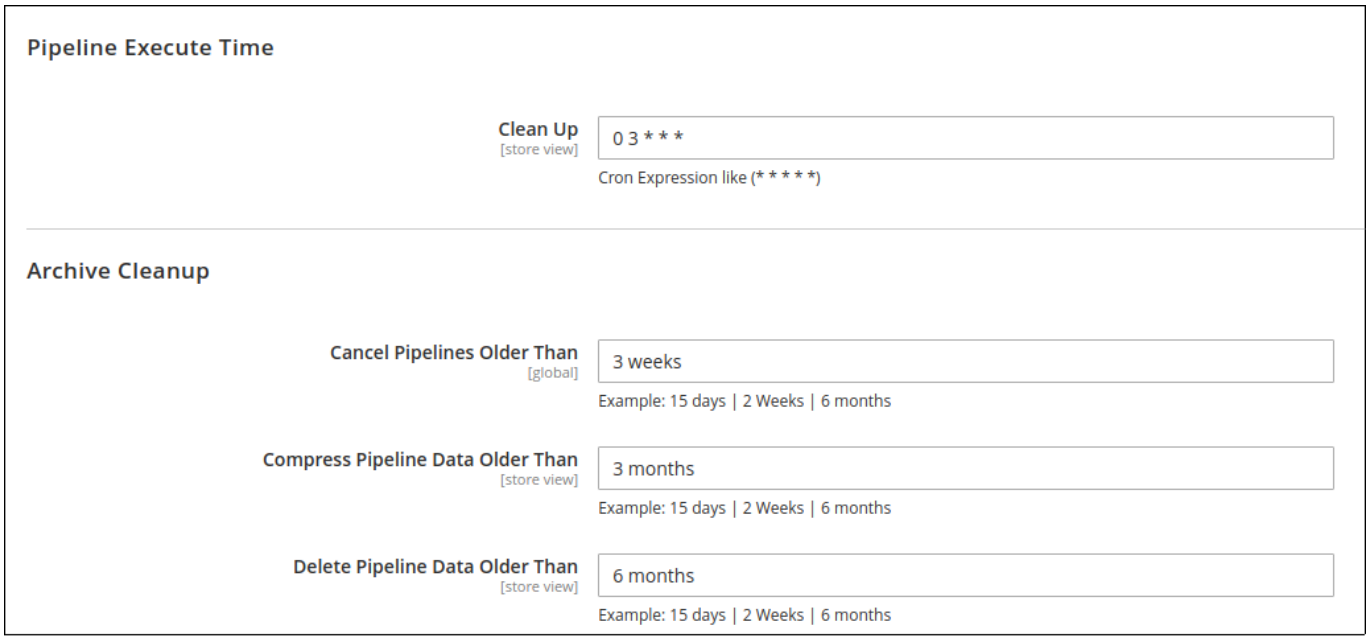


Figure 6. Configuration in Magento Backend

Archive Pipeline

The configuration Pipeline Execute Time > Clean Up contains a cron expression value, which defines when the clean up (archive pipeline) for the working directories should run. By default, this value is set to 3 o'clock every day.

The configuration Archive Cleanup > Compress Pipeline Data Older Than defines the period for the file compression. By default, this value is 3 months.

The configuration Archive Cleanup > Delete Pipeline Data Older Than defines the period for the file compression. By default, this value is 6 months.

Pipeline Cancel

To avoid never-ending pipelines (this depends on the implementation of custom pipelines), there is a cancel routine, which cancels all pipelines older than the configured time in Archive Cleanup > Cancel Pipelines Older Than. This value is 3 weeks by default.

Catalog Import

The catalog import is one of the predefined pipelines (What is a pipeline?), which you can use after installing Pacemaker. There are sample files, which can be used as a template for your data files or testing purposes (see Run your first predefined import jobs). On this page, you will read how it works, how to configure, and how to customize this pipeline.

NOTE

Components & Concept

Please refer to Import Processes for technical insights.

Pipeline Definition

The catalog import pipeline is designed to import the whole catalog data at once. Therefore attributes, attribute-sets, categories and products need to be handled in the same pipeline. However, all of these import steps are optional and it depends on the given files, whether this data will be imported or not.

The import pipeline contains the following steps:

Stage	Step	Description
Prepare	move_files	Move import files to the working directory of the current pipeline
Transformation	product_transformation	This step has a dummy executor in default and is designed to customize for mapping and transformation purpose
Pre-Import	index_suspender_start	Activates delta index suspending to avoid cron based re-indexing during the import
Attribute Set Import	attribute_set_import	Create/Update attribute sets
Attribute Import	attribute_import	Create/Update attributes
Category Import	category_import	Create/Update categories
Product Import	product_import	Create/Update products
Post-Import	index_suspender_stop	Disable suspending of delta indexers

Configuration

In Magento's backend (admin-ui) you'll find settings for the catalog import under following path: Stores > Configuration > Pacemaker > Import > Catalog Import

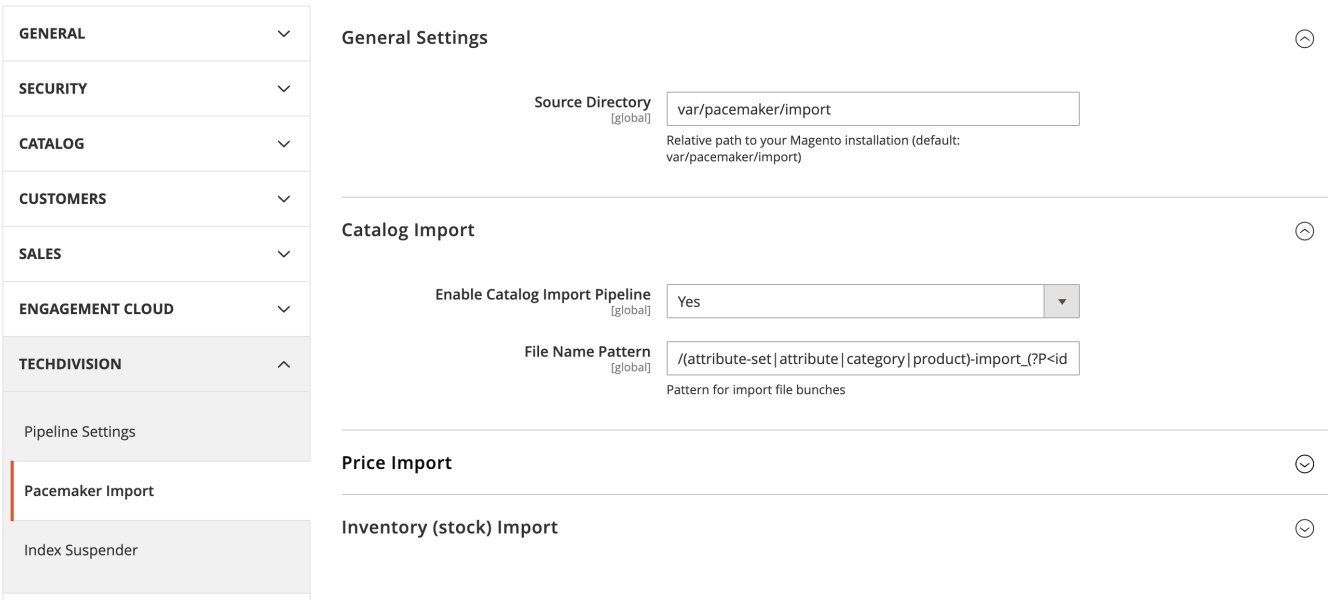


Figure 7. Configuration in Magento Backend

Configuration	Description	Default Value
General Settings > Source Directory	Defines the source directory for import files. This directory will be observerd by Pacemaker in oder to intialize a import pipeline. **This setting is for all pacemaker imports.**	var/pacemaker/import
Catalog Import > Enable Catalog Import Pipeline	Active toggle for the source directory observer for catalog import.	Yes
Catalog Import > File Name Pattern	Regular expression, which defines the source file name for source directory observer.	/(attribute-set|attribute|category|product)-import_(?P<identifier>[0-9a-z\-]*)([_0-9]*?).(csv|ok)/i

Price Import

The price import is one of the predefined pipelines (What is a pipeline?), which you can use after installing Pacemaker. There are sample files, which can be used as a template for your data files or testing purposes (see Run your first predefined import jobs). On this page, you will read how it works, how to configure, and how to customize this pipeline.

NOTE

Components & Concept

Please refer to Import Processes for technical insights.

Pipeline Definition

The import pipeline contains the following steps:

Stage	Step	Description
Prepare	move_files	Move import files to the working directory of the current pipeline
Transformation	price_transformation	This step has a dummy executor in default and is designed to customize for mapping and transformation purpose
Pre-Import	index_suspender_start	Activates delta index suspending to avoid cron based re-indexing during the import
Import	price_import	Update price data
Post-Import	index_suspender_stop	Disable suspending of delta indexers

Configuration

In Magento's backend (admin-ui) you'll find settings for the price import under following path: Stores > Configuration > Pacemaker > Import > Price Import

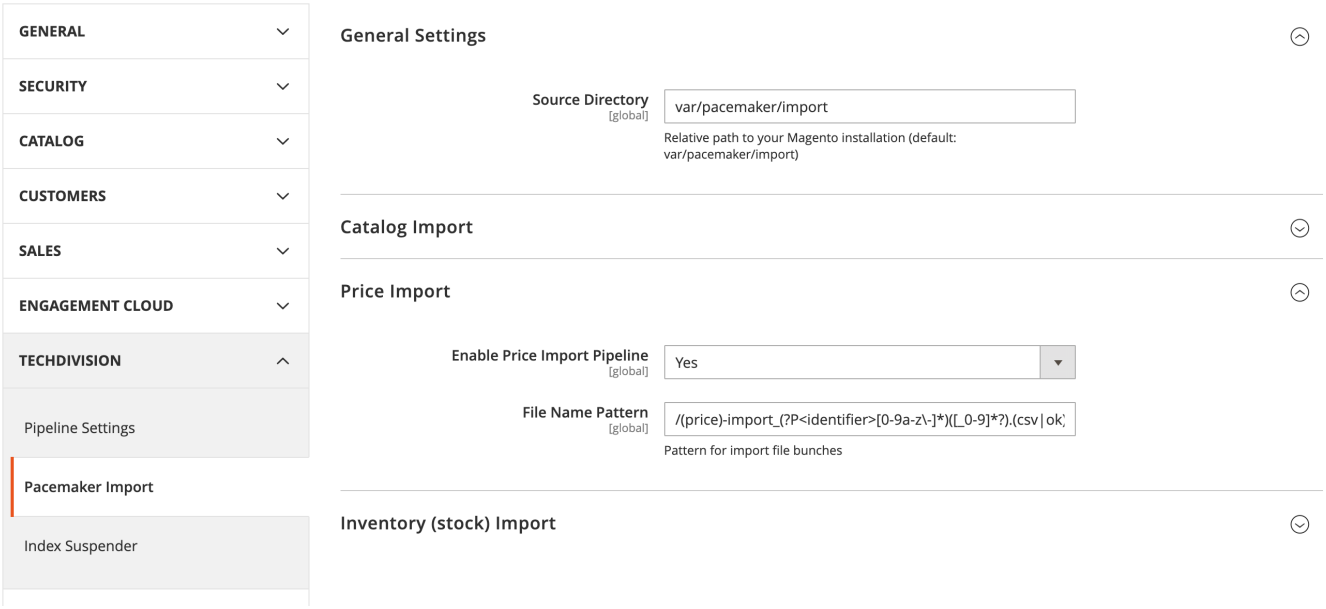


Figure 8. Configuration in Magento Backend

Configuration	Description	Default Value
General Settings > Source Directory	Defines the source directory for import files. This directory will be observerd by Pacemaker in oder to intialize a import pipeline. **This setting is for all pacemaker imports.**	var/pacemaker/import
Price Import > Enable Price Import Pipeline	Active toggle for the source directory observer for price import.	Yes
Price Import > File Name Pattern	Regular expression, which defines the source file name for source directory observer.	/(price)-import_(?P<identifier>[0-9a-z\-]*)([_0-9]*?).(csv|ok)/i

Inventory Import

The inventory import is one of the predefined pipelines (What is a pipeline?), which you can use after installing Pacemaker. There are sample files, which can be used as a template for your data files or testing purposes (see Run your first predefined import jobs). On this page, you will read how it works, how to configure, and how to customize this pipeline.

NOTE

Components & Concept

Please refer to Import Processes for technical insights.

Pipeline Definition

The import pipeline contains the following steps:

Stage	Step	Description
Prepare	move_files	Move import files to the working directory of the current pipeline

Stage	Step	Description
Transformation	inventory_transformation	This step has a dummy executor in default and is designed to customize for mapping and transformation purpose
Pre-Import	index_suspender_start	Activates delta index suspending to avoid cron based re-indexing during the import
Import	inventory_import	Update inventory data
Post-Import	index_suspender_stop	Disable suspending of delta indexers

Configuration

In Magento's backend (admin-ui) you'll find settings for the price import under following path: Stores > Configuration > Pacemaker > Import > Inventory (stock) Import

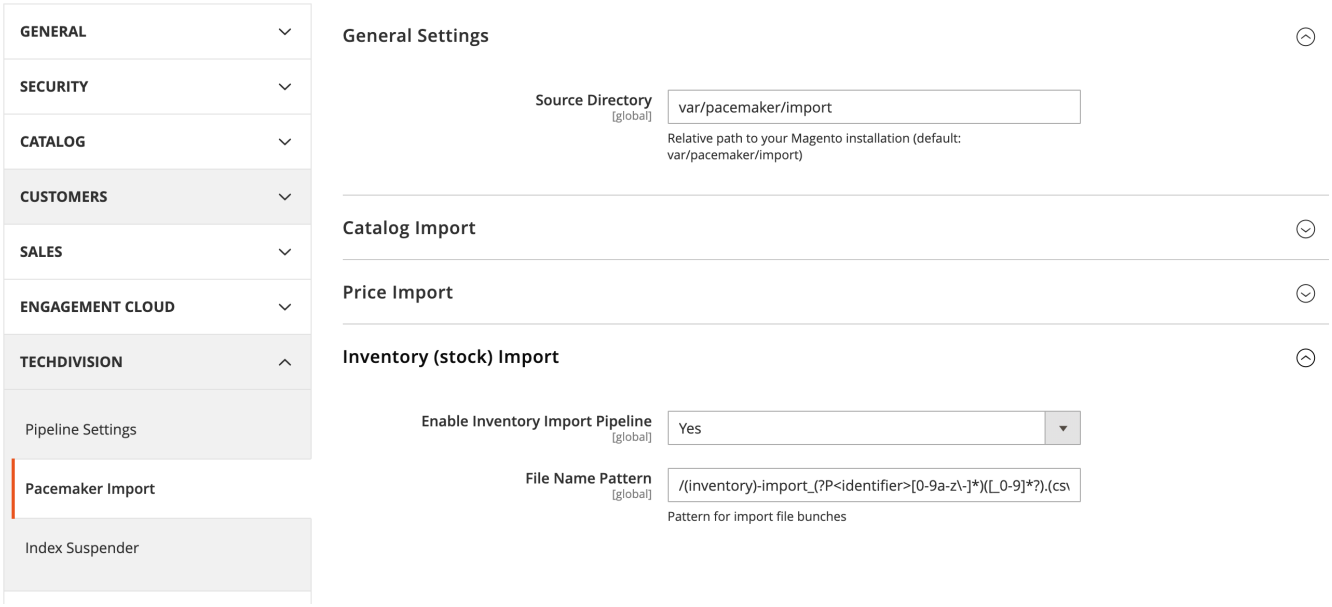


Figure 9. Configuration Inventory Import

Configuration	Description	Default Value
General Settings > Source Directory	Defines the source directory for import files. This directory will be observerd by Pacemaker in oder to intialize a import pipeline. **This setting is for all pacemaker imports.**	var/pacemaker/import
Inventory (stock) Import > Enable Inventory Import Pipeline	Active toggle for the source directory observer for price import.	Yes
Inventory (stock) Import > File Name Pattern	Regular expression, which defines the source file name for source directory observer.	/(inventory)-import_(?P<identifier>[0-9a-z\-]*)([_0-9]*?).(csv|ok)/i

Order Export

NOTE

Components & Concept

Please refer to Order Workflow for technical insights.

Once this feature is enabled (configurable) an export pipeline will be created for each order, which matches the filter conditions (configurable). Depending on the following settings the pipeline contains 2, 3 or 4 steps.

Step	Description
transform	This step transforms the Magento order into a target format.
transport	This step transports the transformed order to the target destination.
handle_response	This step exists, if a response handler is configured. It is designed to handle async responses from ERP/OMS.
handle_notification	This step exists, if a notification handler is configured. This step can be used to send a order confirmation mail after the ERP/OMS accepts the order.

The configurations for the order export can be done in Magento's admin UI Stores > Configuration > Pacemaker > Order Workflow

Configuration

General Settings

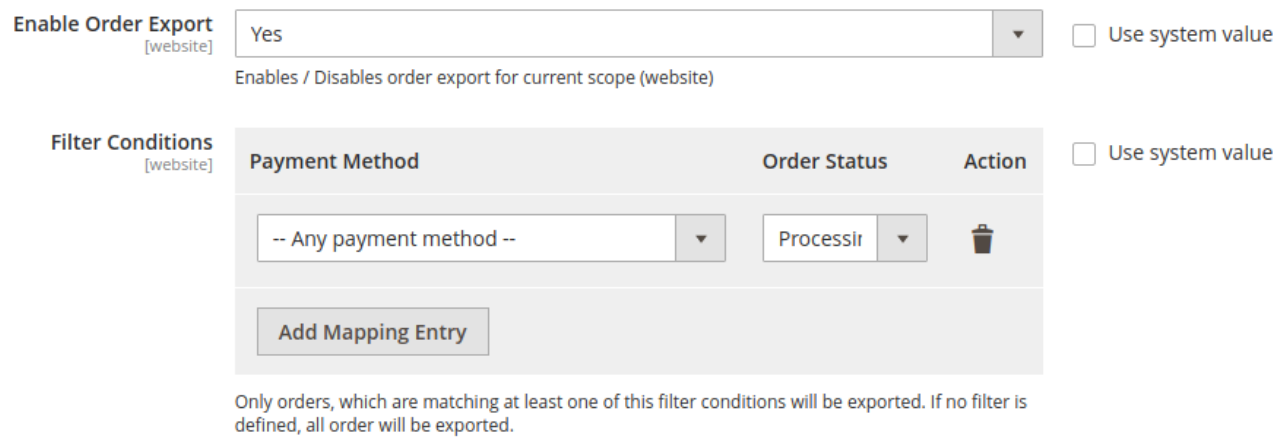


Figure 10. Order Export Settings

Configuration	Description	Default Value
Enable Order Export	Feature toggle for the whole export pipeline.	No

Configuration	Description	Default Value
Filter Conditions	This mapping grid allows to define conditions, when an order should be exported. It is a combination of payment method and order status. Once one of the defined conditions matches the order the export pipeline will be triggered for this order. Leave the configuration grid blank to export all orders without filtering.	payment method: any | status: processing

Export Format

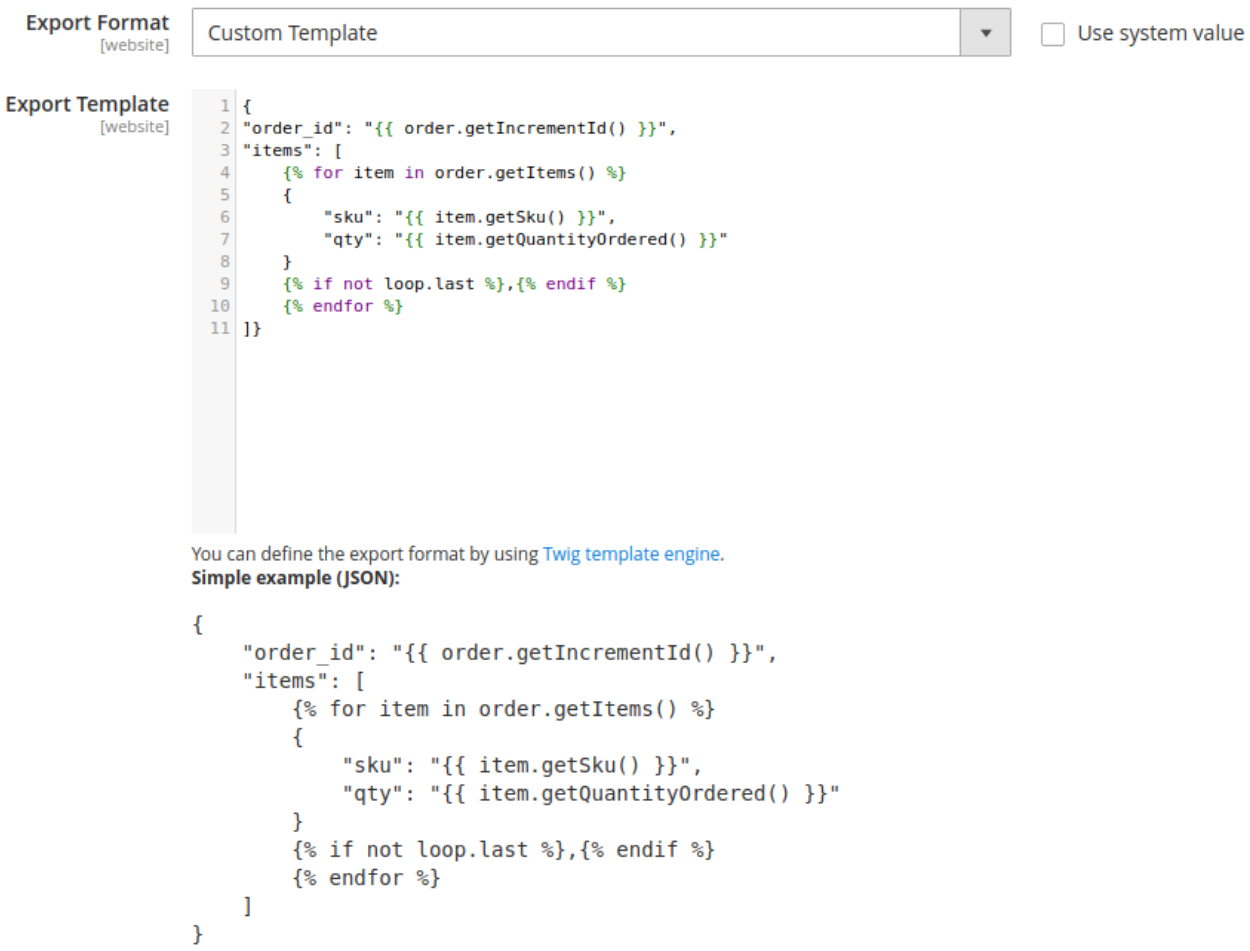


Figure 11. Order Export Format

Configuration	Description	Default Value
Export Format	This setting defines, which formater should be used to transform the order to the target format. The formats are extendable, please refer to Add custom export format chapter for details. By default there is the **Custom Template** formater available.	Custom Template

Configuration	Description	Default Value
Export Template	This option appears if the value **Custom Template** is chosen for Export Format. Here you can create a template for the target order format by using the twig templating syntax. Please refer to Twig Documentation. The only variable, which is passed to the renderer is order, which is an instance of the Magento's OrderInterface	

Transport Adapter

Figure 12. Transport Adapter

Configuration	Description	Default Value
Transport Adapter	This setting defines, which adapter should be used to transport the order to the target destination. The adapters are extendable, please refer to Add custom transport adapter chapter for details. By default there is the **Local Filesystem** adapter available.	Local Filesystem
Local Filesystem Target Directory	This option appears if the value **Local Filesystem** is chosen for Transport Adapter. Here you can specify the target destination within the filesystem. The path can be absolute or relative to the Magento root directory.	var/pacemaker/export
Local Filesystem Filename Pattern	This option appears if the value **Local Filesystem** is chosen for Transport Adapter. Here you can specify the naming of the export file. There are following placeholders available: INCREMENT_ID,ENTITY_ID,WEBSITE_CODE,STORE_CODE#	order-INCREMENT_ID.json

Response Handler

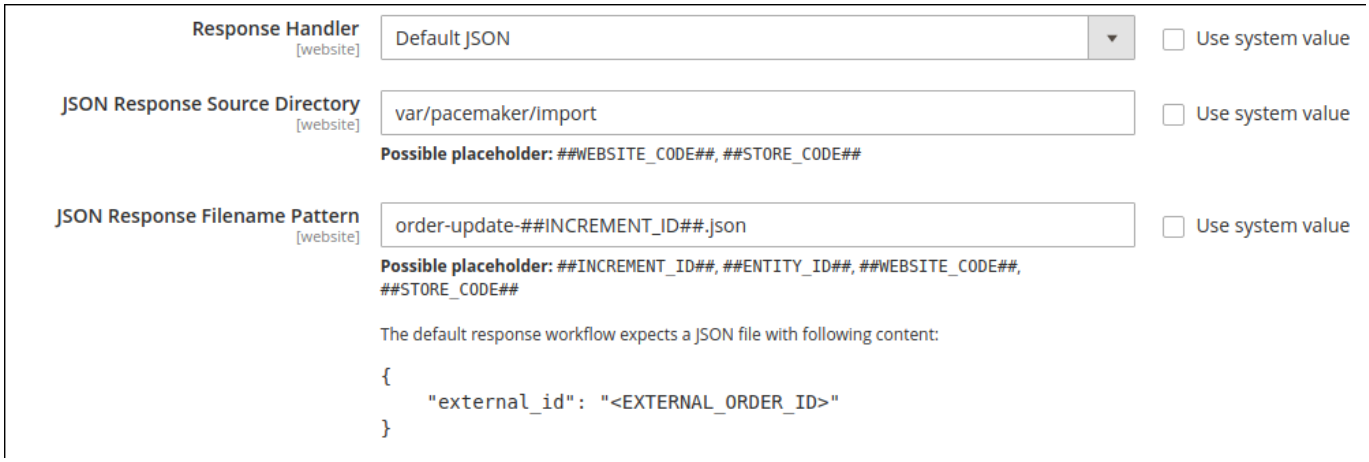


Figure 13. Response Handler

Configuration	Description	Default Value
Response Handler	This setting defines, which handler should be used to handle the response from the target system. This handler is optional. If **No response handler** is chosen, there will no corresponding step be spawned within the export pipeline. The handlers are extendable, please refer to Add custom response handler chapter for details. By default there is the **Default JSON** adapter available.	No response handler
JSON Response Source Directory	This option appears if the value **Default JSON** is chosen for Response Handler. Here you can specify the target directory, which should be observed for response files.	var/pacemaker/import
JSON Response Filename Pattern	This option appears if the value **Default JSON** is chosen for Response Handler. Here you can specify the naming of the response file.	order-update-INCREMENT_ID.json

Notification Handler

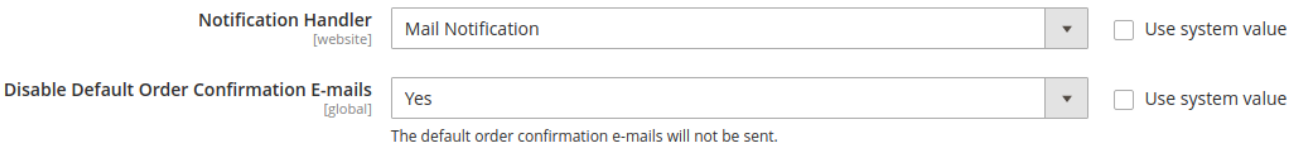


Figure 14. Notification Handler

Configuration	Description	Default Value
Notification Handler	This setting defines, which handler should be used to handle the notification step, which is the last step in the pipeline. This handler is optional. If **No notification handler** is chosen, there will no corresponding step be spawned within the export pipeline. The handlers are extendable, please refer to Add custom notification handler chapter for details. By default there is the **Default JSON** adapter available.	No notification handler
Disable Default Order Confirmation E-mails	If you want notify customer about a successful purchase after this sale was approved from the ERP/OMS, you probably want to disable the default order confirmation mail.	No

Line Item Extension

This component provides extension attributes for order items. It can be enabled independent to the order export component. It generates line item numbers, which are unique within the dedicated order. This is something Magento does not have by default but is required sometimes by ERP/OMS.

Configuration

The configurations for the line item extension can be done in Magento's admin UI Stores > Configuration > Pacemaker > Order Workflow

General Settings

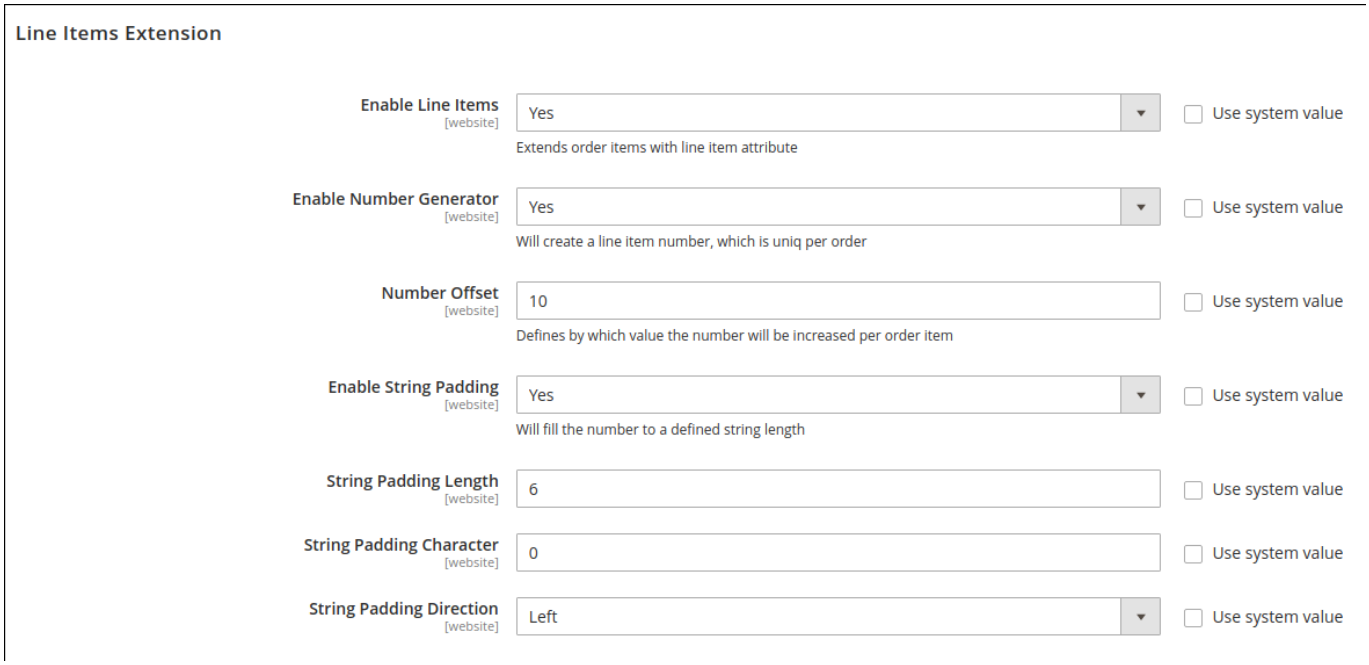


Figure 15. Line Item Extension Settings

Configuration	Description	Default Value
Enable Line Items	Feature toggle for the whole module. If enabled the extension attributes will be loaded for each order item.	No
Enable Number Generator	If enabled a plugin will generate unique line item numbers per order every time a order is placed	No
Number Offset	This option appears if Enable Number Generator is set to Yes. The value defines the count sequence for line item numbers. For example the value 3 would mean the numbers will be 3, 6, 9, 12, and so on.	1

String Padding

Configuration	Description	Default Value
Enable String Padding	If enabled the function str_pad will be applied to the generated number. Please refer to the function description in PHP manual.	No
String Padding Length	Defines the number of chars to which the number will fill up	6
String Padding Character	The character, which will be used to fill up the padding	0
String Padding Direction	In which direction the padding should be applied	Left

Example

For SAP systems we usually use the settings:

Configuration	Value
Enable Line Items	Yes
Enable Number Generator	Yes
Number Offset	10
Enable String Padding	Yes
Enable String Length	6
Enable String Character	0
Enable String Direction	LEFT

This leads in line item numbers like 000010, 000020, 000030, …

Components & Concepts

Process Pipelines

One of the central components of the Pacemaker is the ProcessPipelines module. The concept behind this module is the Pipeline Design Pattern.

The Pattern

To describe this pattern in short: You need to split up each process into multiple stages. Each stage has **at least one step**. While the stages have a clear sequence, the steps inside these stages can run in parallel or in random order, since they are not depending on each other, but on the stage.

This requires a decoupling of the execution and the organization of these steps. Usually, a runner-driven architecture is used in order to fulfill this requirement (SPMD). You'll find such integrations in build- and deployment tools like Jenkins or GitLab.

Realization

In the Get Started section, you already touched the both major parts of ProcessPipelines, while configuring the heartbeat cron job (which is the organizational unit) and configuring the pipeline runners (which are the execution unit).

Heartbeat and Runner

In the following image, you can see the responsibilities of both units.

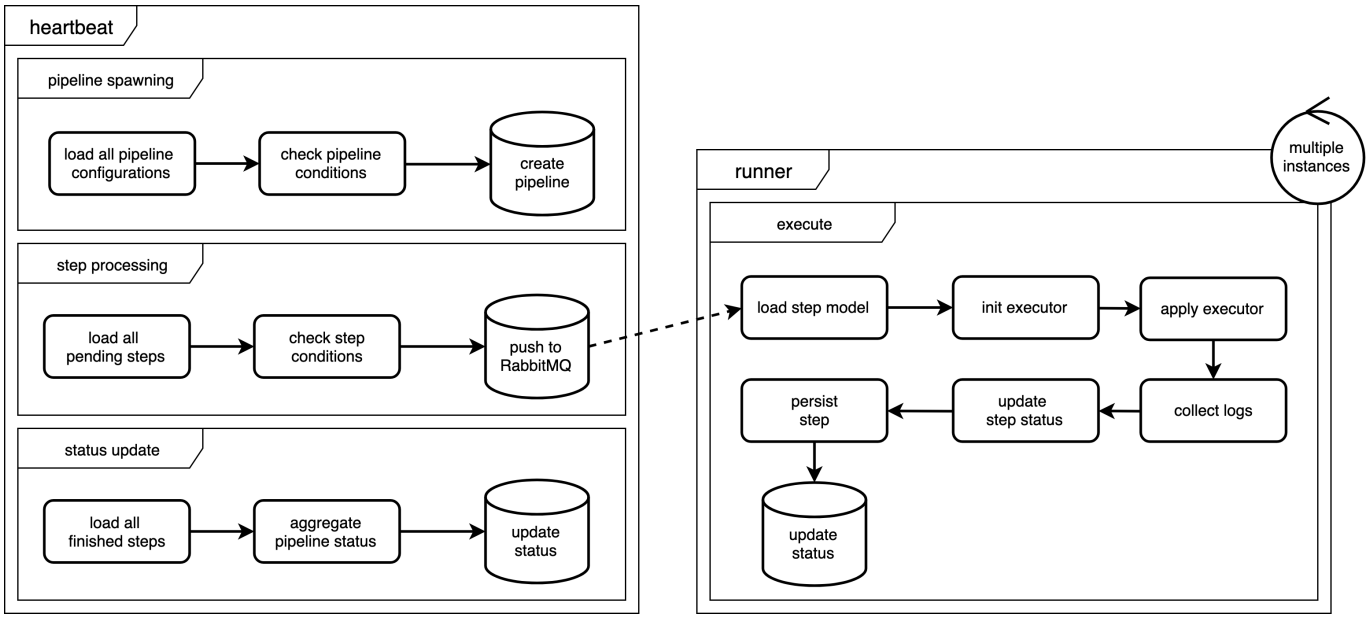


Figure 16. Heartbeat and Runner

The heartbeat is responsible for the initialization of new pipelines and publishing of messages for the runner if a step can be executed. And since the status of a pipeline is an aggregation of all status of its steps the heartbeat is also responsible for this aggregation.

The runner executes each step, which is in the queue and does not take care of its status or anything else. This gives us great scalability since these runners can run on different or multiple machines.

Configuration and Process Execution

The pipelines are usually configured in XML files, which are merged between the modules. This happens in the same way as you already know from Magento's configuration XMLs (e.g. config.xml, di.xml).

TIP

Pipeline Initializer

Pipelines can also be defined by code (without XML files) if you need to create pipelines in a dynamic way. Read Pipeline Initializer documentation for details.

- Please refer to the How to Configure a ProcessPipeline page

A pipeline configuration has **at least one** condition, which will be checked periodically by the heartbeat. If a pipeline is ready to be spawned the heartbeat creates a new pipeline in the database.

- Please refer to the How to create a pipeline condition page

Every step within a pipeline has **at least one** condition, which will be checked periodically by the heartbeat. If a step is ready to be executed the heartbeat pushes an execution message to the RabbitMQ.

- Please refer to the How to create a step condition page

Every message in the RabbitMQ will be executed by the next free runner. **This means that the amount of runners is at the same time the limiting factor for parallel execution.** The runner instantiates and executes executor class.

- Please refer to the How to create an executor page

This component was designed to instantiate new pipelines, which are not part of the own code or not defined in an XML file at all (e.g. you need a flexible count of steps).

How it works?

This module integrates into the heartbeat execution. Every time the heartbeat is executed, a data fetcher chain (TechDivision\PacemakerPipelineInitializer\Model\InitializationDataFetcherChain), which is an instance of TechDivision\PacemakerPipelineInitializer\Api\InitializationDataFetcherInterface collects the data from all registered data fetcher. And initiates pipelines depending on the given data.

Interfaces

InitializationDataFetcherInterface

```
interface InitializationDataFetcherInterface
{
    /**
     * @return InitializationDataInterface[]
     */
    public function execute(): array;
}
```

InitializationDataFetcherInterface

```
interface InitializationDataInterface
{
    /**
     * @return array
     */
    public function getPipelineConfiguration(): array;

    /**
     * @param array $pipelineConfiguration
     * @return void
     */
    public function setPipelineConfiguration(array $pipelineConfiguration): void;

    /**
     * @return array
     */
    public function getArguments(): array;

    /**
     * @param array $arguments
     * @return void
     */
    public function setArguments(array $arguments): void;
}
```

When to use this component?

You should use this component if you need to create pipelines dynamically. If the steps of your pipeline are fix, you should use the XML configuration.

With this module, you can create a pipeline by defining a PHP array or by loading an existing pipeline configuration from XML as an array and passing it to the data fetcher chain.

You'll find an example for an implementation of this approach in techdivision/pacemaker-import-base package TechDivision\PacemakerImportBase\Model\ImportFilesDataFetcher

Import Processes

General thoughts

There are several ways how to put data into Magento. One of these ways is the already existing flat-file import of Magento. Another way is to use WebAPI, which supports also bulk and asynchronous operations. Why do we need Pacemaker?

There are several pros and cons for the existing solutions as well as there are pros and cons for using Pacemaker. Let's compare them.

Magento's Flat File Import

Magento provides an implementation for flat file import. The file needs to be uploaded via the admin UI. The synchronous process validates and persists the data from the uploaded CSV file. In comparison to the WebAPI this approach can handle big amount of data pretty fast, but you need to configure your webserver not to abort this long-running request and accept big data POST methods.

The import will be executed instantly. Depending on the load of your server this could cause performance issues (for sure this depends on your infrastructure). The import will cause a data re-indexing if your indexer mode is "on schedule", which also harms server resources. Indexer mode "on save" will require a manual re-indexing, which also needs to be done after the import. The re-indexing will cause a cache invalidation. However, all these resource-consuming processes should be done in a controlled way and time frame.

Magento's WebAPI

WebAPI is a good way to read, add and update data in Magento. But the WebAPI is not designed to handle a huge amount of data. The bulk and asynchronous WebAPI can handle big data, but not very perform. Depending on your infrastructure the update of a single product could take 1-3 seconds. For a catalog with 20k products, this would lead in processing time between 5 and 8 hours, but we face often shops with several millions of products.

The Pacemaker Import Approach

The basic idea behind Pacemaker is to decouple the third party system and user interactions with resource-consuming processes on Magento side. Therefore we are using Process Pipelines. To handle a huge amount of data we are using M2IF, which is a powerful import framework and many times faster than Magento's flat file import.

But the import process at all is not just to put the data into the database. The pipelines take also aware of fetching, transforming and persisting the data as well as the post-import processes like re-indexing and cache invalidation. All these operations are parts of a big process chain and can be executed synchronous and asynchronous depending on your current system state.

Common Process Design

Usually, the import processes are designed as following

1. Trigger import process chain
   e.g. periodically (daily), via web service notification, by observing the file system, etc.
2. Fetching data from source
   e.g. reading files, calling WebAPIs, etc.
3. Transforming data to target format
   e.g. executing own scripts, use external libraries, etc.
4. Execute the import
   e.g. running M2IF library, etc.
5. Run indexers and invalidate caches
   e.g. using Magento's APIs

Pacemaker's import pipelines by default provide an observer for the local filesystem, which triggers the pipeline initialization. The transformation step is for customization, since it depends on your data source, whether the files need to be transformed or not. Please refer to How to extend > Transform foreign import source to learn how to customize this step. By default, M2IF library is running the import and there are executors for Magento's indexers and cache invalidation.

Import Files Observer

Since Pacemaker 1.2 the observation of the filesystem is designed as an own pipeline any more. It is part of the heartbeat process now. There it uses the techdivision/pacemaker-pipeline-initializer package to trigger the import pipelines, once the required files are present in the file system. Please refer to Components & Concepts > Pipeline Initializer for details.

What is a file bunch?

Since Pacemaker is using M2IF it is possible to split all import files into multiple files. And because Pacemaker is running attribute-set, attribute, category and product import in one pipeline a bunch could grow to a big number of files. All these files need the same **identifier** in the file name. This identifier is defined in the **File Name Pattern** configuration within this part of the regular expression (?P<identifier>[0-9a-z\-]*).

According to the default expression, the filenames need to be in the following pattern: <IMPORT_TYPE>-import_<BUNCH_IDENTIFIER>_<COUNTER>.<SUFFIX>. There are example files provided in Pacemaker packages, please refer to Run your first predefined import jobs. **Of course, you can change the expression if necessary, just take care to define an identifier within the pattern.**

Examples

The following files would result in **one** import pipeline because the identifier is the same for all files. Also, only the steps attribute and product import would be executed. Attribute-set and category import would be skipped because there are no files given.

```
- attribute-import_20190627_01.csv
- attribute-import_20190627.ok
- product-import_20190627_01.csv
- product-import_20190627_02.csv
- product-import_20190627_03.csv
- product-import_20190627.ok
```

The following files would result in **two** import pipelines, while the first bunch import all entities and the second bunch imports only product data.

```
- attribute-set-import_20190627-1_01.csv
- attribute-set-import_20190627-1.ok
- attribute-import_20190627-1_01.csv
- attribute-import_20190627-1.ok
- category-import_20190627-1_01.csv
- category-import_20190627-1.ok
- product-import_20190627-1_01.csv
- product-import_20190627-1_02.csv
- product-import_20190627-1_03.csv
- product-import_20190627-1.ok
- product-import_20190627-2_01.csv
- product-import_20190627-2_02.csv
- product-import_20190627-2_03.csv
- product-import_20190627-2.ok
```

How to Extend / Examples

Process Pipelines

How to configure a pipeline

TIP

Following pages could also be interesting for you:

- How to add steps to an existing pipeline
- How to change the executor for an existing pipeline step

To define a new pipeline, you need to add a pipeline.xml file into the etc directory of your module.

Here is some sample content for a pipeline.xml file:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="my_custom_pipeline" description="Some description" use-working-
directory="true">
        <conditions>
            <pipeline_condition
type="MyCompany\MyModule\Helper\Condition\Pipeline\CheckSomeThing" description="Some
description"/>
        </conditions>
        <step name="my_first_step"
executorType="MyCompany\MyModule\Model\Executor\DoSomeThing" sortOrder="10" description="" >
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit5"
description="Try step up to 5 times"/>
            </conditions>
        </step>
        <step name="my_second_step"
executorType="MyCompany\MyModule\Model\Executor\DoSomeMoreStuff" sortOrder="20"
description="" >
            <arguments>
                <argument key="some_key" value="some_value" />
            </arguments>
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Run after the first step"/>
                <step_condition
type="MyCompany\MyModule\Helper\Condition\Step\CheckSomeThing" description="Check
something..."/>
```

```
            </conditions>
        </step>
        <step name="my_third_step"
executorType="MyCompany\MyModule\Model\Executor\DoClearingStuff" sortOrder="30"
description="" runAlwaysStep="true">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try step up to 1 times"/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsFinished"
description="Run always when one step started"/>
            </conditions>
        </step>
    </pipeline>
</config>
```

Description for XML nodes and attributes

Node	Description
pipeline	Definition of a new pipeline configuration. You can overwrite or extend existing pipelines by using the same name
-- name	**Required**, string; Unique identifier, which is used for instantiating a pipeline by CLI and can be used for condition rules, etc.
-- description	**Optional**, string; Description text for the pipeline. Will be displayed in UI and CLI as 'name' of the pipeline.
-- use-working-directory	**Optional**, boolean; If defined a working directory will be autogenerated while pipeline instantiation. The path for this directory is configurable and can be retrieved within an executor by $step->getWorkingDir().
pipeline/conditions	**Optional**; One or multiple conditions which should be true to cause a new pipeline execution
pipeline/conditions/pipeline_condition	Configuration of a pipeline condition
-- type	**Required**, string; Defines the class, which will be executed to run the condition check.
-- description	**Optional**, string; Description text for given condition. Will be displayed on UI, logs, CLI, etc.
pipeline/step	Each pipeline has at least one step. The step describes which executor should be run once the step conditions are true
-- name	**Required**, string; Unique identifier, which is used for running/executing the step by CLI and can be used for condition rules, etc.
-- executorType	**Required**, string; Defines the class, which will be executed to run the step.

Node	Description
-- sortOrder	**Required**, string; Defines the sorting of steps. Useful for resorting to the steps by other modules/extensions.
-- description	**Optional**, string; Description text for given step. Will be displayed on UI, logs, CLI, etc.
-- runAlwaysStep	**Optional**, boolean; If this flag is set, the step will be executed in any case. Even if the pipeline is canceled or abandoned. Kind of 'finally' step.
pipeline/step/arguments	**Optional**; Holds one or many executor arguments
pipeline/step/arguments/argument	Step executor argument
-- key/value	**Required**, string; Since arguments are array at all, both arguments represent the key and value of each array node.
pipeline/step/conditions	One or multiple conditions which should be true to cause the step execution.
pipeline/step/conditions/step_condition	Configuration of a step condition
-- type	**Required**, string; Defines the class, which will be executed to run the condition check.
-- description	**Optional**, string; Description text for given condition. Will be displayed on UI, logs, CLI, etc.

NOTE There are already some default condition and executors available.

How to create a pipeline condition

Pipeline Conditions

Each pipeline has at least one condition, which implements the TechDivision\ProcessPipelines\Api\PipelineConditionInterface interface.

Existing Conditions

There are some ready-to-use conditions.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn

Disables auto spawning of given pipeline. Pipelines, which use this condition can not be spawned by heartbeat.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoSiblingInProgress

Disable spawning of the given pipeline, while this pipeline is already in progress.

Usage

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline">
```

```
        <conditions>
            <pipeline_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn"
description="Disable auto spawning"/>
        </conditions>
        ...
    </pipeline>
</config>
```

Provide arguments from condition to steps

Since version 1.1.0 there is an additional interface (TechDivision\ProcessPipelines\Api\ArgumentProviderInterface) for conditions, which enables you to provide arguments from the pipeline condition all pipeline steps.

Reasons

Imagine you have a condition, which checks the file system for a specific file pattern. If the file is present, you need to run a new pipeline for exactly this file. Therefore you need to provide the information to your steps, which should handle the file.

Usage Examples

```
<?php

use TechDivision\ProcessPipelines\Api\PipelineConditionInterface;
use TechDivision\ProcessPipelines\Api\ArgumentProviderInterface;

class FileExistsCondition implements PipelineConditionInterface, ArgumentProviderInterface
{
    const SOME_MAGIC_FILE_PATTERN = '...';

    public function isReady(array $pipelineConfiguration)
    {
        $this->searchForFiles(self::SOME_MAGIC_FILE_PATTERN);
        return $this->hasFiles();
    }

    public function getArguments()
    {
        return $this->getFiles();
    }
}
```

Virtual Conditions

By using the virtualType feature of Magento's DI it is easy to create new conditions with some new params. Therefore there are some "template" conditions.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\CronExpression

Defines execution time for a pipeline using regular cron expression.

TechDivision\ProcessPipelines\Helper\Condition\Pipeline\ConfigurableCronExpression

Defines execution time for a pipeline using regular cron expression, while this expression can be configured in Magento admin.

Usage

Create virtual spawn conditions using di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType name="MyCompany\MyModule\Virtual\Condition\EveryNight"
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\CronExpression">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="cron_expression" xsi:type="string">0 2 * * *</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\FullImport"
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\ConfigurableCronExpression">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="config_path"
xsi:type="string">my_module/crontab/full_import</item>
            </argument>
        </arguments>
    </virtualType>
    ...
</config>
```

Use the virtual condition in pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Some example pipeline">
        <conditions>
            <pipeline_condition type="MyCompany\MyModule\Virtual\Condition\EveryNight"
description="Run once in the night"/>
        </conditions>
        ...
    </pipeline>
    <pipeline name="another_example_pipeline" description="Some example pipeline">
        <conditions>
            <pipeline_condition type="MyCompany\MyModule\Virtual\Condition\FullImport"
description="Customer defined cron expression"/>
```

```
        </conditions>
        ...
    </pipeline>
</config>
```

How to create a step condition

Step Conditions

Each step can have conditions, which implement the TechDivision\ProcessPipelines\Api\StepConditionInterface interface.

Existing Conditions

There are some ready-to-use conditions.

TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted
Run the given step only if the previous steps are successfully finished.

TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsFinished
Run the given step only if the previous steps are not in a pending, running or enqueued state.

Usage

NOTE See How to configure a pipeline for more details about the XML structure.

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline">
        ...
        <step name="my_third_step"
executorType="MyCompany\MyModule\Model\Executor\DoSomeThing" sortOrder="1522" description=""
runAlwaysStep="">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Run only after previous"/>
            </conditions>
        </step>
        ...
    </pipeline>
</config>
```

Virtual Conditions

Template Conditions

By using the virtualType feature of Magento's DI it is easy to create new conditions with some new params. Therefore there are some "template" conditions.

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit
Limitation of retries for given steps.

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts
Timeout between retries.

TechDivision\ProcessPipelines\Helper\Condition\Step\WaitForStepIsNotRunning
Disables execution of a step while the defined steps (even in other pipelines) are running. For example: only one import process at the same time possible…

TechDivision\PacemakerImportBase\Model\Condition\Step\NoConflictingStepsInProcess
The list of conflicting step names can be set via DI. Verifies that this steps are not running or enqueued. This check works over all existing pipelines.

TechDivision\PacemakerImportBase\Model\Condition\Step\IsStageReady
This condition was designed to verify, whether a stage is ready to be executed.

Ready to use virtual conditions

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1
Attempts limit 1

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit2
Attempts limit 2

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit5
Attempts limit 5

TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit10
Attempts limit 10

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes5
5 minutes delay between attempts

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes15
15 minutes delay between attempts

TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes30
30 minutes delay between attempts

Usage

TIP In following example module virtual conditions are used to allow parallel execution.

Create virtual spawn conditions using di.xml

```
<?xml version="1.0"?>
```

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType name="MyCompany\MyModule\Virtual\Condition\AttemptsLimit42"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="limit" xsi:type="string">42</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\TimeBetweenAttempts\Minutes42"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="minutes" xsi:type="string">42</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\AllDownloadsAreReady"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="step_filter"
xsi:type="string">download_media_step,download_csv_step</item>
            </argument>
        </arguments>
    </virtualType>
    <virtualType name="MyCompany\MyModule\Virtual\Condition\NoImportIsRunning"
type="TechDivision\ProcessPipelines\Helper\Condition\Step\WaitForStepIsNotRunning">
        <arguments>
            <argument name="data" xsi:type="array">
                <item name="step_names"
xsi:type="string">import_stock_step,import_products_step</item>
            </argument>
        </arguments>
    </virtualType>
    ...
</config>
```

Use the virtual condition in pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline">
        ...
        <step name="some_step" executorType="MyCompany\MyModule\Model\Executor\DoSomeThing"
```

```
sortOrder="10" description="" >
            <conditions>
                <step_condition type="MyCompany\MyModule\Virtual\Condition\AttemptsLimit42"
description="Retry up to 42 times"/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\TimeBetweenAttempts\Minutes5"
description="Wait 5 minutes between attempts"/>
                <step_condition
type="MyCompany\MyModule\Virtual\Condition\AllDownloadsAreReady" description="Start only if
downloads are ready"/>
                <step_condition
type="MyCompany\MyModule\Virtual\Condition\NoImportIsRunning" description="Ensure no other
import process is active"/>
            </conditions>
        </step>
        ...
    </pipeline>
</config>
```

How to create an executor

Pipeline Step Executor

Each pipeline step has one executor, which implements the interface \TechDivision\ProcessPipelines\Api\ExecutorInterface.

Abstract Executors

There are some abstract executors, which should be extended by your executor.

TechDivision\ProcessPipelines\Model\Executor\AbstractExecutor
: Base executor, which already implements methods for logging and warnings

TechDivision\ProcessPipelines\Model\Executor\AbstractShellExecutor
: Shell executor for run CLI and bash scripts.

Concrete Executors

There are some ready-to-use executors.

TechDivision\ProcessPipelines\Model\Executor\DropCache
: Cache invalidation executor

TechDivision\ProcessPipelines\Model\Executor\Reindex
: Reindex executor

Usage

To use these executors, you need to define them as following in your pipeline.xml.

```
<?xml version="1.0"?>
```

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision/ProcessPipelines/etc/pipeline
.xsd">
    <pipeline name="example_pipeline" description="Example pipeline for reindex and drop
cache">
        <conditions>
            <pipeline_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn"
description="Disable auto spawning"/>
        </conditions>
        <step name="example_reindex"
executorType="TechDivision\ProcessPipelines\Model\Executor\Reindex" sortOrder="10"
description="" >
            <arguments>
                <argument key="indexes"
value="catalog_category_product,catalog_product_category,catalogsearch_fulltext" />
            </arguments>
        </step>
        <step name="example_cache_drop"
executorType="TechDivision\ProcessPipelines\Model\Executor\DropCache" sortOrder="20"
description="" >
            <arguments>
                <argument key="caches" value="collections,full_page,block_html" />
            </arguments>
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Run after the first step"/>
            </conditions>
        </step>
    </pipeline>
</config>
```

Testing executors on a local environment (DEV)

While developing a new executor it is necessary to execute them without waiting for heartbeat and without having some runner up and running (see [Cron and Consumer/Runner](./system-cron-consumer.md)). Therefore we introduce the CLI command pipeline:dev:run:executor.

Usage

Run a customer executor in the same way as the runner would do.

```
bin/magento pipeline:dev:run:executor --arguments='{\"arg1\": \"value-1\", \"arg2\": 2}'
--pipeline-id=102 --step-id=1562 '\MyCompany\MyModule\Model\Executor\DoSomeThing'
```

The options --arguments, --pipeline-id and --step-id are optional. Sometimes there are dependencies inside your executor for these values.

How to deactivate a pipeline

In some cases, e. g. if a custom process is necessary, it'll be useful to deactivate the default pipelines that'll be provided by the modules

- TechDivision_PacemakerImportCatalog
- TechDivision_PacemakerImportInventory
- TechDivision_PacemakerImportPrice

which are part of the Pacemaker distribution.

Usage

To deactivate the default Pipelines, simple disable the appropriate modules, assuming that you're in the magento root directory, from the commandline with

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <process_pipelines>
            <mini_heartbeat>
                <enabled>1</enabled>
            </mini_heartbeat>
        </process_pipelines>
        <techdivision_pacemaker_import>
            <general>
                <source_directory>../import</source_directory>
                <media_source_directory>../import/media</media_source_directory>
            </general>
            <catalog>
                <!-- Disable default pacemaker pipeline -->
                <enabled>0</enabled>
                <!-- Enable project specific import pipeline -->
                <my_project_pipeline_enabled>1</my_project_pipeline_enabled>
                <ready_file_name>import.ok</ready_file_name>
            </catalog>
            <price>
                <enabled>0</enabled>
            </price>
            <inventory>
                <enabled>0</enabled>
            </inventory>
        </techdivision_pacemaker_import>
    </default>
</config>
```

and invoke bin/magento setup:upgrade again.

Import Processes

Transform foreign import source

Often the import files are not ready to be imported, when we receive them from the foreign systems. Usually there data missing, which needs to filled with Magento insides like category paths or website codes.

Therefor there is a transformation step inside the given import pipelines. In this example we will replace the default executor with custom logic.

Create an own module

First of all we need to introduce a custom module. Please refer to Create a New Module in Magento's developer documentation.

In this example we name the module MyModule_CustomCatalogImport.

```
mkdir -p app/code/MyModule/CustomCatalogImport/etc
touch app/code/MyModule/CustomCatalogImport/etc/module.xml
touch app/code/MyModule/CustomCatalogImport/registration.php
```

Listing 1. Content of registration.php

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_CustomCatalogImport',
    __DIR__
);
```

Listing 2. Content of module.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="MyModule_CustomCatalogImport" setup_version="1.0.0">
        <sequence>
            <module name="TechDivision_PacemakerImportCatalog"/>
        </sequence>
    </module>
</config>
```

We need to specify TechDivision_PacemakerImportCatalog module in the sequence of our new module.xml to be able to overwrite the existing pipeline configuration.

Manipulate the pipeline

By creating a own pipeline.xml in the etc folder of our module, we can create own pipelines and overwrite existing. In this example we want to change the executor of the step product_transformation step within the pacemaker_import_catalog pipeline.

Listing 3. Content of app/code/MyModule/CustomCatalogImport/etc/pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="pacemaker_import_catalog">
        <step name="product_transformation"
executorType="MyModule\CustomCatalogImport\Model\Executor\TransformationExecutor" />
    </pipeline>
</config>
```

We create a pipeline node with the name "pacemaker_import_catalog" and add here a single step node with the name "product_transformation". This both nodes are merged between the pipelines. By defining an own executorType we will change the previous defined executor, since the sequence in the module.xml of our module gives our module higher priority.

Transformation logic

Once we created our module and changed the executor for the transformation step, we need to create the executor class. In the previous part we already chose the class name MyModule\CustomCatalogImport\Model\Executor\TransformationExecutor, which means we need to create the file app/code/MyModule/CustomCatalog/Import/Model/Executor/TransformationExecutor.php according to the PSR-4 autoloader.

TIP We suggest the directory structure Model/Executor, Model/Condition/Step and Model/Condition/Pipeline for the corresponding classes.

Place following code we already have a valid executor.

```
<?php
declare(strict_types=1);

namespace MyModule\CustomCatalogImport\Model\Executor;

use TechDivision\ProcessPipelines\Api\StepInterface;
use TechDivision\ProcessPipelines\Model\Executor\AbstractExecutor;

class TransformationExecutor extends AbstractExecutor
{
    /**
     * @inheritdoc
     */
    public function process(StepInterface $step): void
    {

    }
}
```

NOTE In order to test your current implementation you can add following code into the body of your process method and execute the pipelines.

```
$this->getLogger()->info("It works ;)");
```

Use the CLI to observe the progress of your pipeline (bin/magento pipeline:status -w 2). See the details for your pipeline with bin/magento pipeline:detail <PIPELINE_ID> and retrieve the log for your transformation step with bin/magento pipeline:step:show <STEP_ID>. The result should be something like

```
--- ATTEMPT 1 ---
[2020-02-11 11:10:40] executor.INFO: It works ;)
```

Now you're able to add your custom logic to this executor to transform the given files. The file paths for this import pipeline are present in the $step as arguments. Execute $files = (array)$step->getArgumentValueByKey('files'); to fetch a list of file paths.

Examples

Checkout following resource for an example module. This example shows how to implement a mapping layer, which transforms data from CSV to CSV before running the import.

Add Steps to an existing Pipeline

Sometimes it is required to add additional steps to an existing import pipeline. Use cases could be

- Add re-indexations and cache invalidation steps
- Add cache warming processes
- Add image cropping executions
- and others

Create an own module

First of all we need to introduce a custom module. Please refer to Create a New Module in Magento's developer documentation.

In this example we name the module MyModule_CustomCatalogImport.

```
mkdir -p app/code/MyModule/CustomCatalogImport/etc
touch app/code/MyModule/CustomCatalogImport/etc/module.xml
touch app/code/MyModule/CustomCatalogImport/registration.php
```

Listing 4. Content of registration.php

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_CustomCatalogImport',
    __DIR__
);
```

Listing 5. Content of module.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="MyModule_CustomCatalogImport" setup_version="1.0.0">
        <sequence>
            <module name="TechDivision_PacemakerImportCatalog"/>
        </sequence>
    </module>
</config>
```

We need to specify TechDivision_PacemakerImportCatalog module in the sequence of our new module.xml to be able to overwrite the existing pipeline configuration.

Manipulate the pipeline

By creating a own pipeline.xml in the etc folder of our module, we are able to create own pipelines and extend existing. In this example we want to add additional steps to the existing pacemaker_import_catalog pipeline.

Listing 6. Content of app/code/TechDivision/CustomCatalogImport/etc/pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="pacemaker_import_catalog">
        <step name="reindex_attributes"
executorType="TechDivision\ProcessPipelines\Model\Executor\Reindex" sortOrder="100"
description="Run full reindex">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
                <step_condition
type="TechDivision\AddNewStepsToExistingPipeline\Virtual\Condition\Step\IsReindexingStage"
description="Ensure it is time for reindexing."/>
            </conditions>
        </step>
        <step name="drop_cache"
executorType="TechDivision\ProcessPipelines\Model\Executor\DropCache" sortOrder="110"
description="Drop relevant caches">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Previous step needs to be finished."/>
            </conditions>
        </step>
```

```
    </pipeline>
</config>
```

We create a pipeline node with the name "pacemaker_import_catalog" and add here two step nodes. With the sortOrder attribute we define the position within our pipeline. In this example the both steps will run as last.

Examples

Checkout following resource for an example module.

Create an additional import process

Sometimes we need additional imports beside the given. In this example we create an own import pipeline, which observes the import directory for files and executes the import without writing any PHP code, but reusing existing executors.

TIP You'll find all this code also in this example module.

Create an own module

First of all we need to introduce a custom module. Please refer to Create a New Module in Magento's developer documentation.

In this example we name the module MyModule_ProductStatusUpdate. The main purpose of our module will be to observer the import directory for a file with a specific name pattern. If this file is given we will introduce an import with the M2IF library.

```
mkdir -p app/code/MyModule/ProductStatusUpdate/etc
touch app/code/MyModule/ProductStatusUpdate/etc/module.xml
touch app/code/MyModule/ProductStatusUpdate/registration.php
```

Listing 7. Content of app/code/MyModule/ProductStatusUpdate/registration.php

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_ProductStatusUpdate',
    __DIR__
);
```

Listing 8. Content of app/code/MyModule/ProductStatusUpdate/etc/module.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="MyModule_ProductStatusUpdate" setup_version="1.0.0">
        <sequence>
            <module name="TechDivision_PacemakerImportBase"/>
        </sequence>
    </module>
</config>
```

Since we will use components from TechDivision_PacemakerImportBase module we need to add this module to the loading sequence of our new module.

Define the import pipeline

We create a pipeline.xml file within the etc folder of our module and add following content:

Listing 9. app/code/MyModule/ProductStatusUpdate/etc/pipeline.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:TechDivision_ProcessPipelines:etc/pipeline
.xsd">
    <pipeline name="product_status_update_import" description="Example Pipeline for a custom
product update import" use-working-directory="true">
        <conditions>
            <pipeline_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn" description="No
automatic start for this pipeline"/>
        </conditions>
        <step name="move_files"
executorType="TechDivision\PacemakerImportBase\Model\Executor\MoveFilesToWorkingDirectory"
sortOrder="10" description="Move files to working directory.">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
            </conditions>
        </step>
        <step name="product_status_update"
executorType="TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor" sortOrder="20"
description="Import product data">
            <conditions>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
description="Try once."/>
                <step_condition
type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
description="Previous step needs to be finished."/>
                <step_condition
type="MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess"
description="Avoid conflicts between import steps."/>
            </conditions>
            <arguments>
                <argument key="command" value="import:products" />
                <argument key="operation" value="add-update" />
                <argument key="configuration" value="MyModule_ProductStatusUpdate::etc/m2if-
config.json" />
            </arguments>
        </step>
```

```
    </pipeline>
</config>
```

In this sample we creat a pipeline with two steps. The first step moves the relevant files to the working directory. The second steps executes the import. But let's go thru the code step-by-step. Also checkout the XML config documentation for deeper insides.

Pipeline definition

The first node of our XML file defines a new pipeline.

```
<pipeline
    name="product_status_update_import"
    description="Example Pipeline for a custom product update import"
    use-working-directory="true" />
```

With the attribute name we give our pipeline a unique name. If there is already a pipeline with the same name exists within our magento instance they will be merged. Please refer to Transform foreign import source and Add steps to an existing pipeline to learn how to use this behaviour to extend existing pipelines.

The attribute description contains a human readable description of the purpose of our pipeline. This content will be used in the admin UI.

Since our new pipeline works with files, we need a working directory for every instance of our pipeline. Therefore we add the attribute use-working-directory and set it to true.

Pipeline conditions

With the pipeline_conditions we define when our pipeline should be initialized.

NOTE Plese refer to How to create a pipeline condition for details.

```
<pipeline>
    <conditions>
        <pipeline_condition
            type="TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn"
            description="No automatic start for this pipeline"/>
    </conditions>
</pipeline>
```

For our product updates we do not want to have periodical instantiation, since we want run the import once the required import file is present in the file system. Therefore we will use the pipeline initializer feature. To avoid an instantiation via the heartbeat we need to add the condition TechDivision\ProcessPipelines\Helper\Condition\Pipeline\NoAutoSpawn.

Move files step

Our first step will simply move the relevant import files to the working directory of our pipeline.

```
<step
    name="move_files"
```

```
executorType="TechDivision\PacemakerImportBase\Model\Executor\MoveFilesToWorkingDirectory"
    sortOrder="10"
    description="Move files to working directory.">

    <conditions>
        <step_condition
            type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
            description="Try once."/>
    </conditions>
</step>
```

We name our step (e.g. move_files). Step names are relevant if we need to specify parallel and sequential execution. As executor we use the already existing class TechDivision\PacemakerImportBase\Model\Executor\MoveFilesToWorkingDirectory. Please refer to How to create an executor.

The sortOrder of our step is important, since we want to execute the steps in a sequence (not parallel).

Since this step is the first in our pipeline it does not require any specific sequence relevant conditions. By adding the condition TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1 we ensure, that our pipeline will be canceled, if errors appear while moving the files to the working directory.

Import data step

The second step will execute the import. We need to ensure this step is running after the first. And we need to check whether other imports are probably already running and wait until they are finished to avoid deadlocks in database.

```
<step
    name="product_status_update"
    executorType="TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor"
    sortOrder="20"
    description="Import product data">

    <conditions>
        <step_condition
            type="TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1"
            description="Try once."/>
        <step_condition

type="TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted"
            description="Previous step needs to be finished."/>
        <step_condition

type="MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess"
            description="Avoid conflicts between import steps."/>
    </conditions>

    <arguments>
        <argument key="command" value="import:products" />
        <argument key="operation" value="add-update" />
        <argument key="configuration" value="MyModule_ProductStatusUpdate::etc/m2if-
```

```
config.json" />
    </arguments>
</step>
```

We name the step and define an executor. To run M2IF based import we can re-use the class TechDivision\PacemakerImportBase\Model\Executor\ImportExecutor.

As sortOrder we need to define a higher number than for the move_files step, since we want use the condition TechDivision\ProcessPipelines\Helper\Condition\Step\PreviousStepsCompleted. This condition verifies, whether steps with lower sortOrder are successfully executed.

To avoid endless retries of this step in case of errors we need to add the TechDivision\ProcessPipelines\Helper\Condition\Step\AttemptsLimit\Limit1 as we did it for the first step.

As third condition for this step we add MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess. This condition does not exists yet, but we will create it in the next part of this documentation.

The import executor expects a set of arguments, which defines the command and operation to execute as well as the path to the configuration we want to pass to the import library.

TIP Take a look into the M2IF documentation for a better understanding for command, operation and configuration.

Avoid execution of conflicting steps

In the previous part of this documentation we've already defined the step condition MyModule\ProductStatusUpdate\Virtual\Condition\Step\NoConflictingStepInProcess. Now we need to create this class. The namespace of this class contains Virtual. This means this class will be auto-generated by Magento. Therefore we create a di.xml in our module and add following content into it.

Listing 10. app/code/MyModule/ProductStatusUpdate/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <virtualType
name="MyModule\ProductStatusUpdateImport\Virtual\Condition\NoConflictingStepInProcess"
type="TechDivision\PacemakerImportBase\Model\Condition\Step\NoConflictingStepsInProcess">
        <arguments>
            <argument name="stepNames" xsi:type="array">
                <item name="product_status_update"
xsi:type="string">product_status_update</item>
                <item name="product_import" xsi:type="string">product_import</item>
            </argument>
        </arguments>
    </virtualType>
</config>
```

TIP Refer to Magento DevDocs for details about virtual types.

This approach allows us to extend the list of conflicting processes. Our condition checks now for active import steps from our pipeline as well as for the import step from the pacemaker_import_catalog pipeline.

We also should extend the list of conflicting steps for the pacemaker_import_catalog pipeline, by adding following code to the di.xml of our module.

Listing 11. app/code/MyModule/ProductStatusUpdate/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType
name="TechDivision\PacemakerImportCatalog\Virtual\Condition\NoConflictingStepInProcess">
        <arguments>
            <argument name="stepNames" xsi:type="array">
                <item name="product_status_update"
xsi:type="string">product_status_update</item>
            </argument>
        </arguments>
    </virtualType>
</config>
```

M2IF configuration file

Now we need to add the M2IF configuration file to the path we pass to our import executor (MyModule_ProductStatusUpdate::etc/m2if-config.json). The content of this configuration file looks as following:

Listing 12. app/code/MyModule/ProductStatusUpdate/etc/m2if-config.json

```
{
    "magento-edition": "CE",
    "magento-version": "2.3.4",
    "operation-name" : "add-update",
    "archive-artefacts" : false,
    "debug-mode" : false,
    "entity-type-code" : "catalog_product",
    "listeners" : [
        {
            "app.set.up" : [
                "import.listener.render.ansi.art",
                "import.listener.initialize.registry"
            ]
        },
        {
            "app.tear.down" : [
                "import.listener.clear.registry"
            ]
        }
    ],
    "databases" : [],
    "loggers": [
        {
            "name": "system",
```

```
            "channel-name": "logger/system",
            "type": "Monolog\\Logger",
            "handlers": [
                {
                    "type": "Monolog\\Handler\\ErrorLogHandler",
                    "formatter": {
                        "type": "Monolog\\Formatter\\LineFormatter",
                        "params" : [
                            {
                                "format": "[%datetime%] %channel%.%level_name%: %message%
%context% %extra%",
                                "date-format": "Y-m-d H:i:s",
                                "allow-inline-line-breaks": true,
                                "ignore-empty-context-and-extra": true
                            }
                        ]
                    }
                }
            ],
            "processors": [
                {
                    "type": "Monolog\\Processor\\MemoryPeakUsageProcessor"
                }
            ]
        }
    ],
    "operations" : [
        {
            "name" : "add-update",
            "plugins" : [
                {
                    "id": "import.plugin.cache.warmer"
                },
                {
                    "id": "import.plugin.global.data"
                },
                {
                    "id": "import.plugin.subject",
                    "subjects": [
                        {
                            "id": "import.subject.move.files",
                            "identifier": "move-files",
                            "file-resolver": {
                                "prefix": "product-status-update"
                            },
                            "ok-file-needed": true
                        },
                        {
                            "id": "import_product.subject.bunch",
```

```
                        "identifier": "files",
                        "file-resolver": {
                            "prefix": "product-status-update"
                        },
                        "params" : [
                            {
                                "copy-images" : false,
                                "clean-up-empty-columns" : [
                                    "base_image",
                                    "small_image",
                                    "swatch_image",
                                    "thumbnail_image",
                                    "special_price",
                                    "special_price_from_date",
                                    "special_price_to_date"
                                ]
                            }
                        ],
                        "observers": [
                            {
                                "import": [
                                    "import.observer.attribute.set",
                                    "import.observer.additional.attribute",
                                    "import_product.observer.product",
                                    "import_product.observer.product.attribute.update"
                                ]
                            }
                        ]
                    }
                ]
            },
            {
                "id": "import.plugin.archive"
            }
        ]
    }
]
}
```

With this configuration we define an import, which simply updates attributes for already existing products. Please refer to M2IF documentation for details.

Pipeline instantiation

In the last part of this documentation we teaches the pipeline-initializer to observe the import directory for our import files and instantiates our pipeline once the file is present. All this can be done completely in declarative way.

System configuration

Let's add some configuration options to Magento's admin UI. We create a system.xml in the etc/adminhtml directory of our module an add following content into this file.

Listing 13. app/code/MyModule/ProductStatusUpdate/etc/adminhtml/system.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
    <system>
        <section id="techdivision_pacemaker_import">
            <group id="product_status_update" showInDefault="1" showInStore="0"
showInWebsite="0" sortOrder="1000" translate="label">
                <label>Product Status Update</label>
                <field id="enabled" translate="label" type="select" sortOrder="25"
showInDefault="1" showInWebsite="0" showInStore="0" canRestore="0">
                    <label>Enable Pipeline</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
                <field id="file_name_pattern" translate="label comment" type="text"
sortOrder="30" showInDefault="1" showInWebsite="0" showInStore="0" canRestore="0">
                    <label>File Name Pattern</label>
                    <comment>Pattern for import file bunches</comment>
                </field>
            </group>
        </section>
    </system>
</config>
```

This will add a new configuration group to **Stores › Configuration › Pacemaker › Import** with two fields. One to enable/disable our pipeline and a second one, which defines the file name pattern for our import files.

Create now a default configuration by creating following file:

Listing 14. app/code/MyModule/ProductStatusUpdate/etc/config.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <techdivision_pacemaker_import>
            <product_status_update>
                <enabled>1</enabled>
                <file_name_pattern><![CDATA[/product-status-update_(?P<identifier>[0-9a-z\-
]*)([_0-9]*?).(csv|ok)/i]]></file_name_pattern>
            </product_status_update>
        </techdivision_pacemaker_import>
    </default>
</config>
```

This will enable our module by default and pre-defines a file name pattern.

Extend Pipeline Initializer

Add following content to the di.xml of our module

Listing 15. app/code/MyModule/ProductStatusUpdate/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    ...
    <virtualType
name="MyModule\ProductStatusUpdateImport\Virtual\ConfigProvider\GetFileNamePattern"
type="TechDivision\PacemakerImportBase\Model\ConfigProvider\GetFileNamePattern">
        <arguments>
            <argument name="scopeConfigPath"
xsi:type="string">techdivision_pacemaker_import/product_status_update/file_name_pattern</arg
ument>
        </arguments>
    </virtualType>
    <virtualType name="MyModule\ProductStatusUpdateImport\Virtual\ImportBunchResolver"
type="TechDivision\PacemakerImportBase\Model\ImportBunchResolver">
        <arguments>
            <argument name="getFileNamePattern"
xsi:type="object">MyModule\ProductStatusUpdateImport\Virtual\ConfigProvider\GetFileNamePatte
rn</argument>
        </arguments>
    </virtualType>
    <type name="TechDivision\PacemakerImportBase\Model\ImportFilesDataFetcher">
        <arguments>
            <argument name="resolverConfig" xsi:type="array">
                <item name="my_module.product_status_update_import" xsi:type="array">
                    <item name="resolver"
xsi:type="object">MyModule\ProductStatusUpdateImport\Virtual\ImportBunchResolver</item>
                    <item name="validator"
xsi:type="object">TechDivision\PacemakerImportBase\Api\ImportBunchValidatorInterface</item>
                    <item name="pipeline_name"
xsi:type="string">product_status_update_import</item>
                    <item name="enable_config_path"
xsi:type="string">techdivision_pacemaker_import/product_status_update/enabled</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

With this configuration we register an additional virtual type (auto-generated class) MyModule\ProductStatusUpdateImport\Virtual\ConfigProvider\GetFileNamePattern. This class retrieves the file pattern to our "ImportBunchResolver". If you do not want a configurable file pattern, you could also implement an own implementation for TechDivision\PacemakerImportBase\Api\GetFileNamePatternInterface and use this instead of the virtual type.

The "ImportBunchResolver" is also a virtual type and requires our previously defined FileNamePattern provider.

The third node of the XML code above extends the resolverConfig array of the class TechDivision\PacemakerImportBase\Model\ImportFilesDataFetcher. The new item we add to this array requires at least three items:

- resolver: This is an instance of TechDivision\PacemakerImportBase\Api\ImportBunchResolverInterface. We add here our virtual type, which we defined before.
- validator: This is an instance of TechDivision\PacemakerImportBase\Api\ImportBunchValidatorInterface. Since we do not add any custom validation logic we can add the interface directly and Magento's DI will instantiate the default preference for it.
- pipeline_name: The name of our pipeline, which should be instantiated once the resolver found import files and the validator approved them.

The fourth item in this array is optional. With enable_config_path we can add the configuration path to our feature toggle. This allows us to disable the instantiation of this pipeline.

Examples

You'll find an example module, which also provides sample CSV files to test this import.

Order Workflow

Add custom export format

Create an own module

First of all we need to introduce a custom module. Please refer to Create a New Module in Magento's developer documentation.

In this example we name the module MyModule_CustomOrderExportFormat.

```
mkdir -p app/code/MyModule/CustomOrderExportFormat/etc
touch app/code/MyModule/CustomOrderExportFormat/etc/module.xml
touch app/code/MyModule/CustomOrderExportFormat/registration.php
```

Listing 16. app/code/MyModule/CustomOrderExportFormat/registration.php

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyModule_CustomOrderExportFormat',
    __DIR__
);
```

Listing 17. app/code/MyModule/CustomOrderExportFormat/etc/module.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
```

```
    <module name="MyModule_CustomOrderExportFormat" setup_version="1.0.0">
        <sequence>
            <module name="TechDivision_PacemakerOrderExport"/>
        </sequence>
    </module>
</config>
```

We need to specify TechDivision_PacemakerOrderExport module in the sequence of our new module.xml to be able to overwrite the existing pipeline configuration.

Create your format executor

Now we need to create an own executor, which will load the order and generate the target format for our export. Therefore we crate a class as following

Listing 18. app/code/MyModule/CustomOrderExportFormat/Model/MyExecutor.php

```
<?php
declare(strict_types=1);

namespace MyModule\CustomOrderExportFormat\Model;

use Magento\Framework\Exception\LocalizedException;
use Magento\Sales\Api\OrderRepositoryInterface;
use TechDivision\ProcessPipelines\Api\ExecutorLoggerInterface;
use TechDivision\ProcessPipelines\Api\StepInterface;
use TechDivision\ProcessPipelines\Model\Executor\AbstractExecutor;

class MyExecutor extends AbstractExecutor
{
    /**
     * @var OrderRepositoryInterface
     */
    private $orderRepository;

    /**
     * @param ExecutorLoggerInterface $logger
     * @param OrderRepositoryInterface $orderRepository
     */
    public function __construct(
        ExecutorLoggerInterface $logger,
        OrderRepositoryInterface $orderRepository
    ) {
        parent::__construct($logger);
        $this->orderRepository = $orderRepository;
    }

    /**
     * @param StepInterface $step
     * @return void
```

```
     * @throws LocalizedException
     */
    public function process(StepInterface $step)
    {
        // Load the order with the order_id given in step arguments
        $order = $this->orderRepository->get((int)$step->getArgumentValueByKey('order_id'));

        // Formatting logic here...
    }
}
```

TIP If you want to use the default transport adapter to move the result to a location within the system, you need to persist the body within the current working directory in the file order-export.body. Please refer to example module for more details.

Register the executor and dropdown select

Finally you need your executor to the format list. This will automatically add your formatter to he dropdown in **Store › Configuration › Pacemaker › Order Workflow**.

Listing 19. app/code/MyModule/CustomOrderExportFormat/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="TechDivision\PacemakerOrderExport\Model\ExportFormatProvider">
        <arguments>
            <argument name="formatList" xsi:type="array">
                <item name="my_custom_format" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_format</item>
                    <item name="label" xsi:type="string">My Custom Format</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportFormat\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module.

Add custom transport adapter

You can add an executor, which transports your transformed order to a target destination, in the same way like described for custom export formatter.

To register your executor you need to add following entry to the di.xml of your module:

Listing 20. app/code/MyModule/CustomOrderExportTransportAdapter/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="TechDivision\PacemakerOrderExport\Model\TransportAdapterProvider">
        <arguments>
            <argument name="transportAdapterList" xsi:type="array">
                <item name="my_custom_transport_adapter" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_transport_adapter</item>
                    <item name="label" xsi:type="string">My Custom Transport Adapter</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportTransportAdapter\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module.

Add custom order response handler

You can add an executor, which handles the order export response, in the same way like described for custom export formatter.

To register your executor you need to add following entry to the di.xml of your module:

Listing 21. app/code/MyModule/CustomOrderExportResponseHandler/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
     <type name="TechDivision\PacemakerOrderExport\Model\ResponseHandlerProvider">
        <arguments>
            <argument name="handlerList" xsi:type="array">
                <item name="my_custom_response_handler" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_response_handler</item>
                    <item name="label" xsi:type="string">My Custom Response Handler</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportResponseHandler\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module.

Add custom notification handler

You can add an executor, which sends a (or multiple) notification, in the same way like described for custom export formatter.

To register your executor you need to add following entry to the di.xml of your module:

Listing 22. app/code/MyModule/CustomOrderExportNotification/etc/di.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="TechDivision\PacemakerOrderExport\Model\NotificationHandlerProvider">
        <arguments>
            <argument name="notificationHandlerList" xsi:type="array">
                <item name="my_custom_notification" xsi:type="array">
                    <item name="code" xsi:type="string">my_custom_notification</item>
                    <item name="label" xsi:type="string">My Custom Notification</item>
                    <item name="type"
xsi:type="string">MyModule\CustomOrderExportNotification\Model\MyExecutor</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Examples

Checkout following resource for an example module.

FAQ

Composer runs into auth issues on my Mac OS X machine.

Question

As Solution Partner or Customer, you received a ext12345 username together with a token. This gives you access via composer to the necessary libraries. These credentials have to be added in your auth.json, which can, for example, be in the source directory of your project like src/auth.json.

Example:

```
{
   "http-basic": {
      "gitlab.met.tdintern.de": {
         "username": "ext00000",
         "password": "asaZIjkUIollKSADnrlm"
      }
   }
}
```

Whenever composer will be invoked, these credentials will be used for the HTTP download, composer will do. In some cases, you'll receive a message from composer that you'll not have the necessary access rights to install pacemaker or one of its packages.

Answer

macOS saves the credentials in the keychain on the host level (https://gitlab.met.tdintern.de). When invoking composer the first time, and it doesn't matter from which project, the first host entry from the keychain will be used and not the one from the auth.json anymore. This may lead to the problem that the Pacemaker libraries can not be loaded anymore because of missing access.

As generally, you'll access the GIT repositories over SSH, in case of pacemaker it will be necessary to disable the caching of the credentials of the system for HTTPS calls. Therefore execute the following commands to remove the credential helper from the GIT configuration

```
git config --local --unset-all credential.helper \
  && git config --global --unset-all credential.helper \
  && git config --system --unset-all credential.helper
```

After that, the credential helper has to re-initialized empty (because of any other tool like xCode for example may also use it) with the following command

```
git config --global --add credential.helper "" && composer clear-cache
```

Finally, search in the keychain tool for met and delete the entry gitlab.met.tdintern.de. If the keychain has been deactivated, in

the future GIT should **ALWAYS** use the credentials from the auth.json from your project or the global one.

The performance on the production/stating system is worse than on my local machine!

Question

The performance between your local and any other system differs significantly. What can be the reason?

Answer

Probably the MySQL transaction log has different settings. The option innodb_flush_log_at_trx_commit by default has the value 1. This means, that the transaction log will be written by MySQL after each commit. This option controls the balance between strict ACID compliance for commit operations and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. Setting this value to 2, you can achieve better performance, but then you can lose transactions in a crash.

Possible values are

0	write and flush once per second
1	write and flush at each commit
2	write at commit, flush once per second

For example, switching this value from 1 to 2 the import performance improves from 02:06:10 to 00:03:29 h which is for sure significant

ID	Pipeline	Created	Finished	Duration
219	xxx_import_catalog	Oct 24, 2019 2:47:04 PM	Oct 24, 2019 4:53:14 PM	02:06:10 h
221	xxx_import_catalog	Oct 24, 2019 5:02:02 PM	Oct 24, 2019 5:05:31 PM	00:03:29 h

Supervisor does not restarts the runners any more

Problem

After working for a while as a charm the supervisor stops restarting the runner consumers. The supervisor log displays an error like FATAL Exited too quickly (process log may have details).

Answer

Even if a process exits with an “expected” exit code, supervisor will consider the start as a failure if the process exits quicker than startsecs. Which is default 1 second. You need to set the option startsecs=0 to the supervisor configuration in order to avoid this behaviour.

Please refer to How to configure the runners.