TechD·vision

# Pacemaker

# Table of Contents

# PACEMAKER

Pacemaker is an extension bundle for Magento 2. It is a new way to organize your background processes inside your shop system. Therefore Pacemaker uses the PipelinePattern, which is familiar to most developers from tools like Gitlab, Jenkins or Travis. Developers are using this tools in order 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

# Get started

## How to install

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

Before you begin the install, you need a running Magento 2 instance. Please refer Magento's documentation: [Install Magento using Composer](https://devdocs.magento.com/guides/v2.3/install-gde/composer.html). You need also to install RabbitMQ and configure it for your Magento instance. Refer the [article from Magento's DevDocs](https://devdocs.magento.com/guides/v2.3/install-gde/prereq/install-rabbitmq.html).

## 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="1.1.*" in order to install the module. == Configure Heartbeat Cron :idprefix: :idseparator: - :experimental: :imagesdir: ../../images

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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 in order 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
```

## 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
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 1 in order to 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 up issues, which are easy to fix but hard to detect.

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

# Setup RabbitMQ on MacOS X

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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'
        ]
    ],
];
```

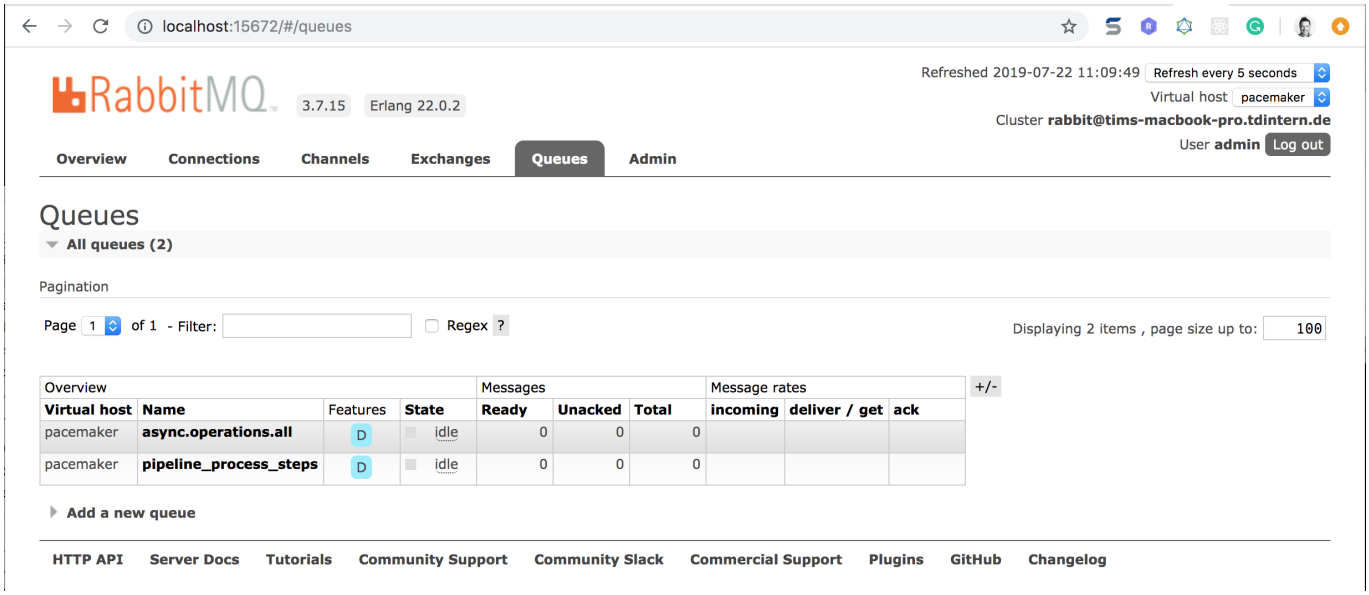
After that, finalize the Magento setup with the following command

```
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

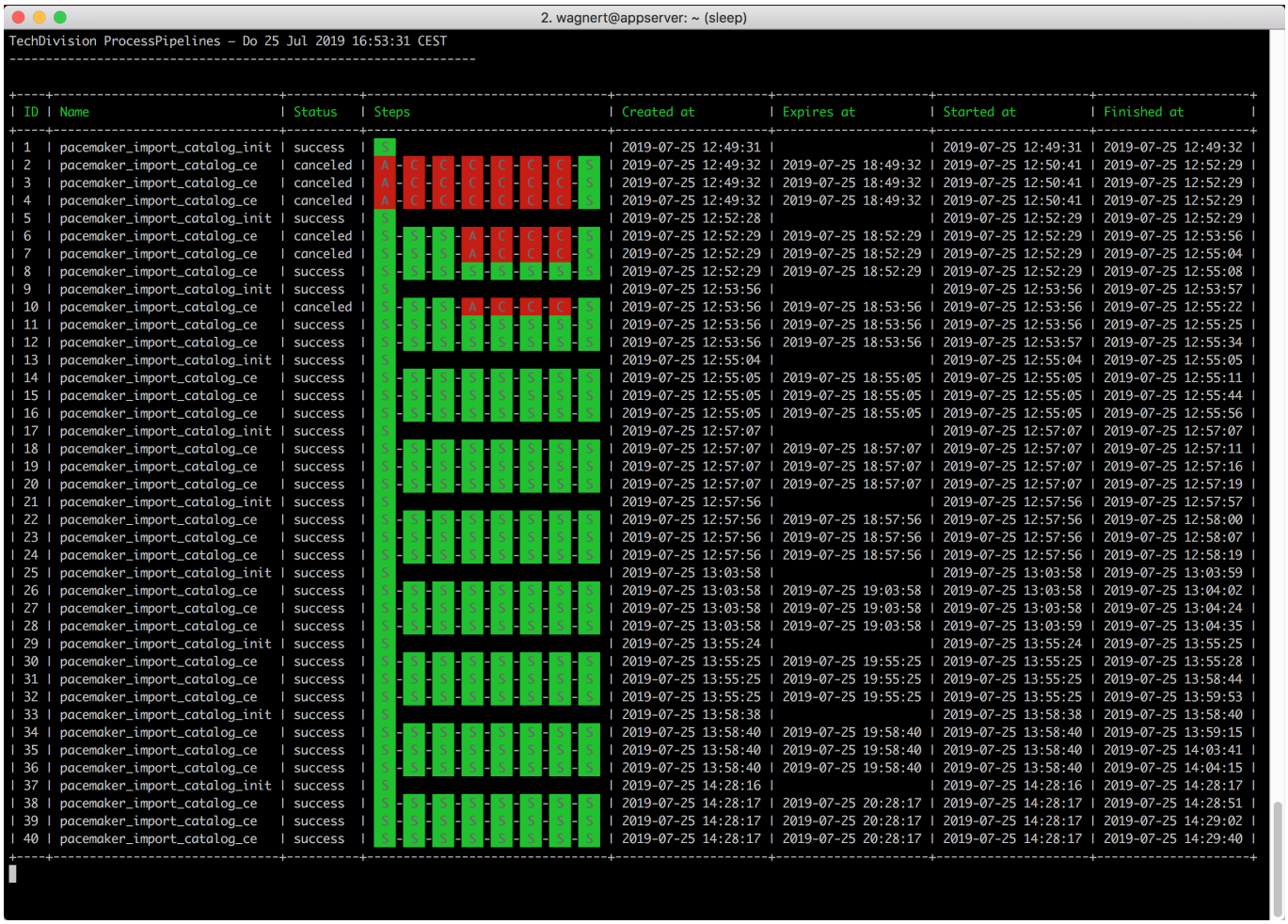


# CLI Status Update

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




# 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

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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. Or alternatively login into the backend (Magento's admin UI) and open the System > Pacemaker > Pipelines Grid.

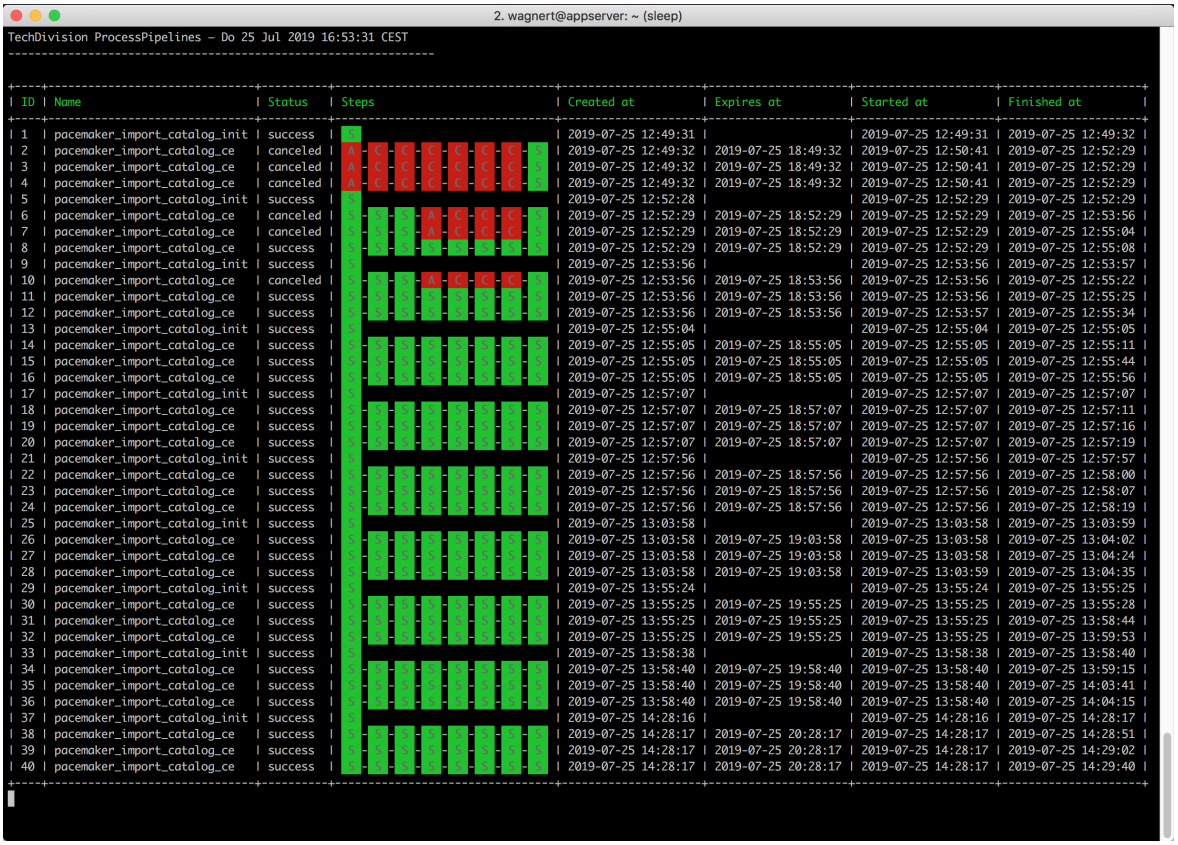



*Figure 1. Result output for* bin/magento pipeline:status

# Ready to use pipelines

## Catalog Import

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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 own data files or for 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.

## 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 in order to customize for mapping and transformation purpose |
| Pre-Import | index_suspender_start | Activates delta index suspending in order 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 > TechDivision > Pacemaker Import > Catalog Import

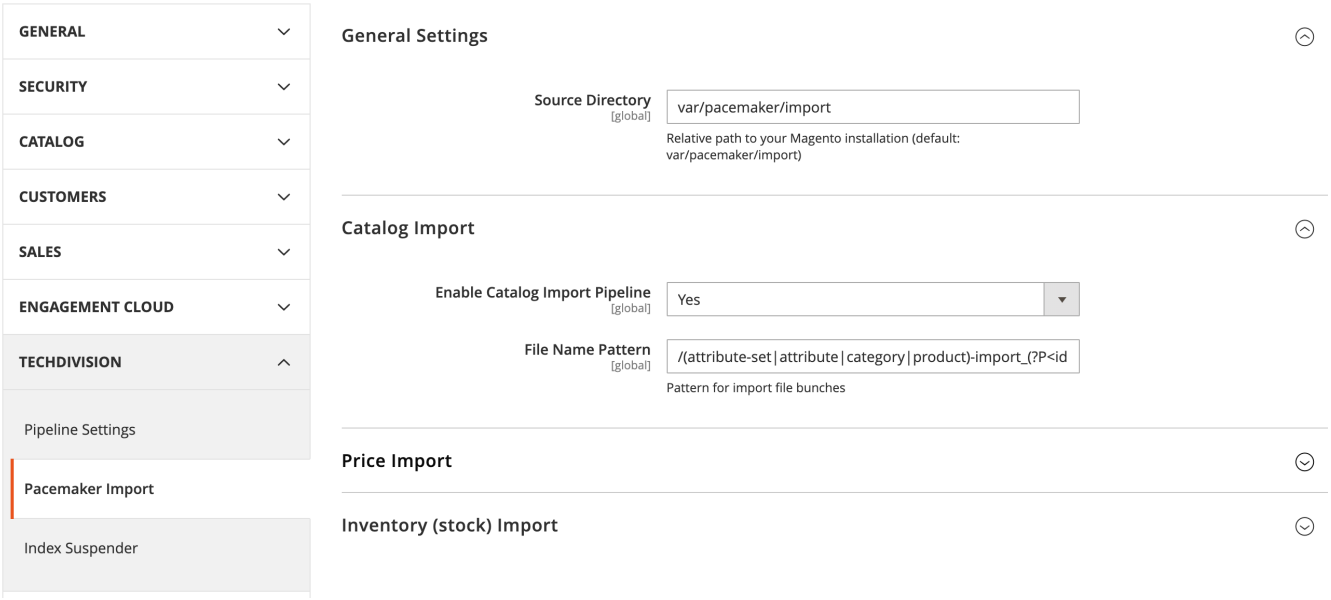



*Figure 2. Configuration in Magento Backend*

| 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 |

# Import Files Observer (How it works)

The observer for import files is also a pipeline. The condition TechDivision\PacemakerImportCatalog\Virtual\Condition\HasImportBunches uses the configuration Catalog Import > File Name Pattern in order to detect importable file bunches. If there are some in the source directory, the step init_import_pipeline will create a new import pipeline for each bunch of files.

## 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
```

# Price Import

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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 own data files or for 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.

## 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 | price_transformation | This step has a dummy executor in default and is designed in order to customize for mapping and transformation purpose |
| Pre-Import | index_suspender_start | Activates delta index suspending in order 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 > TechDivision > Pacemaker Import > Price Import

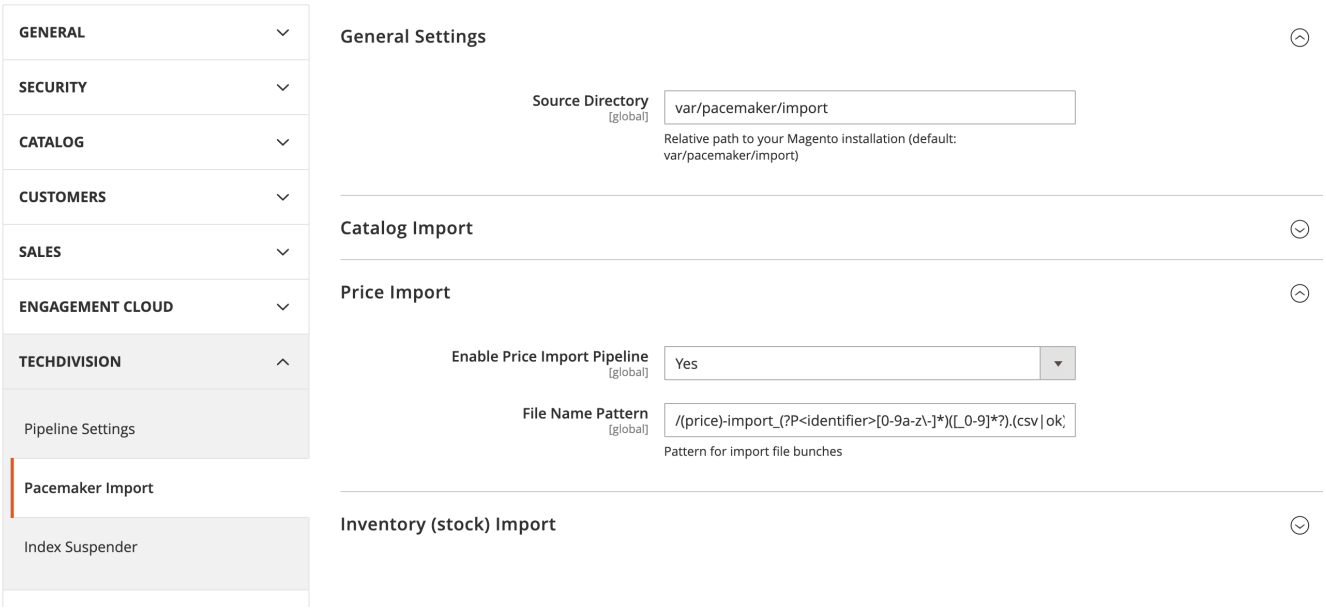



*Figure 3. Configuration in Magento Backend*

| 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 |

# Import Files Observer (How it works)

Source directory observer works in the same way like for Catalog Import. Please refer this page in order to understand how initialization of new import pipelines work. == Inventory Import :idprefix: :idseparator: - :experimental: :imagesdir: ../../images

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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 own data files or for 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.

# 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 | inventory_transformation | This step has a dummy executor in default and is designed in order to customize for mapping and transformation purpose |
| Pre-Import | index_suspender_start | Activates delta index suspending in order 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 > TechDivision > Pacemaker Import > Inventory (stock) Import

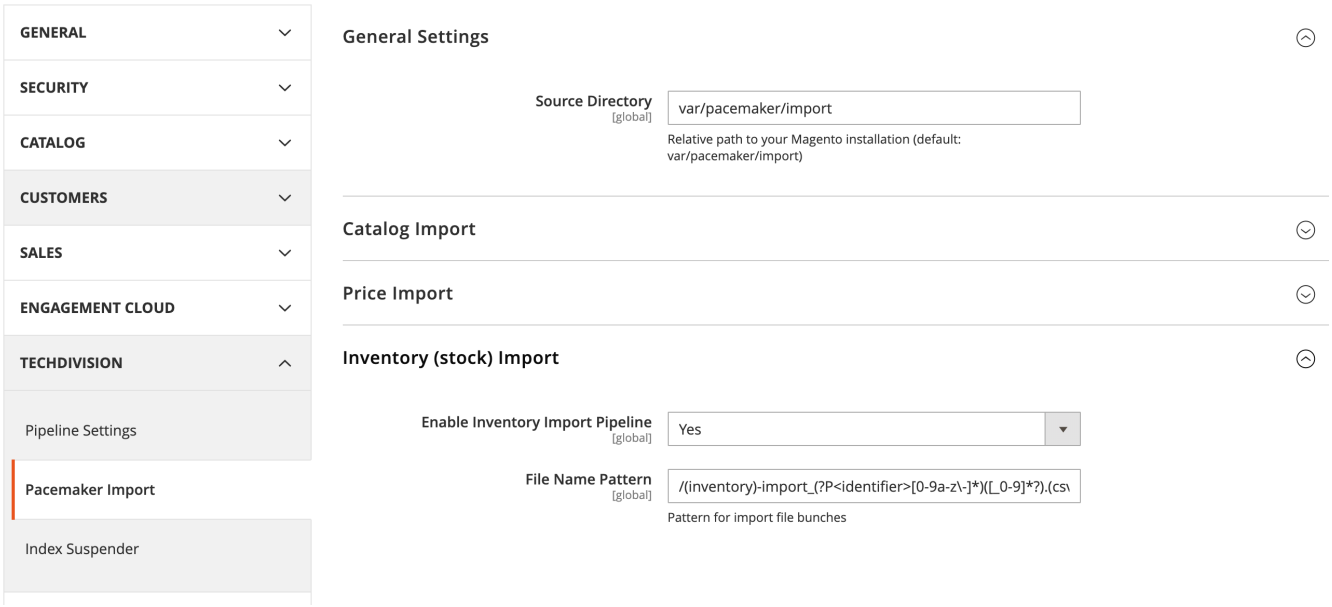



*Figure 4. Configuration Inventory Import*

| 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 |

# Import Files Observer (How it works)

Source directory observer works in the same way like for Catalog Import. Please refer this page in order to understand how initialization of new import pipelines work.

# Components & Concepts

## Process Pipelines

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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.

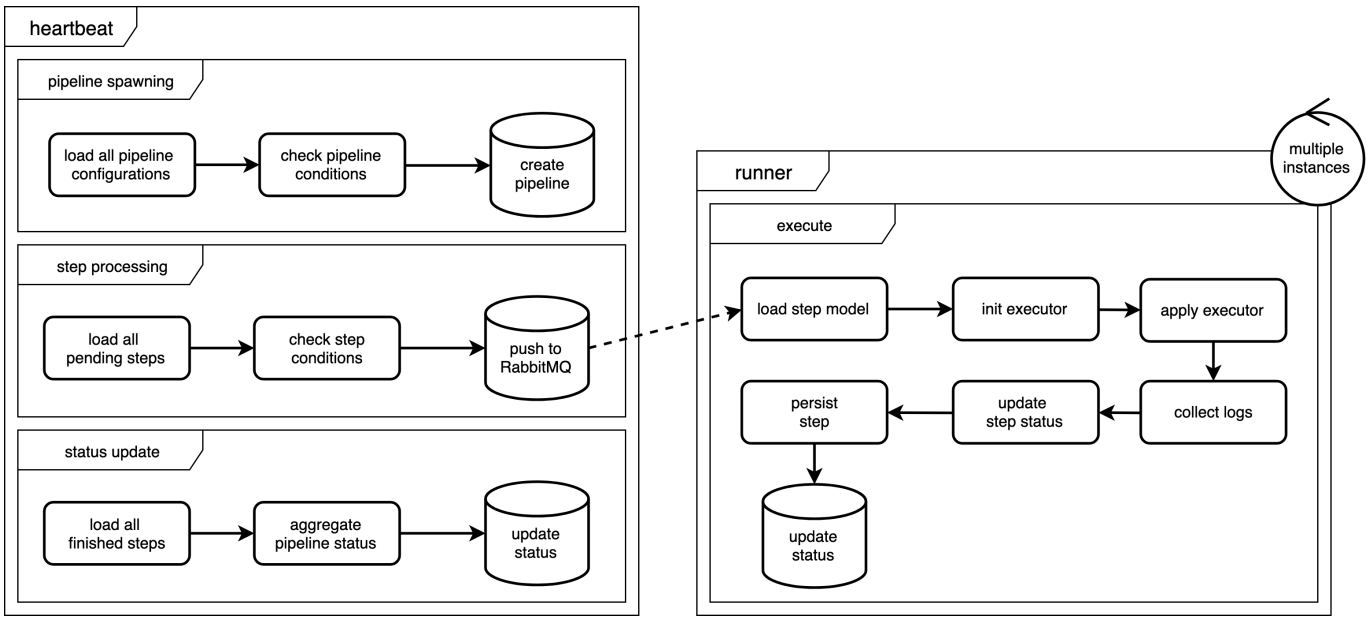



*Figure 5. Heartbeat and Runner*

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 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).

- Please refer to the How to Configure a ProcessPipeline page

Every 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 == How to configure a pipeline :idprefix: :idseparator: - :experimental: :imagesdir: ../../../images

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

In order 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 in order to cause a new pipeline execution |
| pipeline/conditions/pipeline_condition | Configuration of a pipeline condition |

| Node | Description |
|---|---|
| -- type | **Required**, string; Defines the class, which will be executed in order 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 in order to run the step. |
| -- 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 in order 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 in order 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

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

## 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

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

## 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.

### 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

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\PreviousStepsCompleted**
This ready-to-use condition can also be extended.

**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…

There are some 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

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

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

## 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 own 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

In order 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 :idprefix: :idseparator: - :experimental: :imagesdir: ../../../images

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

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

In order 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. == M2IF :idprefix: :idseparator: - :experimental: :imagesdir: ../../images

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

M2IF is an open-source import framework for Magento 2. Please refer to M2IF Website for more details.

# Professional Edition (PE)

Beside the Community Edition that can be downloaded from the our website (see above), an Professional Edition (PE) is available. The PE is part of the Pacemaker Distribution, but can also be installed standalone. The main advantage of the PE is a performance improvement comparing to the CE of round about 30 - 50 %.

# Installation

There are two installation options available. The first one is the possiblity to download it as PHAR, the second one to install via Composer.

To install it via Composer. For the Composer installation, the same How to install as for Pacemaker has to be done. But instead of installing Pacemaker invoke composer require techdivision/import-cli-por="3.7.*" in order to install the M2IF PE.

# Inside Pacemaker

Pacemaker is using M2IF as a component, which drives the imports. Therefore it brings its own configuration files for M2IF. These files are located in the etc directory of each pacemaker-import package (see vendor/techdivision/pacemaker-import-catalog/etc/import-config or vendor/techdivision/pacemaker-import-price/etc/import-config, etc.).

Each import step defines the used import configuration file as an argument inside the pipeline.xml.

```
<argument key="configuration" value="TechDivision_PacemakerImportCatalog::etc/import-
config/community/products.json" />
```

# FAQ

**NOTE** This documentation is not for the latest version Pacemaker version. Click here to switch to version 1.2

# 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 credendials 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 it's packages.

## Answer

Mac OS saves the credentials in the keychain on 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 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 fom 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 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/staging 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 |