

# Solr Bundle

For use with eZ Publish 5.4, go to [the corresponding documentation page](#) which covers the v1.0 version of the bundle compatible with eZ Publish 5.4.

## What is Solr Search Engine Bundle?

`ezplatform-solr-search-engine` as the package is called, aims to be a transparent drop in replacement for the SQL based "Legacy" search engine powering eZ Platform Search API by default. By enabling Solr and re-indexing your content, all your existing Search queries using `SearchService`, will be powered by Solr automatically. This allows you to scale up your eZ Platform installation and be able to continue development locally against SQL engine, and have a test infrastructure, Staging and Prod powered by Solr, thus removing considerable load from your database so it can focus on more important things, like publishing 😊.

See [Architecture page](#) for further information on the architecture of eZ Platform.

## How to set up Solr Search engine

### Step 0: Enable Solr Bundle

#### Not needed with eZ Platform

This step is not needed as of eZ Platform 15.09, however it is kept here for reference in case you have previously disabled the bundle.

1. Check in `composer.json` if you have the `ezsystems/ezplatform-solr-search-engine` package, if not add/update composer dependencies:

#### command line

```
composer require --no-update
ezsystems/ezplatform-solr-search-engine:~1.0
composer update
```

2. Make sure `EzPublishSolrSearchEngineBundle` is activated with the following line in `app/AppKernel.php` file: `new EzSystems\EzPlatformSolrSearchEngineBundle\EzSystemsEzPlatformSolrSearchEngineBundle()`

### Step 1: Configuring & Starting Solr

Example here is for single core, look to [Solr documentation](#) for configuring Solr in other ways, also see the provided configuration for some examples.

#### In this topic:

- What is Solr Search Engine Bundle?
- How to set up Solr Search engine
  - Step 0: Enable Solr Bundle
  - Step 1: Configuring & Starting Solr
  - Step 2: Configuring bundle
  - Step 3: Configuring repository with the specific search engine
  - Step 4: Clear prod cache
  - Step 5: Run CLI indexing command
- Extending the Solr Search engine Bundle
  - Document Field Mappers
- Providing feedback

First download and extract Solr, **we currently support Solr 4.10.4**:

- [solr-4.10.4.tgz](#) or [solr-4.10.4.zip](#)

Secondly, copy configuration files needed for eZ Solr Search Engine bundle, *here from the root of your project to the place you extracted Solr*:

#### Command line example

```
# Make sure to change the /opt/solr/ path with where you
have placed Solr
cp -R
vendor/ezsystems/ezplatform-solr-search-engine/lib/Resources/config/solr/*
/opt/solr/example/solr/collection1/conf/

/opt/solr/bin/solr start -f
```

Thirdly, Solr Bundle does not commit solr index changes directly on repository updates, leaving it up to you to tune this using `solrconfig.xml` as best practice suggests, example config:

#### solrconfig.xml

```
<autoCommit>
  <!-- autoCommit is here left as-is like it is out of
the box in Solr 4.10.4, this controls hard commits for
durability/replication -->
  <maxTime>${solr.autoCommit.maxTime:15000}</maxTime>
  <openSearcher>false</openSearcher>
</autoCommit>

<autoSoftCommit>
  <!-- Soft commits controls mainly when changes becomes
visible, by default we change value from -1 (disabled)
to 100ms, to try to strike a balance between Solr
performance and staleness of HttpCache generated by Solr
queries -->
  <maxTime>${solr.autoSoftCommit.maxTime:100}</maxTime>
</autoSoftCommit>
```

## Step 2: Configuring bundle

The Solr search engine bundle can be configured many ways, in the config further below it assumes you have parameters setup for solr dsn and search engine (*however both are optional*), example:

#### parameters.yml

```
search_engine: solr
solr_dsn: 'http://localhost:8983/solr'
```

On to configuring the bundle.

## Single Core example (default)

Out of the box in eZ Platform the following is enabled for simple setup:

### config.yml

```
ez_search_engine_solr:
  endpoints:
    endpoint0:
      dsn: %solr_dsn%
      core: collection1
  connections:
    default:
      entry_endpoints:
        - endpoint0
      mapping:
        default: endpoint0
```

## Shared Core example

In the following example we have decided to separate one language as the installation contains several similar languages, and one very different language that should receive proper language analysis for proper stemming and sorting behavior by Solr:

config.yml

```
ez_search_engine_solr:
  endpoints:
    endpoint0:
      dsn: %solr_dsn%
      core: core0
    endpoint1:
      dsn: %solr_dsn%
      core: core1
  connections:
    default:
      entry_endpoints:
        - endpoint0
        - endpoint1
      mapping:
        translations:
          jpn-JP: endpoint1
          # Other languages, for instance eng-US and other
          western languages are sharing core
        default: endpoint0
```

## Multi Core example

If full language analysis features are preferred, then each language can be configured to separate cores.

*Note: Please make sure to test this setup against single core setup, as it might perform worse than single core if your project uses a lot of language fallbacks per siteaccess, as queries will then be performed across several cores at once.*

```

config.yml
ez_search_engine_solr:
  endpoints:
    endpoint0:
      dsn: %solr_dsn%
      core: core0
    endpoint1:
      dsn: %solr_dsn%
      core: core1
    endpoint2:
      dsn: %solr_dsn%
      core: core2
    endpoint3:
      dsn: %solr_dsn%
      core: core3
    endpoint4:
      dsn: %solr_dsn%
      core: core4
    endpoint5:
      dsn: %solr_dsn%
      core: core5
    endpoint6:
      dsn: %solr_dsn%
      core: core6
  connections:
    default:
      entry_endpoints:
        - endpoint0
        - endpoint1
        - endpoint2
        - endpoint3
        - endpoint4
        - endpoint5
        - endpoint6
      mapping:
        translations:
          - jpn-JP: endpoint1
          - eng-US: endpoint2
          - fre-FR: endpoint3
          - ger-DE: endpoint4
          - esp-ES: endpoint5
          # Not really used, but specified here for
          fallback if more languages are suddenly added by content admins
          default: endpoint0
          # Also use separate core for main languages
          (differs from content object to content object)
          # This is useful to reduce number of cores
          queried for always available language fallbacks
          main_translations: endpoint6

```

### Step 3: Configuring repository with the specific search engine

The following is an example of configuring Solr Search Engine, where connection name is same as in example above, and engine is set to solr:

### ezplatform.yml

```
ezpublish:
  repositories:
    default:
      storage: ~
      search:
        engine: %search_engine%
        connection: default
```

%search\_engine% is a parameter that is configured in `app/config/parameters.yml`, and should be changed from its default value "legacy" to "solr" to activate Solr as the Search engine.

## Step 4: Clear prod cache

While Symfony dev environment keeps track of changes to yml files, prod does not, so to make sure Symfony reads the new config we clear cache:

```
php app/console --env=prod cache:clear
```

## Step 5: Run CLI indexing command

### Make sure to configure your setup for indexing

Some exceptions might happen on indexing if you have not configured your setup correctly, here are the most common issues you may encounter:

- Exception if Binary files in database have an invalid path prefix
  - Make sure `ezplatform.yml` configuration `var_dir` is configured properly.
  - If your database is inconsistent in regards to file paths, try to update entries to be correct (*but make sure to make a backup first*).
- Exception on unsupported Field Types
  - Make sure to implement all Field Types in your installation, or to configure missing ones as `NullType` if implementation is not needed.
- Content not immediately available
  - Solr Bundle is on purpose not committing changes directly on Repository updates (*on indexing*), but letting you control this using Solr configuration. Adjust Solr **autoSoftCommit** (*visibility of change to search index*) and/or **autoCommit** (*hard commit, for durability and replication*) to balance performance and load on your Solr instance against needs you have for "NRT".
- Running out of memory during indexing
  - In general make sure to run indexing using prod environment to avoid debuggers and loggers from filling up memory.
  - Stash: Disable `in_memory` cache as recommended on [Persistence cache](#) for long running scripts.
  - Flysystem: An open issue exists where you can find further info <https://jira.ez.no/browse/EZP-25325>

Last step is to execute initial indexation of data:

```
php app/console --env=prod --siteaccess=<name>
ezplatform:solr_create_index
```

Since v1.7.0 the `ezplatform:solr_create_index` command is deprecated, use `php app/console ezplatform:reindex` instead:

```
php app/console --env=prod --siteaccess=<name>
ezplatform:reindex
```

## Extending the Solr Search engine Bundle

### Document Field Mappers

**V1.2+, AVAILABLE IN EZ PLATFORM V1.7+**

As a developer you will often find the need to index some additional data in the search engine. The use cases for this are wide, for example the data could come from an external source (*for example from recommendation system*), or from an internal source.

The common use case for the latter is indexing data through the Location hierarchy, for example from the parent Location to the child Location, or in the opposite direction, indexing child data on the parent Location. The reason might be you want to find the content with fulltext search, or you want to simplify search for a complicated data model.

To do this effectively, you first need to understand how the data is indexed with Solr Search engine. Documents are indexed per translation, as Content blocks. In Solr, a block is a nested document structure. In our case, parent document represents Content, and Locations are indexed as child documents of the Content. To avoid duplication, full text data is indexed on the Content document only.

Knowing this, you have the option to index additional data on:

- all block documents (meaning Content and its Locations, all translations)
- all block documents per translation
- Content documents
- Content documents per translation
- Location documents

Indexing additional data is done by implementing a document field mapper and registering it at one of the five extension points described above. You can create the field mapper class anywhere inside your bundle, as long as when you register it as a service, the "class" parameter in your `services.yml` matches the correct path. We have three different field mappers. Each mapper implements two methods, by the same name, but accepting different arguments:

- `ContentFieldMapper`
  - `::accept(Content $content)`
  - `::mapFields(Content $content)`
- `ContentTranslationFieldMapper`
  - `::accept(Content $content, $languageCode)`
  - `::mapFields(Content $content, $languageCode)`
- `LocationFieldMapper`
  - `::accept(Location $content)`
  - `::mapFields(Location $content)`

These can be used on the extension points by registering them with the container using service tags, as follows:

- all block documents
  - `ContentFieldMapper`
  - `ezpublish.search.solr.document_field_mapper.block`
- all block documents per translation
  - `ContentTranslationFieldMapper`
  - `ezpublish.search.solr.field_mapper.block_translation`
- Content documents
  - `ContentFieldMapper`
  - `ezpublish.search.solr.document_field_mapper.content`
- Content documents per translation
  - `ContentTranslationFieldMapper`
  - `ezpublish.search.solr.field_mapper.content_translation`
- Location documents

- LocationFieldMapper
- ezpublish.search.solr.field\_mapper.location

The following example shows how to index data from the parent Location content, in order to make it available for full text search on the children content. A concrete use case could be indexing webinar data on the webinar events, which are children of the webinar. Field mapper could then look something like this:

```
<?php

namespace My\WebinarApp;

use
EzSystems\EzPlatformSolrSearchEngine\FieldMapper\Content
FieldMapper;
use eZ\Publish\SPI\Persistence\Content\Handler as
ContentHandler;
use eZ\Publish\SPI\Persistence\Content\Location\Handler
as LocationHandler;
use eZ\Publish\SPI\Persistence\Content;
use eZ\Publish\SPI\Search;

class WebinarEventTitleFulltextFieldMapper extends
ContentFieldMapper
{
    /**
     * @var
     \eZ\Publish\SPI\Persistence\Content\Type\Handler
     */
    protected $contentHandler;

    /**
     * @var
     \eZ\Publish\SPI\Persistence\Content\Location\Handler
     */
    protected $locationHandler;

    /**
     * @param
     \eZ\Publish\SPI\Persistence\Content\Handler
     $contentHandler
     * @param
     \eZ\Publish\SPI\Persistence\Content\Location\Handler
     $locationHandler
     */
    public function __construct(
        ContentHandler $contentHandler,
        LocationHandler $locationHandler
    ) {
        $this->contentHandler = $contentHandler;
        $this->locationHandler = $locationHandler;
    }

    public function accept(Content $content)
    {
        // ContentType with ID 42 is webinar event
        return
        $content->versionInfo->contentInfo->contentTypeid == 42;
    }
}
```

```
public function mapFields(Content $content)
{
    $mainLocationId =
$content->versionInfo->contentInfo->mainLocationId;
    $location =
$this->locationHandler->load($mainLocationId);
    $parentLocation =
$this->locationHandler->load($location->parentId);
    $parentContentInfo =
$this->contentHandler->loadContentInfo($parentLocation->
contentId);

    return [
        new Search\Field(
            'fulltext',
            $parentContentInfo->name,
            new Search\FieldType\FullTextField()
        ),
```



```
    ];  
    }  
}
```

Since we index full text data only on the Content document, you would register the service like this:

```
my_webinar_app.webinar_event_title_fulltext_field_mapper  
:  
    class:  
    My\WebinarApp\WebinarEventTitleFulltextFieldMapper  
    arguments:  
        - '@ezpublish.spi.persistence.content_handler'  
        - '@ezpublish.spi.persistence.location_handler'  
    tags:  
        - {name:  
            ezpublish.search.solr.field_mapper.content}
```

## Providing feedback

After completing the installation you are now free to use your site as usual. If you get any exceptions for missing features, have feedback on performance, or want to discuss, join our community slack channel at <https://ezcommunity.slack.com/messages/ezplatform-use/>