Configuration Management in Drupal 8

Fabian Bircher - fabian@nuvole.org

Nuvole

a 100% Drupal company

Our Distributed Team

Italy

Belgium

Czech Republic

Our Clients

Our Projects

- International organisations
- Institutions
- Fast delivery: several developers working simultaneously on the same site
- Frequent configuration changes: need for safe updates

Challenges We Face

- Remote collaboration on site development
- Keeping track of all configuration changes during development
- Pushing upgrades to production sites

Chapter 1

The Evolution of

Code-Driven Development in Drupal 8

The long march to a "code-driven" Drupal

- Historically, Drupal has kept both configuration and content in the same database.
  Every time you click in the administrative interface, no record is kept.
- **Drupal 6**: Features appears, with the possibility to export configuration to PHP code.
- **Drupal 7**: Features support is mature, but still relying on third parties and incomplete
- **Drupal 8**: Configuration and content are separated, configuration is text-based.

The database-driven workflow disadvantages

- Default in Drupal 6 and Drupal 7 (core)
- Standard approach: you click, you save, Drupal saves to database and forgets.
- Bad: Mixing configuration and content.
- Bad: Losing track of configuration changes.
- Theoretically still possible in Drupal 8!

The Features-driven workflow drawbacks in D7

- A fundamental structural flaw: to package configuration into modules, you need to make it exportable to "code" (text files)
- Features is very good for packaging, not as good for exporting; but there's no packaging without exporting
- Not everything is exportable/traceable.
- You must "whitelist" elements to be tracked: you never have the whole site under control.

Code-driven is not just Features

- It’s a global technical choice.
- For example, it includes makefiles and profiles.
- Keywords: text-based configuration, traceability, repeatability, reuse.

New in D8: Configuration Management System

Formely known as Configuration Management Initiative (CMI)

Reference Use Case

Modify the configuration of a production site:

- Keeping the site online all the time.
- Developing/testing the new configuration on a development copy.
- Exporting the configuration changes from development and importing them into production.

A guided example of Configuration Management

🎬 Step 1 of 6

Clone Site to Dev

Production

- Install Site.
- Full backup:
  - Database.
  - Full Drupal tree.
  - Files.

Development

- Restore the backup.

🎬 Step 2 of 6

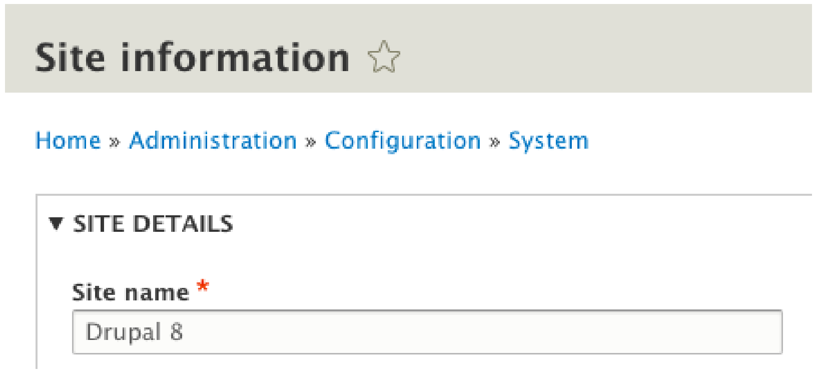
Modify Configuration

Production

Site operates normally:

- new users.
- new content.

Development



Site information ☆

Home » Administration » Configuration » System

▼ SITE DETAILS

Site name *

Drupal 8

🎬 Step 3 of 6

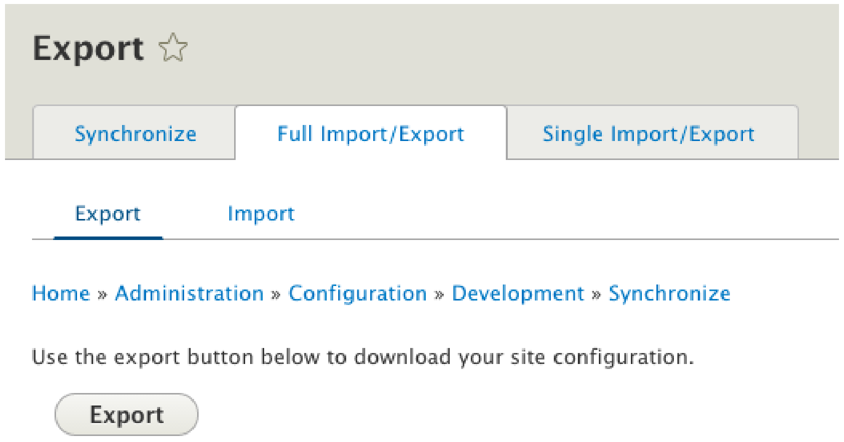
Export Configuration

Production

Site operates normally:

- new users.
- new content.

Development


Export ☆

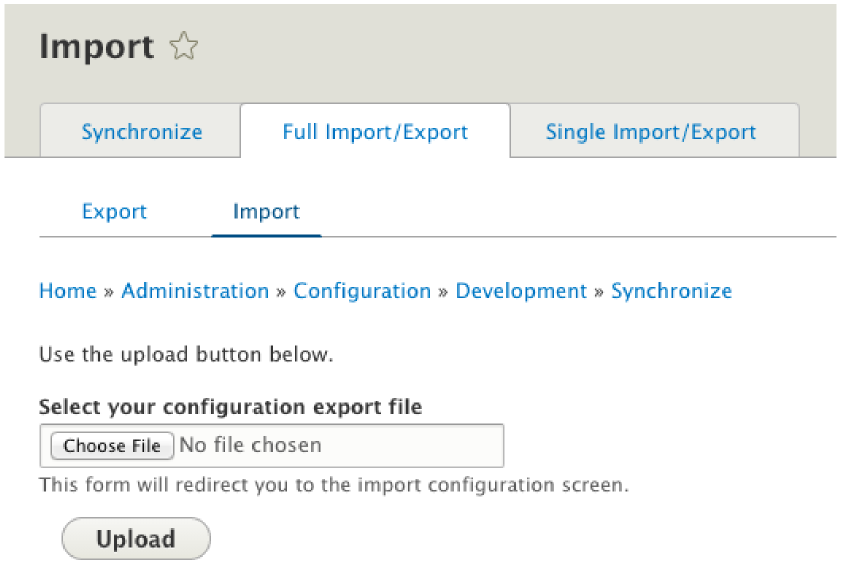
Home » Administration » Configuration » Development » Synchronize

Use the export button below to download your site configuration.

Step 4 of 6

Import into Staging

Production



Import ☆

Home » Administration » Configuration » Development » Synchronize

Use the upload button below.

Select your configuration export file

Choose File No file chosen

This form will redirect you to the import configuration screen.

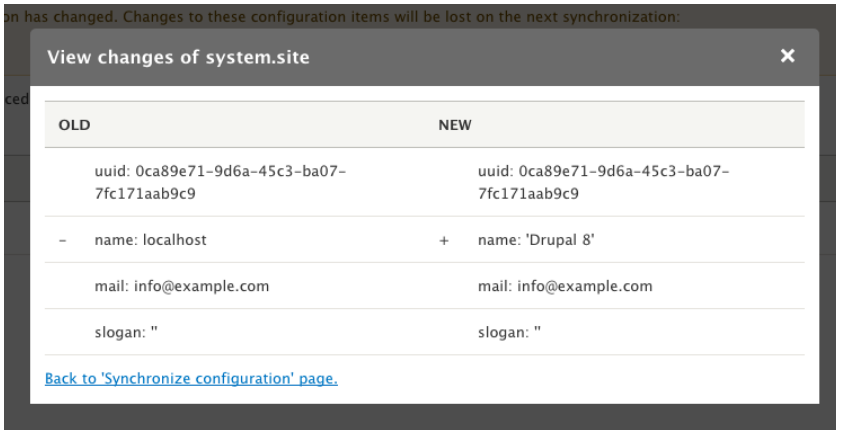
Development

Development goes on normally.

Step 5 of 6

Review Changes

Production



n has changed. Changes to these configuration items will be lost on the next synchronization:

ced

View changes of system.site ×

	OLD		NEW
	uuid: 0ca89e71-9d6a-45c3-ba07-7fc171aab9c9		uuid: 0ca89e71-9d6a-45c3-ba07-7fc171aab9c9
-	name: localhost	+	name: 'Drupal 8'
	mail: info@example.com		mail: info@example.com
	slogan: ''		slogan: ''

Back to 'Synchronize configuration' page.

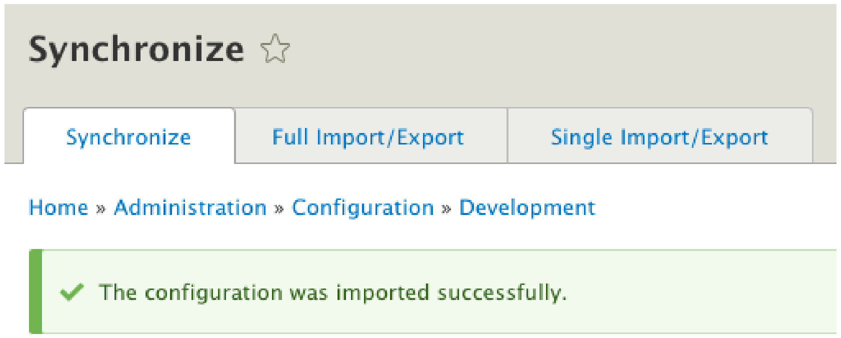
Development

Development goes on normally.

Step 6 of 6

Apply Changes

Production



✓ The configuration was imported successfully.

Development

Development goes on normally.

How this would have worked in Drupal 7

- Clone site to development environment.
- Create a feature exporting the site name variable.
- Development: update the feature.
- Transfer the feature to Production.
- Production: enable/revert the feature.

Chapter 2

A closer look at Configuration Management

Configuration Manager

- `config` core module.
- Provides **import/export** functionality for site configuration.
- Allows to deploy configuration from one environment to another, provided they are the same site.

Configuration

- **Original** configuration can be provided by profiles, modules, and themes.
- Configuration is stored in code, in `YAML` files, one per configuration object.
- **Original** configuration becomes **active** configuration after installation.
- **Active** configuration is stored in the database by default.

A sample YAML file

system.site.yml

```
uuid: f67e15b4-c824-4e7d-af00-0e9b2fe34814
name: 'D8 test'
mail: site@example.com
slogan: ''
page:
403: ''
404: ''
front: /node
admin_compact_mode: false
weight_select_max: 100
langcode: en
default_langcode: en
```

Configuration stores

- The **active store** is the actual site configuration (without possible overrides)
- The **staging store** is used for temporary storage.
- The two have the same structure.

Original configuration

- Defined in the `config/install` sub-directory.
- One file per configuration item.
- Imported when module is enabled.
- Then **fully owned** by the site (original files are ignored)

Configuration dependencies

```
core.entity_view_display.node.article.default.ym
```

```
dependencies:
  config:
    - field.field.node.article.body
    - field.field.node.article.comment
    - field.field.node.article.field_image
    - field.field.node.article.field_tags
    - node.type.article
  module:
    - comment
    - image
```

Importing, exporting, and synchronizing configuration

- Exported configuration will be stored in **staging** directory.
- Staged configuration can be imported to become **active** configuration.
- Once import is run, new modules are enabled, new fields, content types, etc. are added, in short all changes are live.

Optional configuration

- Defined in the `config/optional` directory.
- Depends on other modules.
- Imported when a module is enabled and/or the relevant dependency is enabled.
- Example: the node module and the views it ships with.

Optional configuration

Optional configuration is installed based on what's specified in the schema, for ex. `views.schema.yml` (from Views module)

```
...
views.view.*:
  type: config_entity
  label: 'View'
```

Meaning: a module can provide a default view in a file named `views.view.frontpage.yml`. Files with this naming pattern (`views.view.*`) are installed only if/when the Views module is enabled.

Setting up staging store

- The default location for the staging directory is inside a randomly-named directory in the public files path
- Convention: change staging config directory in `sites/default/settings.php`

```
$config_directories['staging'] = 'sites/default/config/staging';
```

Configure it so that it is git-versioned and protected.

`drush`

- **DRU**pal **SH**ell that everybody loves!
- Drupal 8 is supported by `drush 8.0.x-dev`
- Both not stable yet.
- Drush should now be installed with `composer`

`drush`

In Drupal 8 we rebuild the cache, even after restoring a database dump:

```
$ drush sql-drop -y
$ drush cr
```

It replaces `drush cc all`

Step 3 of 6 — Drush Style

Export Configuration

Development

```
$ drush config-export
The current contents of your export directory
(sites/default/config/staging) will be deleted. (y/n): y

Configuration successfully exported to [success]
sites/default/config/staging.
```

🎬 Step 5 of 6 — Drush Style

Review Changes

Production

```
$ drush config-import --preview=diff
Configuration successfully exported to /tmp/drush_tmp_xy. [success]

diff -u /tmp/drush_tmp_xy/system.site.yml sites/.../staging/system.site.yml
--- /tmp/drush_tmp_xy/system.site.yml
+++ sites/default/config/staging/system.site.yml
@@ -1,5 +1,5 @@
uuid: ca04efa4-51bf-4d12-8b00-e7b244b97aef
-name: 'localhost'
+name: 'Drupal 8'
mail: info@example.com
slogan: ''
page:
Import the listed configuration changes? (y/n):
```

Chapter 3

Configuration in Drupal 7 vs Drupal 8

State of the art in Drupal 8

- The current state of **D8 + modules** allows same productivity of **D7 + Features**
- **Drupal 7**: one multi-purpose tool (Features), mixed success.
- **Drupal 8**: several dedicated tools/modules, in general working better.

Use case

D7: Features

- A collection of logically related Drupal elements.
- Packaged into PHP code, using hooks.
- Exportability is a precondition to packaging.

D8: Config Management

Reference for the whole site configuration, development to production

Configuration format

D7: Features

PHP

- Imperative.
- Interpreted
  Can break site if corrupted
- Located in folders for modules.
- Treated as modules.

D8: Config Management

YAML

- Declarative.
- Parsed
  Cannot break anything if corrupted
- Located in specific folders for config.
- Treated as data
  Like Rules' JSON in D7

Support

D7: Features

Optional

- Modules must offer support for Features.
- No guarantees.

D8: Config Management

Mandatory

- Core configuration.
- The only way to supply configuration.

🔧 Configuration and modules

D7: Features

- Features are special modules.
- Once a Feature is enabled, its configuration is tracked forever.

D8: Config Management

- Modules provide initial values.
- In this sense, every module is a Feature.
- Configuration is decoupled from modules after installation (site owns configuration)

Components selection

D7: Features

- Explicitly listed in info file.
- Rest is not tracked.

D8: Config Management

- All configuration is tracked.
- Configuration is saved per config item.
- Can be individually imported/exported.
- Config synchronisation requires all files to be present (missing = deleted)

Configuration staging

D7: Features

- Feature states: normal, overridden, needs review.
- Operations: features update/revert.
- Diff available.

D8: Config Management

- Active store and staging store (multiple stores possible)
- Operations: import and export.
- Diff available.

Drush workflow

D7: Features

- `drush fu`
- `drush fr`

D8: Config Management

- `drush cex`
- `drush cim`

(With Drush 8.0.x-dev)

Cross-site compatibility

D7: Features

- Write once, deploy anywhere.
- A feature is ready to be deployed on multiple sites.

D8: Config Management

- Specific to multiple instances (dev, prod) of the **same** site.
- Cross-site is **not** the CMI use case.
- Configuration Management relies on UUIDs.

Boundaries of configuration

D7: Features

- Entities through entity api, CTools plugins.
- Variables with Strongarm.
- Content with `features_uuid`
- Menu links, custom and contrib modules can be problematic.

D8: Config Management

- Configuration.
- Content.
- State.
- All clearly defined.

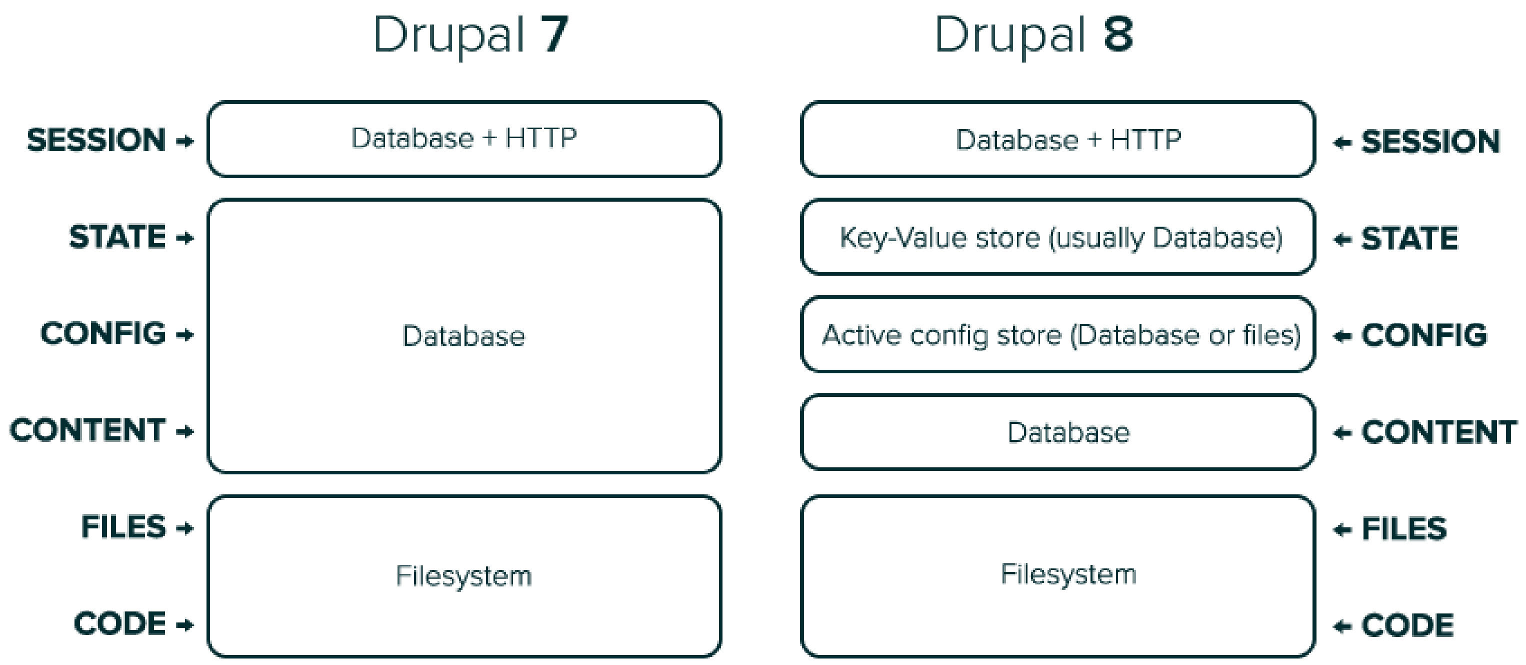
Features 8.x-3.x exists!

- Still under development, solves the **packaging** problem.
- Interface much similar to the D7 one, inner working much different.
- Usage pattern has changed: do not use for deployment!

Chapter 4

Configuration API

An overview


Drupal 7
Drupal 8
SESSION
STATE
CONFIG
CONTENT
FILES
CODE
Database + HTTP
Database
Filesystem
Database + HTTP
Key-Value store (usually Database)
Active config store (Database or files)
Database
Filesystem
SESSION
STATE
CONFIG
CONTENT
FILES
CODE

Configuration entities and objects

Configuration items, as seen from PHP, can be either:

- **Configuration Objects**: for simpler configuration (like variables), extend `ConfigBase`
- **Configuration Entities**: for configuration items like Views or Date Format, extend `ConfigEntityBase`

Configuration Entities

- In Drupal 8 entities are used for both content and configuration.
- Configuration Entities extends `ConfigEntityBase` class.
- Two namespaces: one for config, one for content.

```
class DateFormat extends ConfigEntityBase implements DateFormatInterface {...}


class Node extends ContentEntityBase implements NodeInterface {...}
```

Mutable and immutable configuration

- **Immutable**: retrieved in read-only mode

```
$config = \Drupal::config('system.site');
```

- **Mutable**: retrieved in read-write mode (use to set values)

```
$config = \Drupal::configFactory()->getEditable('system.site');
```

Working with configuration

Reading and writing configuration

```
// Get site name.
$site_name = \Drupal::config('system.site')->get('name');
// Set site name.
\Drupal::configFactory()->getEditable('system.site')
->set('name', 'My site')->save();
```

`variable_get()` and `variable_set()` died.

Overriding "on the fly"

- The `$conf` array is still available as `$config`
- Useful in `settings.local.php`:
  Differentiate **development** and **production** environment

Overridden configuration

- Scenario: a module provides configuration, a forced setting (like `$config` or GUI) overrides it.
- The configuration object contains original and overrides.
- The GUI form does not show the overridden value.

Example: Site name was edited in GUI.

```
$original = \Drupal::config('system.site')->getOriginal('name', FALSE);

$overridden = \Drupal::config('system.site')->get('name');
```

D7 variables: State or Config?

Instance-specific	Last cron run	➡	State
Configuration	Site mail	➡	Config

Site = filesystem + content + configuration + state

Working with states

Reading and writing states

```
// Get last cron run timestamp.
$time = \Drupal::state()->get('system.cron_last');
// Set cron run timestamp.
\Drupal::state()->set('system.cron_last', REQUEST_TIME);
```

Chapter 5

Demo!

Chapter 6

Features for Drupal 8

Features for Drupal 8

- Depend on Configuration Update Manager.
- Focus on packaging configuration for reuse purpose only.
- Meant to be a development module: features do not depends on Features module.

What’s new?

- Assignment plugin.
- Features bundles.
- User interface is provided by a separate module, included in the project package.
- Naming conventions enforcement.

The new role of Features

- A module for developers.
- Administration interface at `config/development/configuration/features` rather than under structure.
- Don’t use for deployment in production (D7 way)
- Don’t even enable it in production (use CM)

Naming conventions enforcement

- Made possible by using bundles.
- Allows to export configuration to a different bundle, making it easy to copy features from one namespace to another.

Drush workflow

- Unchanged: `drush fc`, `drush fu`, `drush fr`, `drush fd`
- `drush features` becomes `drush fl`
- `--add-profile` add features to an install profile.

Chapter 7

A taste of

Paradigm shift

Multi developer workflow

- Configuration needs to be exported and imported!
- Version **all** configuration in git. (current site config state)
- Commit to git before synchronizing. (as a backup)
- Import merged configuration before continuing.

Δ config = development

- Lock configuration changes on the live site. config_readonly (https://www.drupal.org/project/config_readonly)
- If locking is not an option: export and commit to a dedicated branch, so developers can merge it into the configuration which will be deployed.
- Best practices yet to be found. Join groups (https://groups.drupal.org/node/466373)

Features workflow

- If you use features 8.x for deployment
  ⇒ you are doing it wrong. ™
- Re-use configuration for other projects!
- Synchronize partial configuration between **different sites**.
- Use features in development environments.

Features workflow

Project A		**Project B**
Production		Production
Staging		Staging
Development	← features →	Development

Thank you!