



From Drupal 7 to Drupal 8

Features vs. Configuration Management

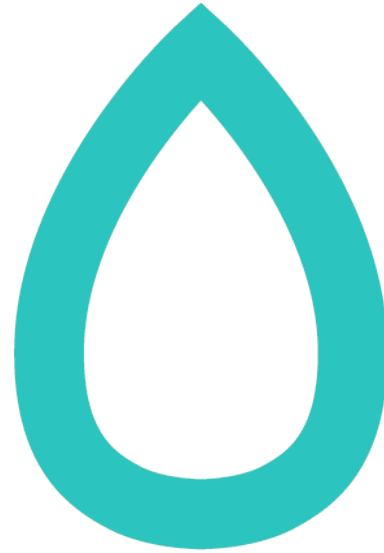
Fabian Bircher - fabian@nuvole.org

0

 <http://nuvole.org>

   [nuvoleweb](#)

Nuvole



a 100% Drupal company



Our Distributed Team



Italy



Belgium



**Czech
Republic**

Our Clients





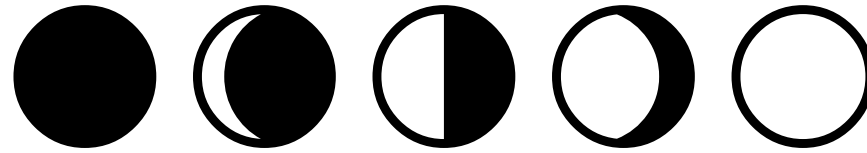
Our Projects

- 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

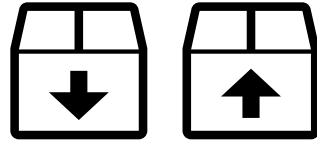
The long march to a "code-driven" Drupal

- 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



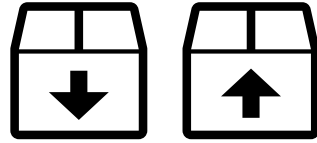
The database-driven workflow disadvantages

- 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 disadvantages

- A structural flaw: to package configuration into modules, you need to make it exportable to "code"
- 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, still applicable in D8
- Keywords: text-based configuration, traceability, repeatability, reuse



New in Drupal 8

Configuration Management



A guided example of Configuration Management



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.

 Step 1 of 6

Clone Site to Dev

Production

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

Development

- Restore the backup
- (or install with config_installer)

Modify Configuration

Production

- Site operates normally:
 - new users
 - new content

Development

Site information

[Home](#) » [Administration](#) » [Configuration](#) » [System](#)

▼ SITE DETAILS

Site name*

Drupal 8

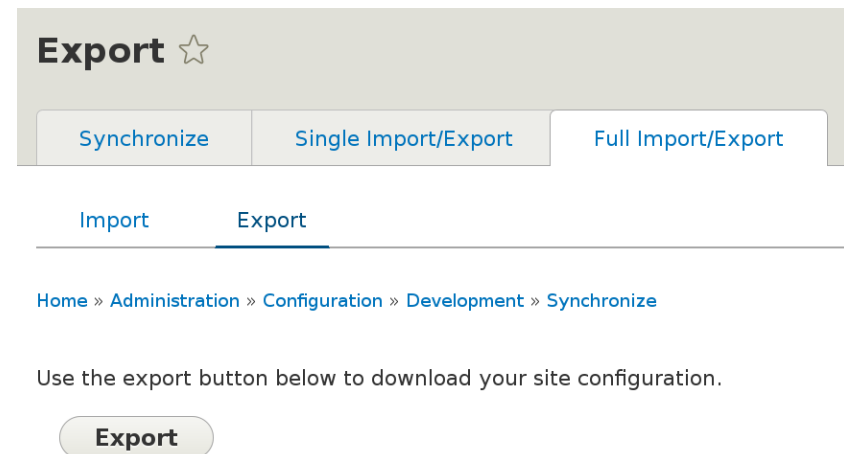
 Step 3 of 6

Export Configuration

Production

- Site operates normally:
 - new users
 - new content

Development



The screenshot shows a web interface for the 'Export' configuration page. At the top, there is a header 'Export' with a star icon. Below the header, there are three buttons: 'Synchronize', 'Single Import/Export', and 'Full Import/Export'. Underneath these buttons, there are two tabs: 'Import' and 'Export', with 'Export' being the active tab. A breadcrumb trail is visible: 'Home » Administration » Configuration » Development » Synchronize'. Below the breadcrumb, there is a text instruction: 'Use the export button below to download your site configuration.' At the bottom, there is a prominent 'Export' button.

 Step 4 of 6

Import into Staging

Production

Development

Import

[Synchronize](#) [Single Import/Export](#) [Full Import/Export](#)

[Import](#) [Export](#)

[Home](#) » [Administration](#) » [Configuration](#) » [Development](#) » [Synchronize](#)

Use the upload button below.

Select your configuration export file

Nessun file selezionato.

This form will redirect you to the import configuration screen.

 Step 5 of 6

Review Changes

Production

Development

OLD	NEW
uuid: 29d3d74e	uuid: 29d3d74e
- name: localhost	+ name: 'Drupal 8'
slogan: ""	slogan: ""

 Step 6 of 6

Apply Changes

Production

Development

Synchronize ☆

[Synchronize](#) [Single Import/Export](#) [Full Import/Export](#)

[Home](#) » [Administration](#) » [Configuration](#) » [Development](#)


✓ The configuration was imported successfully.

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

drush


- DRUpal SHell: everybody loves!
- Drupal 8 ↔ drush 7
- Both not stable yet, but matching beta releases
- A new installation method: composer

 Step 3 of 6 — Drush Style

Export Configuration

Development

```
$ drush config-export  
The current contents of your export directory  
(sites/default/files/config_H6raw/staging) will be deleted. (y/n): y  
Configuration successfully exported to [success]  
sites/default/files/config_H6raw/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/files/config_H6raw/staging/system.site.yml
@@ -1,5 +1,5 @@
  uuid: ca04efa4-51bf-4d12-8b00-e7b244b97aef
  -name: 'Drupal 8'
  +name: 'DDD 2015'
  mail: drupal@example.com
  slogan: ''
  page:
Import the listed configuration changes? (y/n):
```


Chapter 2



Inner workings of

Configuration Management and Features



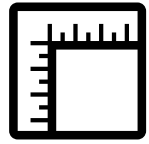
Use case

Features

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

Configuration Management

Reference for the whole site configuration, development to production



Configuration format

PHP

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

YAML

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

Support

Optional

- Modules must offer support for Features
- No guarantees

Mandatory

- Core configuration
- The only way to supply configuration



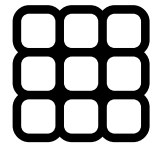
Configuration and modules

Drupal 7

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

Drupal 8

- Modules provide initial values
- In this sense, every module is a Feature
- Configuration is decoupled from modules after installation



Components selection

Drupal 7

- Explicitly listed in info file
- Rest is not tracked

Drupal 8

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



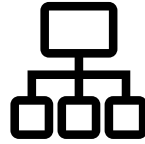
Configuration staging

Drupal 7

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

Drupal 8

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



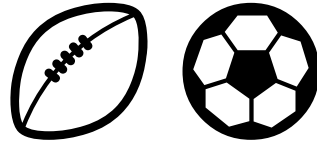
Drush workflow

Drupal 7

- drush features-update
- drush features-revert

Drupal 8 (with drush 7.x-dev)

- drush cex
- drush cim



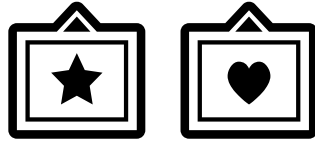
Cross-site compatibility

Drupal 7

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

Drupal 8

- Specific to multiple instances (dev, prod) of the **same** site
- This is the CMI use case
- Configuration Management relies on UUIDs



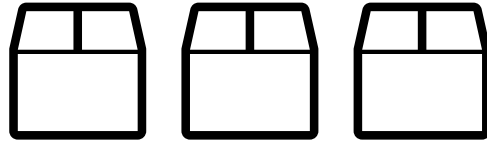
Boundaries of configuration

Drupal 7

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

Drupal 8

- Configuration
- Content
- State
- All clearly defined



Features done right?

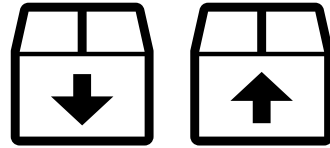
Drupal 7

- Not optimal
- A bit "forced" at times since it is not native in Drupal 7

Drupal 8

- Clean
- Native
- Not as powerful as D7+Features yet

Chapter 3



How to build

Re-usable components



Use case

Drupal 7: Features

- Package configuration
- Reuse configuration
- Focus on efficiency

Drupal 8: CM

- Deploy configuration
- Tracking configuration
- Focus on reliability

We want both, **efficiency** and **reliability**

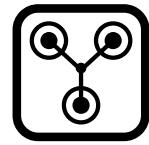
Drawbacks

Drupal 7: Features

- Not designed for deployment
- Used for deployment
- It drives people crazy

Drupal 8: CM

- Designed for deployment
- Not designed for packaging
- Possible shortcomings when creating re-usable configuration.



Features 8.x-3.x

- Goes back to the core mission of Features: package functionality for re-use.
- No more project specific features!
- Under development, alpha available.
- Phase2 blog post [Announcing Features for Drupal 8](http://www.phase2technology.com/blog/announcing-features-for-drupal-8/) (<http://www.phase2technology.com/blog/announcing-features-for-drupal-8/>)
- Development module, not for production sites using CMI

Chapter 4

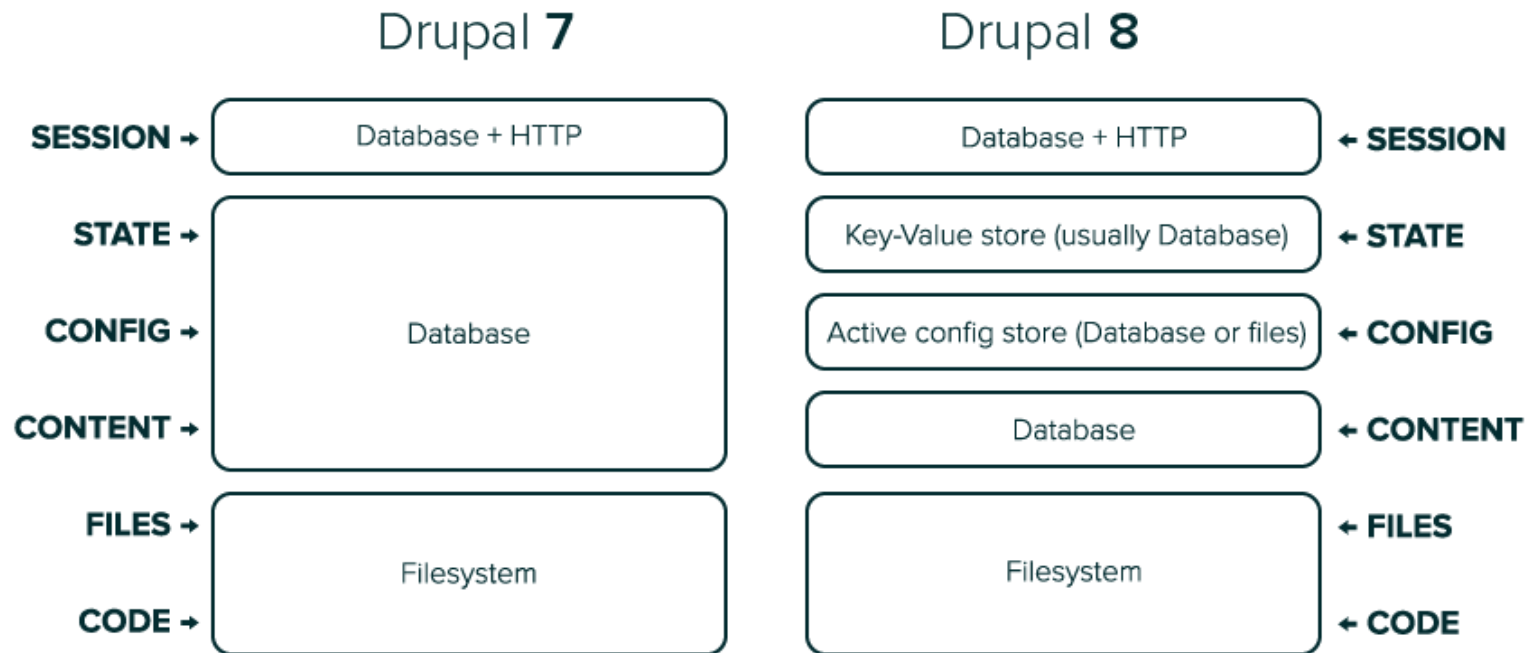


Configuration Management

For Developers

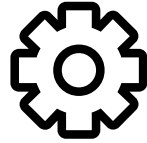


An overview



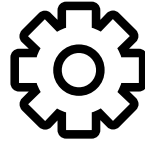
API import

- The Config namespace configuration must always be imported/exported through the API, never edit the files!
- The filesystem is not a good option, and this is the reason for Drupal 8 to store the active config in database by default:
 - Performance
 - Safety
 - Security



Configurables in Drupal 8

- `ConfigEntityBase` class
- Two entity namespaces: one for config, one for content



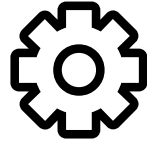
Working with configuration

Reading and writing configuration

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



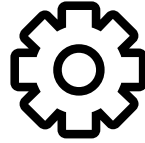
`variable_get()` and `variable_set()` died.



Information about the system state

- instance-specific? (e.g., last cron run) → state
- configuration? (e.g., site mail) → config

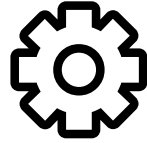
Site = filesystem + content + configuration + state



Working with states

Reading and writing states

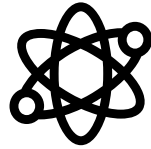
```
// Get last cron run timestamp.  
$time = \Drupal::state()->get('system.cron_last');  
// Set cron run timestamp.  
\Drupal::state()->set('system.cron_last', REQUEST_TIME);
```



Overriding "on the fly"

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

Chapter 8



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.
([patch \(https://www.drupal.org/node/2416109\)](https://www.drupal.org/node/2416109) pending)

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





Thank you!