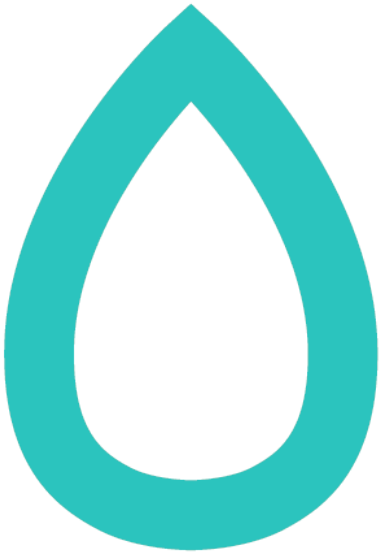
Configuration Management in Drupal 8

Antonio De Marco - antonio@nuvole.org

Fabian Bircher - fabian@nuvole.org

http://nuvole.org nuvoleweb

Nuvole



a 100% Drupal company

Our Distributed Team

Italy

Belgium

Czech Republic

👍 Our Clients

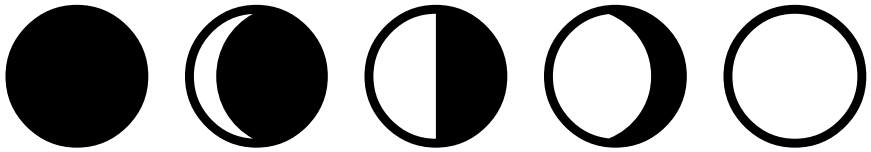
Our Projects

- International organisations
- Institutions
- Fast delivery: several developers working simultaneously on the same project
- 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

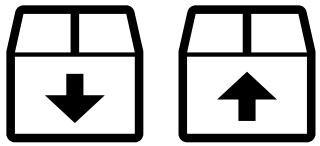
Configuration Management in Drupal

 The long march to config management in 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. (imperative)
- **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. (declarative)

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!



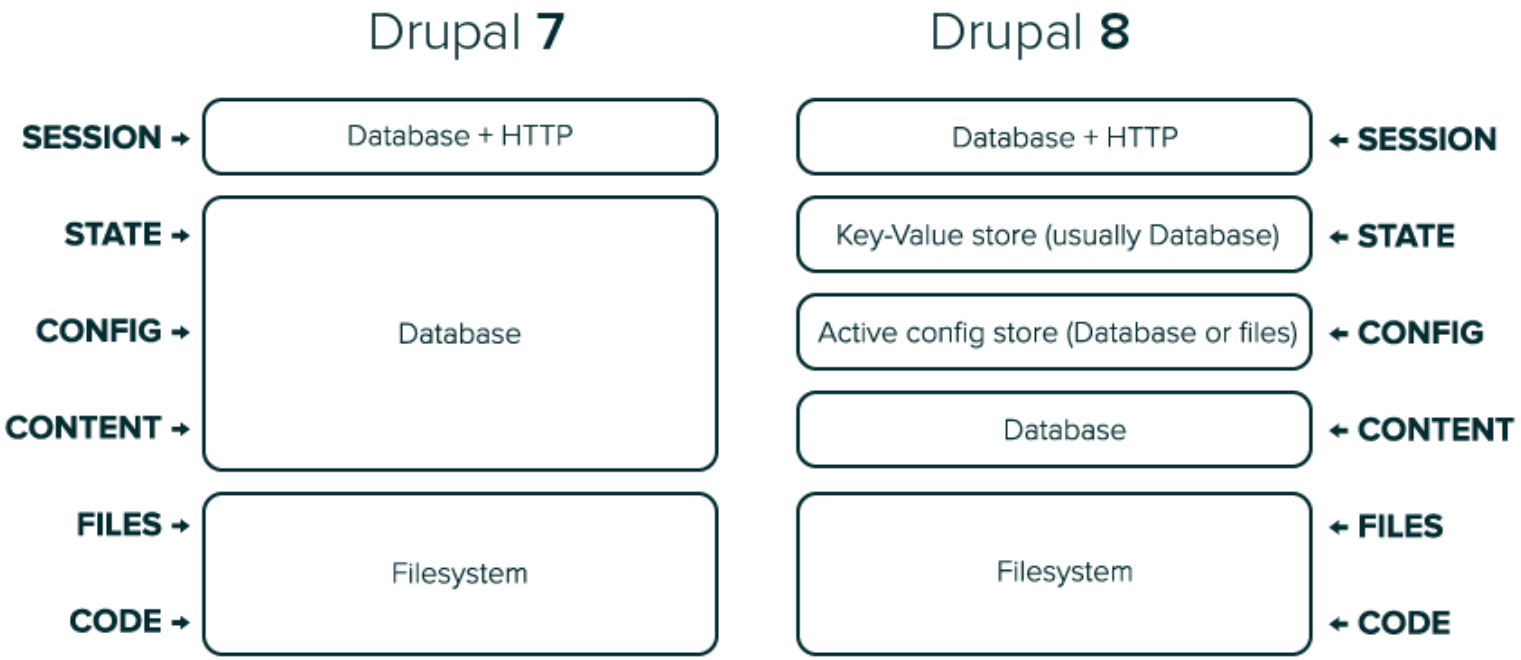
The Features-driven workflow drawbacks in D7

- A fundamental structural flaw: to package configuration into modules, you need to make it exportable to "code"
- 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.
- (Theoretically) still possible in Drupal 8!

Chapter 2

A closer look at Configuration Management

An overview


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

Configuration Management

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

Configuration

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

A sample YAML file

system.site.yml

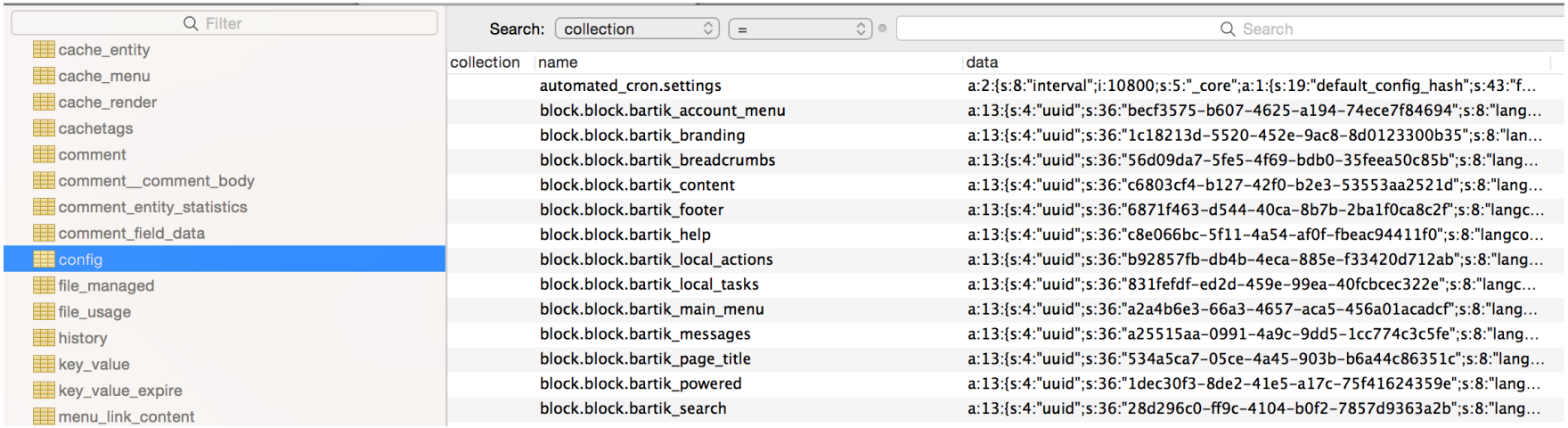
```
uuid: d1329a45-acab-4599-bce9-972d6707b1e6
name: 'Drupal 8'
mail: info@example.com
slogan: ''
page:
  403: ''
  404: ''
  front: /node
admin_compact_mode: false
weight_select_max: 100
langcode: en
default_langcode: en
_core:
  default_config_hash: yXadRE77Va-G6dxhd2kPYapAvbnSvTF6hO4oXiOEynI
```

Configuration stores

- **Active store**: the actual site configuration.
- **Sync store**: used for temporary storage.
  Initially empty it's populated at the first configuration export
- The two have the same structure.

Active store

The **active store** is in the database by default.



Sync store

The **sync store** is in the filesystem.

Setting up sync store

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

```
$config_directories = array(
  CONFIG_SYNC_DIRECTORY => '/directory/outside/webroot',
);
```

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

Importing, exporting, and synchronizing configuration

- Exported configuration will be stored in **sync** 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.
- **IMPORTANT:** importing and exporting entirely wipes out the current configuration.

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
    - text
    - user
```

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)
- Needs to depend on module that provides it.

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.

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.
- **NOT** transfering partial config between different sites.

🎬 Step 1 of 6

Clone Site to Dev

Production

- Install Site.
- Make a full backup:
  - Database.
  - Files.

Development

- Clone production site by restoring the backup, or...
- Install it with the **Configuration installer** project.

🎬 Step 2 of 6

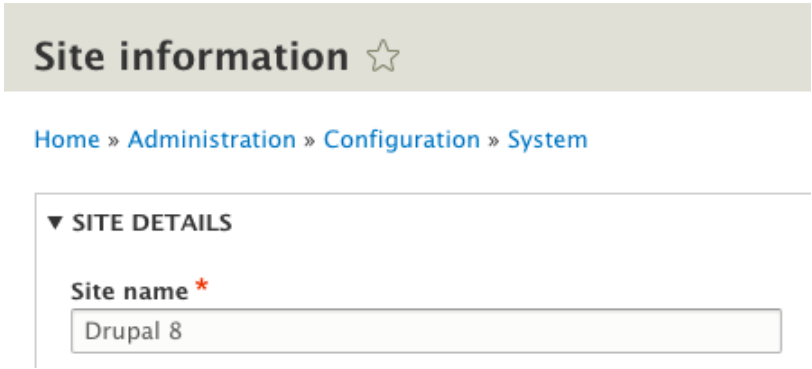
Modify Configuration

Production

Site operates normally:

- new users.
- new content.

Development



Configuration changes...

🎬 Step 3 of 6

Export Configuration

Production

Site operates normally:

- new users.
- new content.

Development

```
$ drush cex
Differences of the active config to th
e export directory:

 Collection  Config       Operation
             system.site  update
The .yml files in your export director
y (config/sync) will be deleted and re
placed with the active config. (y/n):
```

Then `$ git commit && git push`

🎬 Step 4 of 6

Import into Sync

Production

```
$ git pull
```

Development

Development goes on normally.

Step 5 of 6

Review Changes

Production

```
$ drush config-import --preview=diff
Only in config/sync: .htaccess
...
 uuid: d1329a45-acab-4599-bce9-972d670
7b1e6
-name: 'localhost'
+name: 'Drupal 8'
 mail: info@example.com
 slogan: ''
 page:
Import the listed configuration change
s? (y/n):
```

Development

Development goes on normally.

Step 6 of 6

Apply Changes

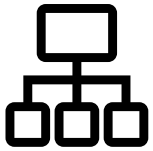
Production

```
$ drush cim
 Collection  Config       Operation
             system.site  update
Import the listed configuration change
s? (y/n):
```

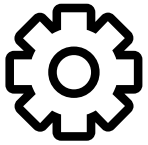
Development

Development goes on normally.

Chapter 3



Configuration API



Configuration items

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

- **Configuration Objects**:
  For singular configuration items (like system.site.yml)
  Extend `\Drupal\Core\Config\ConfigBase`
- **Configuration Entities**:
  For multiple configuration items like Views or Date Format
  Extend `\Drupal\Core\Config\Entity\ConfigEntityBase`

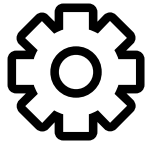
Mutable and immutable configuration

- **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');
```



Overriding "on the fly"

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

```
$config['system.logging']['error_level'] = 'verbose';
```

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.

```
$config['system.site']['name'] = 'New title';
```

```
$original = \Drupal::config('system.site')->getOriginal('name', FALSE);
// Returns "Drupal 8".

$overridden = \Drupal::config('system.site')->get('name');
// Returns "New title".
```

Chapter 4

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)

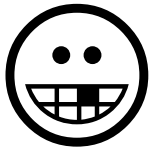
Features workflow

- 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

Chapter 5



Working with

Configuration Management

Working with

Team of developers

- Share a Git repository for both code and configuration
- Install site starting from initial configuration.
- Adopt “A successful Git branching model” (cit.)

Project bootstrap

First developer:

- Installs site locally.
- Exports configuration to **sync**
- Commits and pushes to shared Git repository.

Other developers and Prod:

- Clone code.
- Have `config_installer` profile available.
- Install site starting from exported configuration.

Day-yo-day development as a team

Developer A

```
$ drush config-export
$ git add && git commit
$ git push
```

Developer B

```
$ drush config-export
$ git add && git commit
$ git push 💀 !*?$*(%!)
$ git pull
```

Fetch, Merge, Resolve.

```
$ drush config-import
```

Inspect and fix.

```
$ drush config-export
```

Repeat.

Production deployment

1. Pull code and configuration from `master` branch.
2. Run updates: `drush updatedb`.
3. Import configuration: `drush cim`.

Important: the order must be respected.
An issue (https://www.drupal.org/node/2628144) on d.o will enforce it.

Note: this assumes no configuration changes on production.

Working with

Changes on production

- Same as **Team of developers**, but this time...
- Configuration on production is changed by Geeky Client™ overnight, without notice.

Changes on production = development

Options for developer's sanity:

- **Option 1**: Lock configuration changes on the live site.
- **Option 2**: Export and commit to a dedicated branch.
  Ideally this would be the responsibility of a dedicated person.

Option 1

Locking configuration on production

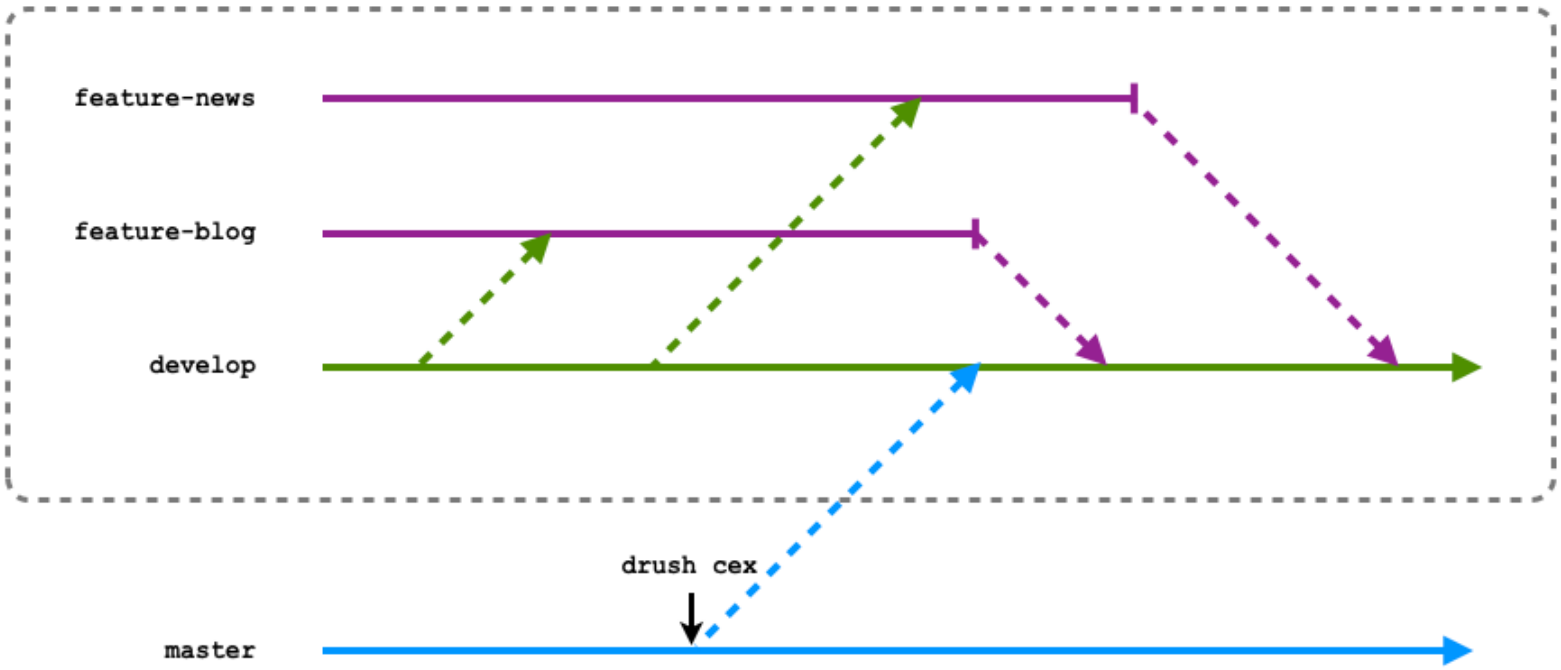
Don’t allow config changes on the production site if ever possible by installing the config_readonly (https://www.drupal.org/project/config_readonly) module.

Note: add this to `settings.php` in production:

```
$settings['config_readonly'] = TRUE;
```

Option 2

Export to a dedicated branch



Working with

Development configuration

- **Scenario:** enable `devel` module only in development environment.
- **Problem:** exporting and importing uses the whole set of configuration.
- **Solution:** `drush cex --skip-modules=devel`
  `drush cim --skip-modules=devel`
  Don't forget to `gitignore` the devel config

Working with

Default content

- **Scenario:** Deploying configuration that depends on content, i.e. blocks or taxonomy terms.
- **Problem:** Content is not exported along with configuration.
- **Solution 1:** handle core event `\Drupal\Core\Config\Importer\MissingContentEver`
- **Solution 2:** Use Migrate module, now in core.
- **Solution 3:** Use default_content (https://www.drupal.org/project/default_content) contrib module.

Working with

Content updates

```
/**
 * Executes an update which is intended to update data, like entities.
 *
 * These updates are executed after all hook_update_N() implementations.
 * At this stage Drupal is already fully repaired so you can use any API
 * as you wish.
 */
function hook_post_update_NAME(&$sandbox) {}
```

Related contrib projects

config_installer: install site from existing config.
config_readonly: locks any configuration changes via UI.
config_update: report changes between original and active config of a module.
config_devel: module helps with developing configuration.
config_tools: automatically commit config changes to git.
features: bundle config for re-use on different site, ideal for distributions.

Thanks to the sponsors