Dominique De Cooman
Saturday, October 23, 2010 - 20:14
best-practices [1]
staging [2]

I've ran into this one quit a few times. We have live production environment, a staging environment, an integration environment and a local development environment. This classic software development chain goes upstream from production to dev and downstream from dev to production.

## Upstream

The upstream process requires usualy just transfer of newly created content, users and user generated settings which is mostly done by transfering database dumps. However on some projects the database is to big to transfer in one piece. When you have 3.5 million nodes in the database you dont want download the entire thing.

Solutions
When you only want the user generated settings from a production environment (example http://drupal.org/project/homebox [3] settings) a possible solution is to use the http://drupal.org/project/backup_migrate [4] module . It allows you to select which tables you want to export. Doing so you only export the tables containing your settings. Although this requires more advanced knowledge on how drupal works (because you have to know which are the correct tables to export) it is an easy repeatable way because you can save the table selection. You can automate this module using cron and it has some drush commands at your disposal so this is a fast tool to update the upstream sites.
If you want only newly created content, drupal settings or views moved upstream to dev use the http://drupal.org/project/deploy [5] module. This video explains all http://www.youtube.com/watch?v=7PjwT0HWHxw [6]

## Downstream

The downstream process that pushes new code and settings down the chain is a bit of problem in drupal.
You have the brutal way by transferring database dumps. Which wont be a great plan when your site constantly gets updated. A variation of this would be like explained above and transfer only tables containing settings using backup and migrate. But you ll run into trouble as soon as you have to start installing modules which act on content. A simple example would be a new field on a content type. This alters your content type table. Also everything that is database transfer related has poor rollback capabilities and poor change logging. Doing database transfers togetter with manualy updating stuff is a caveman update strategy.

To solve this problem more elegant lets install a system that makes use of three components.

### Features

This module http://drupal.org/project/features [7] has the power to export datastructures which live in the database into code. Which is great because you can track changes in your version control tool and revert changes. This is truly the savior module. BUT it doesnt support all datastructures. It supports menu's, permissions, content types, views, displays, rules, ... (check the project page for more)
Use feature for the datastructures it supports. Hopefully all datastructure in drupal get supported and this will be the only thing needed to solve the staging problem.
How does it work? Install it, create one or more features, export them and when you have updates in your datastructures it will show them. Use drush fu [name-feature]  to update them, commit them to your repository. When the svn up on downstream envirronments happens your site will use the new datastructures

### Deploy

You know this one from above use it if you have some content staged. A rollback system is not in place so to rollback delete the content it has created manualy. The reason why I dont use it to transfer settings or views or other things than content is because there is no rollback.

### Hook update

Use http://api.drupal.org/api/function/hook_update_N/6 [8] to do all changes that cannot be done by the former two. An example would be a custom or a contrib datastructure that lives in the database and that cant be exported by features or deploy. If you want a rollback you ll have to write it yourself using yet another hook_update which hopefuly you wont need to do that to often :)

We also use hook update to do updates which are envirronment specific. For example you dont want devel enabled on production.
In settings.php set an envirronment variable. Since settings.php is different on each environment set a setting so you can have your drupal know on which environment it is. For example on production in settings.php.

```
<?php
$conf['environment'] = 'production';
?>
```

Now in hook_update you do

```
<?php
function module_update_6001() {
  if (variable_get('environment', 'dev') == 'production') {
    module_disable(array('devel'));
  }

  return array();
}
?>
```

### Conclusion

The key is to think everytime you change the database and grow the habitude to get those changes into code. Using above strategy will require more time especialy if more hook updates are required. It will consume even more time if you need rollbacks in place. But it gives you more control.

So we come down to a tradeof between control and development time. This should be made clear to everyone involved from client, project manager, developer, maintainer that installing this workflow in your project consumes time.

### Future

The real solution to this problem lies in the lack of support in core. A core system registering all datastructures and a stage content marking should come in place. The system should be able to connect all environments and show the status of the content and structures. According to that releases could be planned, executed, rolled back, logged, ... And then offcourse all contrib modules contributing structures should integrate with this core system :)

For now that systems seems not become reality soon but I think features comes close for settings and datastructures and deploy comes close for content. I think we ll row with hook update to close the gaps for a couple of years.

**Source URL:** https://dominiquedecooman.com/blog/drupal-staging-problem

**Links:**
[1] https://dominiquedecooman.com/blog-topics/best-practices
[2] https://dominiquedecooman.com/blog-topics/staging-0
[3] http://drupal.org/project/homebox

[4] http://drupal.org/project/backup_migrate
[5] http://drupal.org/project/deploy
[6] http://www.youtube.com/watch?v=7PjwT0HWHxw
[7] http://drupal.org/project/features
[8] http://api.drupal.org/api/function/hook_update_N/6