Running a CakePHP App in Different Operating Environments

Brian Porter

Project Lead & Web Developer at Loadsys

Overview

- What do I mean by "operating environments"?
- How can an app's needs change based on the environment?
- Various methods for configuring Cake per-environment.
- Properties of an ideal env-aware config system.
- Examples.

What are "operating environments"?

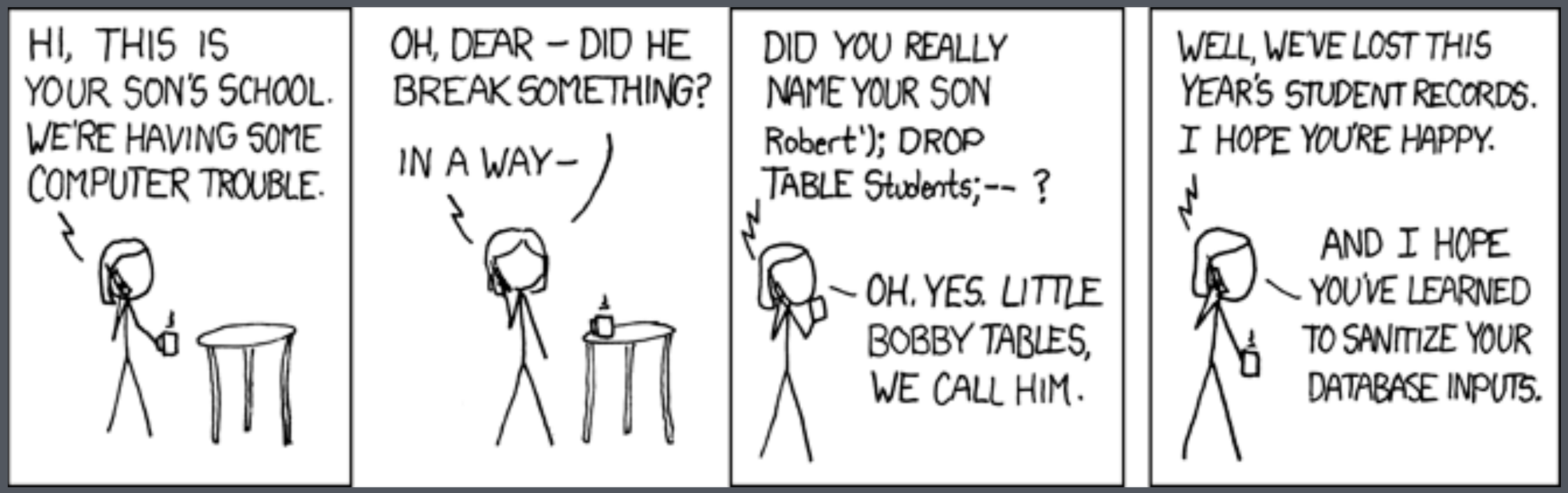
- Developer's workstation or Vagrant virtual machine.
- Continuous integration or testing server. (Travis, Jenkins)
- Quality assurance site.
- Staging site used for stakeholder review.
- Production instances.

How are your app's needs different in each environment?

- Database connections.
- SSL availability or enforcement.
- Email delivery configuration.
- External API integration (Paypal, Stripe, Mailchimp, Google Analytics).

The most basic example

Who hasn't needed to set different database connection values in development and production?



config/app.php in development

```
return [
    'Datasources' => [
        'default' => [
            // *snip*
            'host' => 'localhost',
            'database' => 'my_app',
            'username' => 'my_app',
            'password' => 'secret',
        ],
    ],
];
```

config/app.php in production

```
return [
    'Datasources' => [
        'default' => [
            // *snip*
            'host' => 'mine.us-east-1.rds.amazonaws.com',
            'database' => 'production_app',
            'username' => 'production_app',
            'password' => 'rw8d&FI.?:@2',
        ],
    ],
];
```

How can we handle this difference?

- There are many approaches.
- They all have different complexity and tradeoffs.
- Let's start with some common ones...

❌ Not using Configure at all ❌

```
// src/Template/Layout/default.ctp
<?php
if ($_SERVER['SERVER_NAME'] === 'www.site.com') {
    echo 'This is production';
} elseif ($_SERVER['SERVER_NAME'] === 'stage.site.com') {
    echo 'This is staging';
} else {
    echo 'This is development';
}
?>
```

❌ Not using Configure at all ❌

```php
<?php
if ($_SERVER['SERVER_NAME'] === 'www.site.com') {
    Configure::write('Datasources.default', [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => 'prod-db-server.amazonaws.com',
        //'port' => 'nonstandard_port_number',
        'username' => 'rdsuser',
        'password' => 'O*&tITbVfr^%CU',
        'database' => 'productionsite',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'cacheMetadata' => true,
        'quoteIdentifiers' => false,
    ];
} elseif ($_SERVER['SERVER_NAME'] === 'stage.site.com') {
    Configure::write('Datasources.default', [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => 'staging-db-server',
        'port' => '3307',
        'username' => 'common',
        'password' => 'password',
        'database' => 'staging',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'cacheMetadata' => true,
        'quoteIdentifiers' => false,
    ];
} else {
    Configure::write('Datasources.default', [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => 'localhost',
        'username' => 'root',
        'password' => 'root',
        'database' => 'default',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'cacheMetadata' => true,
        'quoteIdentifiers' => false,
    ];
}
?>
```

What's wrong with that?

- — Unnecessarily verbose.
- — Code must change if domain names change.
- — Hardcoded to 3 specific environments.
- — The env flag being checked is duplicated in the code.

❌ Not storing configs in the repo at all ❌

```
$ cat .gitignore
tmp/
vendor/
config/app.php
#...
```

What's wrong with excluding configs from the repo?

- ➕ It's straightforward(?)
- ➕ No sensitive info in the repo.
- ➖ No backups or history.
- ➖ Troubleshooting is harder.
- ➖ Still not DRY.
- ➖ Still fragile.

❌ Store individual config files in the repo ❌

```
# Access the server
$ ssh deploy@production-server.com
$ ls config/app*
app.prod.php
app.staging.php
app.qa.php
app.dev.php

# "Deploy" new code
$ git pull origin master

# Copy updated config into place.
$ cp config/app.staging.php config/app.php
```

Oops!

```
$ ssh deploy@production-server.com
#                  ^
#                  production server

$ cp config/app.staging.php config/app.php
#                  ^
#                  copied the wrong config!
```

What's wrong with copying files?

- ➕ Easy to understand.
- ➕ Configs stored in repo.
- ➖ Not DRY.
- ➖ Fragile for devs *and* sysadmins.
- ➖ Potential security risk.

Concepts

What are the properties of the *ideal* system for handling custom configurations per environment?

A single "switch" **from** the environment defines it

Example using Apache's `SetEnv`:

```
# my_apache_vhost.conf
<VirtualHost *:80>
    ServerName stagingsite.com
    SetEnv APP_ENV staging
</VirtualHost>
```

and a command line env var:

```
# ~/.profile or ~/.bash_profile
# Make sure env is set for Cake Shells.
export APP_ENV=staging
```

The environment switch should be "artificial"

The environment flag used must be maleable to adapt to changing circumstances.

Define your own "independently controllable" environment switch to maintain control of your own destiny.

---

image John Holm (CC BY 2.0)
flickr.com/photos/29385617@N00/2366471410

All non-sensitive configs are tracked

All environment configs (except those deemed "sensitive") must be tracked in the repo with the code that utilizes them.

Developers must have access to add or change config both where they are defined *and* where they are used.

Config changes must not require more than one role (dev + sysadmin) or be done in more than one place (repo + servers).

Convention is favored over configuration

The app must first be designed to function regardless of the environment.

In other words: Whenever possible, build so that it doesn't matter what environment you are running in.

Adding environment-specific settings must be done only when there is no other choice.

Env-specific settings are checked and loaded **once**

The app must check *"the thing that defines the environment"* (environment var, Apache SetEnv, hostname, etc.) **exactly once**.

The app must perform logic for loading configs for that environment **at that point**.

If the environment detection needs to change in the future, there is only one place in the code to change.

Production is the default case

The *most important environment* your app runs in must be the master of all things.

The app must function correctly in that environment even without an explicit environment set.

Protect the mission-critical environment from being effected by a missing or invalid environment setup.

Only necessary keys are overridden (and minimally so)

Leverage the production valus as "defaults" as much as possible.

Keep the other configs DRY by overriding **only** what is different in each environment.

Reduce the risk of "missing" a config that is defined in multiple places to the minimum possible.

Support untracked overrides for testing and security

The app must allow for a developer to change a config while testing a feature locally without committing the "test" settings.

The app must support situations where sensitive configs must not be stored in the repo.

*Provide a fallback for situations where tracking **all** config files is a hinderance to get the best of both worlds.*

App is ignorant of its environment(s)

The app itself must never be "aware" of the different environments.

It should always be presented with a **single** set of configs to use.

Provide the same collection of config keys to all environments. Change only their values per-environment.

All environments read the same keys

The app must not wrap retrieved config values in conditional statements. (`if/else`)

Do **not** do different things depending on a config value:
Do the same thing using the different values.

*Let the config bootstrapping process **be** the conditional statement by relocating values from the app into the configs.*

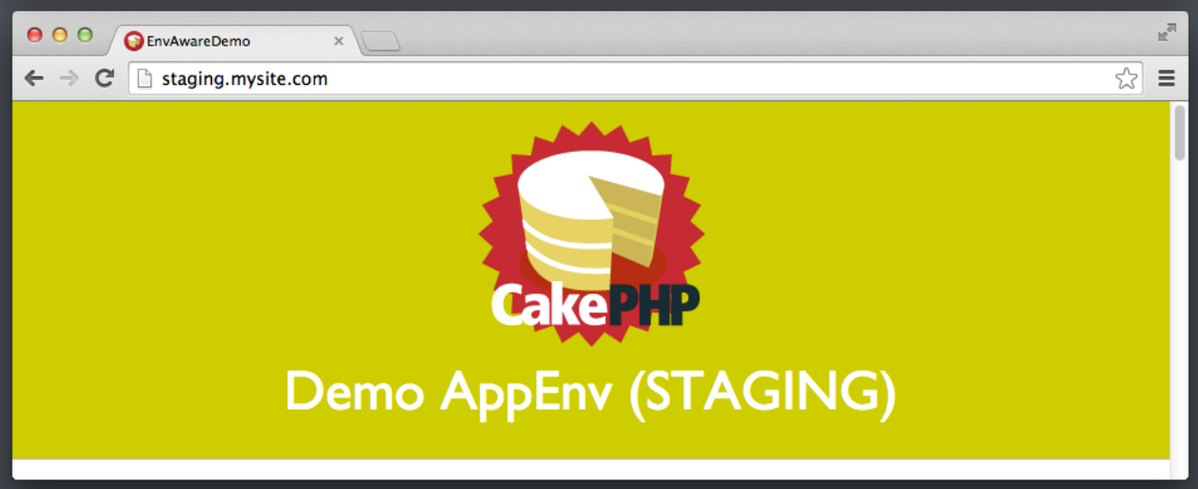
Remember this example?

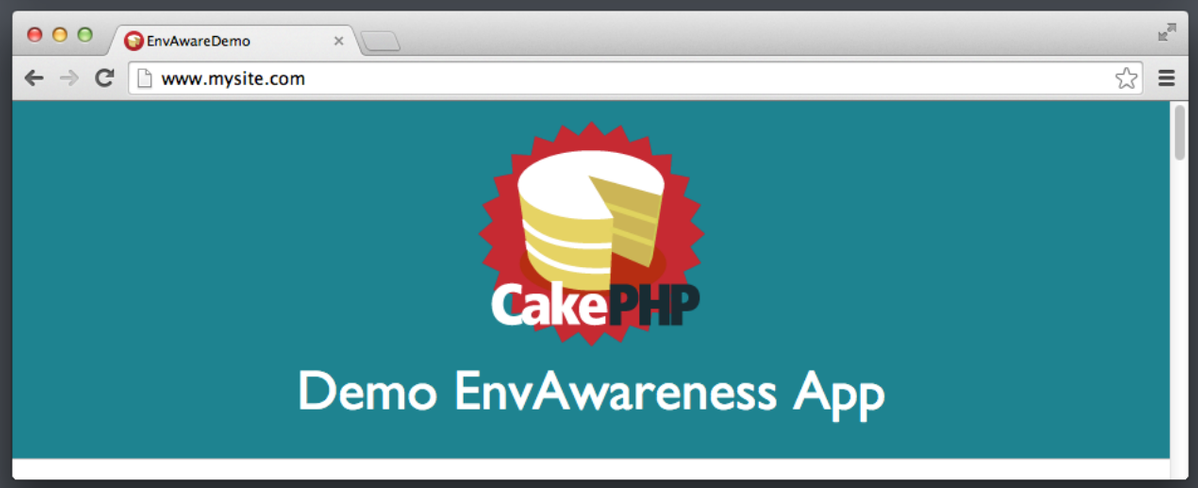
```php
// src/Template/Layout/default.ctp
<?php
if ($_SERVER['SERVER_NAME'] === 'www.site.com') {
    echo 'This is production';
} elseif ($_SERVER['SERVER_NAME'] === 'stage.site.com') {
    echo 'This is staging';
} else {
    echo 'This is development';
}
?>
```

How about this instead?

```
// src/Template/Layout/default.ctp
<?php
    echo Configure::read('Env.Message');
?>
```

How do we make that happen?





Cake's `Configure` class

Cake stores runtime configuration using the `Configure` class.

```
// Define a key in config/app.php:
Configure::write('MySection.MyKey', 'Cake is awesome');
```

```
// Recall the value anywhere else:
echo Configure::read('MySection.MyKey');
```

Cake's `Configure` class

- Has an excellent API for setting, overriding and accessing values.
- Is accessible nearly everywhere in a Cake app.
- With Cake 3, it is also better unified.
  (DB, Email, Cache and App settings all in one place.)

This makes it an ideal mechanism for storing environment-specific settings.

Leveraging the Configure class

- By default, Cake already loads all settings defined in `config/app.php`.
- This is now your "mission-critical" (default) config, typically production.
- We will define per-key override values in additional environment-specific config files.
- Which additional file is loaded will depend on the value of your environment flag.

Environment reminder

Apache `SetEnv`:

```
# my_apache_vhost.conf
<VirtualHost *:80>
    ServerName stagingsite.com
    SetEnv APP_ENV staging
</VirtualHost>
```

and a command line environment variable:

```
# ~/.profile or ~/.bash_profile
export APP_ENV=staging
```

Stock `config/bootstrap.php`

Cake 3 ships with the following code:

```
// config/bootstrap.php
try {
    Configure::config('default', new PhpConfig());
    Configure::load('app', 'default', false);
} catch (\Exception $e) {
    // Failure to load the mission-critical
    // config is a fatal error.
    die($e->getMessage() . "\n");
}
```

Loading additional env-specific configs

```
// config/bootstrap.php

// After loading the stock config file,
// load the environment config file
// and the local config file (when present.)
try {
    $env = getenv('APP_ENV');
    Configure::load("app-{$env}", 'default');
    Configure::load('app-local', 'default');
} catch (\Exception $e) {
    // It is not an error if these files are missing.
}
```

config/app.php

```
return [
    'debug' => false,
    'Env' => [
        'FancyName' => 'Wonderful Application',
        'SignalColor' => '#ffffff', // White admin bg in production.
    ],
];
```

config/app-staging.php

Development, overrides only:

```
return [
    'debug' => true, // Turn debug on in development environments.
    'Env' => [
        // (Note that we don't change the [FancyName] key.)
        'SignalColor' => '#77cccc', // Red admin bg in staging.
    ],
];
```

What gets committed?

- `config/app.php`
- `config/app-*.php`
  - *except* `config/app-local.php`

- Add `/config/app-local.php` to `.gitignore`.

How well does this meet the ideal requirements?

✅ Uses a single switch from the environment.

✅ The environment switch is artificial.

```
<VirtualHost *:80>
    SetEnv APP_ENV staging
</VirtualHost>
```

How well does this meet the ideal requirements?

✅ All non-sensitive configs are tracked.

```
$ ls config/app*
app.php
app-staging.php
app-quality.php
app-vagrant.php
app-local.php
```

How well does this meet the ideal requirements?

✅ Env-specific settings are checked and loaded once.

✅ Production is the default case.

✅ Untracked overrides supported for testing/security.

```
try {
    $env = getenv('APP_ENV');
    Configure::load("app-{$env}", 'default');
    Configure::load('app-local', 'default');
} catch (\Exception $e) {}
```

How well does this meet the ideal requirements?

✅ App is ignorant of its environment(s).

✅ All environments read the same keys.

```php
<?php
    // We don't need to use `getenv()`
    // anywhere in our code.
    echo Configure::read('Env.Message');
?>
```

How well does this meet the ideal requirements?

✅ Convention is favored over configuration.

✅ Only necessary keys are overridden.

```php
// config/app-staging.php
return [
    'debug' => true,
    'Env' => [
        'SignalColor' => '#77cccc',
    ],
];
```

Example Project

github.com/beporter/CakePHP-EnvAwareness

- A demo Cake 3 app with vagrant.
- Includes loadsys/ConfigReadShell for command line access.
- Switch app background color based on env.

Other random points

- Works in 2.x via `Config/core.php`.
    - Requires a boilerplate `database.php` and `email.php` that load their configs from `Configure` instead of defining static class properties.
- @TODO: Examples to come in the demo repo.

Other random points

- Even works all the way back in 1.2/1.3.
  - **Mind `Configure::load()` in 1.x**: It overwrites entire keys instead of merging.
    - No examples *(get away from 1.x please)*, but you can ask me about it.
- Make sure your cron jobs execute with the correct environment set.

@TODO:

- Example: styleForEnv()-ish case
- Example: Passing an environment to Javascript (Ember) in default layout.

Questions?

Brian Porter
@beporter

(although you shouldn't follow me, you'll be disappointed.)

Project Lead and Web Developer
for Loadsys Web Strategies
loadsys.com

Slides, Sample Project
github.com/beporter/CakePHP-EnvAwareness