The Twelve-Factor App

III. Config

Store config in the environment

An app's *config* is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

- Resource handles to the database, Memcached, and other backing services
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Apps sometimes store config as constants in the code. This is a violation of twelve-factor, which requires **strict separation of config from code**. Config varies substantially across deploys, code does not.

A litmus test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.

Note that this definition of "config" does **not** include internal application config, such as `config/routes.rb` in Rails, or how code modules are connected in Spring. This type of config does not vary between deploys, and so is best done in the code.

Another approach to config is the use of config files which are not checked into revision control, such as `config/database.yml` in Rails. This is a huge improvement over using constants which are checked into the code repo, but still has weaknesses: it's easy to mistakenly check in a config file to the repo; there is a tendency for config files to be scattered about in different places and different formats, making it hard to see and manage all the config in one place. Further, these formats tend to be language- or framework-specific.

The twelve-factor app stores config in *environment variables* (often shortened to *env vars* or *env*). Env vars are easy to change between deploys without changing any code; unlike config files, there is little chance of them being checked into the code repo accidentally; and unlike custom config files, or other config mechanisms such as Java System Properties, they are a language- and OS-agnostic standard.

Another aspect of config management is grouping. Sometimes apps batch config into named groups (often called "environments") named after specific deploys, such as the `development`, `test`, and `production` environments in Rails. This method does not scale cleanly: as more deploys of the app are created, new environment names are necessary, such as `staging` or `qa`. As the project grows further, developers may add their own special environments like `joes-staging`, resulting in a combinatorial explosion of config which makes managing deploys of the app very brittle.

In a twelve-factor app, env vars are granular controls, each fully orthogonal to other env vars. They are never grouped together as "environments", but instead are independently managed for each deploy. This is a model that scales up smoothly as the app naturally expands into more deploys over its lifetime.

Italiano (it) | 简体中文 (zh_cn) | 한국어 (ko) | 日本語 (ja) | Polski (pl) | Русский (ru) | English (en) | Turkish (tr) | Français (fr) | Brazilian Portuguese (pt_br) | Українська (uk) | Deutsch (de) | Español (es)
« Previous
Next »
Italiano (it) | 简体中文 (zh_cn) | 한국어 (ko) | 日本語 (ja) | Polski (pl) | Русский (ru) | English (en) | Turkish (tr) | Français (fr) | Brazilian Portuguese (pt_br) | Українська (uk) | Deutsch (de) | Español (es)
Written by Adam Wiggins
Last updated 2017
Sourcecode
Download ePub Book

Privacy Policy