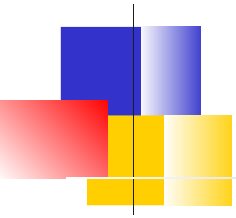


12 Factor App

Advise for Projects

Twelve-Factor App

Guidelines & advise for creating software-as-a-service applications.

Goals: Web apps and web services that are

- scalable
- portable
- maintainable, avoid erosion
- enable distributed collaboration

Developed by the Heroku team, based on their experience with "hundreds" of app.

Erosion vs Long-lived Software

Most software can't be maintained over time.

- depends on other software that is no longer maintained or API has changed
- no one knows how to update it
- features no longer meet customer requirements

Software Maintenance costs more than development.

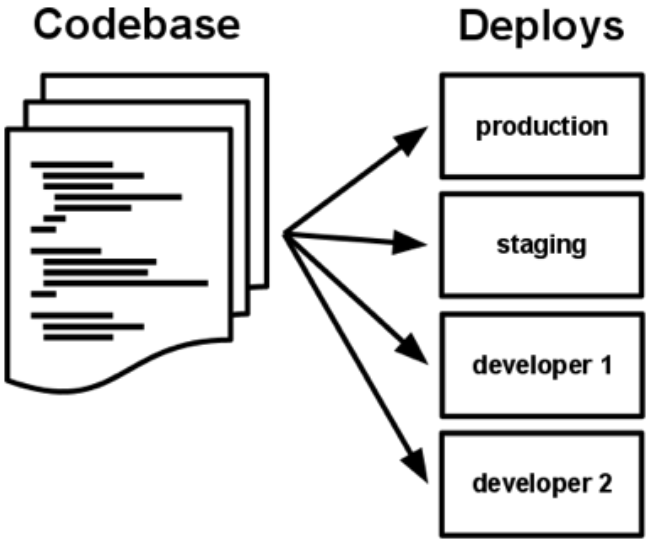
(*This could be a useful elective course: Develop and apply a concrete guide for long-lived senior projects.*)

1. One Code base, many deploys

Code should be maintained in Version Control (VCS)

"One app one code base"

- No separate versions for "local" and "cloud"
- Don't share code between apps (import as a dependency using dependency manager)
- Don't require 2 repos to install (use git submodule or dependency manager)



2. Explicitly Declare Dependencies

1. Use a dependency manager: pip for Python; npm for Javascript; Maven, Gradle, or Ivy for Java.
2. Declare all dependencies: requirements.txt, package.json
3. Don't rely on presence of any system-wide packages

Example:

- App uses the Python "`Requests`" package.
"Requests" may not be installed on another person's machine -- declare it as a dependency.
- `browser = webdriver.Safari()`
Only works on MacOS.

... and Isolate Dependencies

For Python, use "virtualenv".

When deploy to a virtual host (PaaS server), isolation is provided by the virtual platform.

Example:

```
python -m venv  env          # create "env"
source env/bin/activate      # on Linux & MacOS
(venv)$ pip install -r requirements.txt
```

Downside:

Requires more disk space and downloads

3. Store Config in the environment

"Config" is everything that is likely to vary between deployments (staging, production, local environment)

- database handles: DATABASE_URL = ...
- credentials for other services your app uses
- sensitive values (Django SECRET_KEY)
- anything else likely to change

OK to use a **configuration file** instead of environment... provided there is a way to specify a different configuration file w/o changing the code.

Java Properties File

```
# Don't commit this file to Git!
jdbc.url =
  jdbc:mysql://cloud.google.com/xxxx
jdbc.user = pollsadmin
jdbc.password = secret
```

Django Example

In settings.py:

```
from decouple import config
import dj_database_url as db_url

SECRET_KEY = config('SECRET_KEY', default="no-secret")

DATABASES = {
    'default': config('DATABASE_URL', cast=db_url.parse)
}
```

`config()` will set 'SECRET_KEY' and 'DATABASE_URL' using either environment variables or values in a file named `.env`. For example:

```
# this is a .env file. Quotes are not needed.
SECRET_KEY = wjtc3c@k5m!3^0m3dq=e^jff_t%q*blm
DATABASE_URL=postgres://admin:secret@localhost:5432/polls
```

8. Export Services via Port Binding

Some apps run inside a container (web app server).

- PHP apps run inside Apache httpd or nginx

- Java apps run in Tomcat or Jetty.

A 12-Factor App is self-contained.

- It provides its own web server:
  - Gunicorn for Python
  - Jetty for Java

Note: Item "4. Backing Services" provides guidance for interfacing with other services your app requires.

10. Keep Dev, Staging, and Production as similar as possible

Someone should be able to checkout, install, and run your "dev" or "current" version using nearly the same instructions as the "production" version in the cloud.

Differences are:

- different set of "Config" values

- initialize database schema and initial database data

Note: Heroku recommends using same database in "dev" environment as in production. This is contrary to other advise: "*View database as a resource with a standard API*".

11. Treat Logs as Event Streams

Logs are a stream of *events* that occur in your app.

Use a Logging API to log events & activity consistently.

12-Factor App Advise:

App should not attempt to manage log files.

Instead, each process writes log events to stdout.

In development, log messages will appear on console.

In production, a log router such as Logplex or Fluentd will handle routing of log messages.

Reference

https://12factor.net/

Software Erosion

https://blog.heroku.com/the_new_heroku_4_erosion_resistance_explicit_contracts

Interesting article with advise.

Some advise is specific to Javascript web apps. Java has its own solution to some of these problems.

True for CPSKE Senior projects: most of them cannot be used later... even *great ones* like "Live Scrum".