12 Factor App

Advise for Web Applications

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- depends on other software that is no longer maintained or API has changed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.

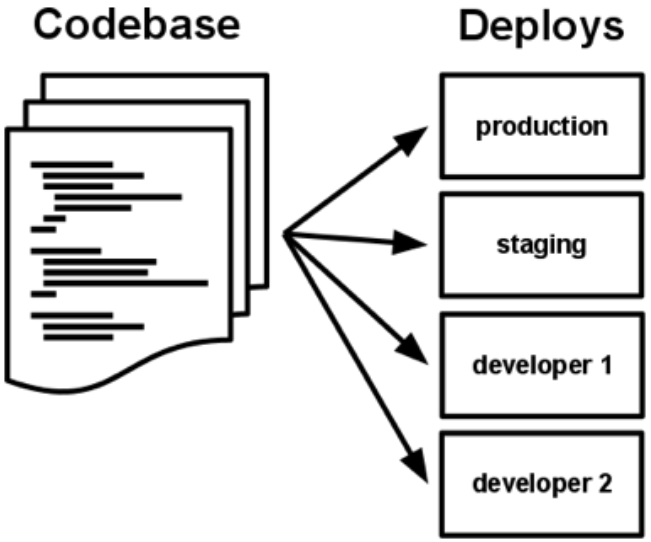
(*This could be a useful elective:*
Develop and apply a concrete guide
for long-lived senior projects.)

1. One Code base, many deploys

Code should be maintained in a 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 a 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

Examples:

- 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.

Downside:

Requires more disk space and downloads

3. Store Config in the environment

"Config" is everything that is likely to vary between deployments

- database handles: DATABASE_URL = ...
- credentials (keys, tokens) for services your app uses
- sensitive values (Django SECRET_KEY)
- location of static files

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.

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 get named values from either:

1. environment variables
2. command line values
3. values in a file named **`.env`**.

```
# 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
```

Java Properties File

```
jdbc.url =
  jdbc:mysql://cloud.google.com/xxxx
jdbc.user = pollsadmin
jdbc.password = secret
```

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.

Application includes its own web server:

- Gunicorn or uWSGI for Python (WSGI)
- May also use nginx or Apache httpd
- 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

(*one code base many deployments*)

Differences are:

- different set of "Config" values
- initialize database schema and initial database data

Note: Heroku recommends using same database in "dev" and "production".

This is contrary to other advise:
"*View database as a resource with a standard API*".

11. Treat Logs as Event Streams

Use a Logging API to log events & activity consistently.

12-Factor App Advise:

App should not attempt to manage log files.

Logging API writes messages to `stdout`.

In development, log messages will appear on console.

In production, a log router will handle routing of log messages (Logplex and Loggly are examples).

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".