The Twelve Factor App

Goals

We want to build apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering maximum portability between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling continuous deployment for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

I. Codebase

One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

- A twelve-factor app never relies on implicit existence of system-wide packages
- Simplifies setup for developers new to the app
- Do not rely on the implicit existence of any system tools

III. Config

Store config in the environment

- Config is everything that is likely to vary between deploys
- Storing configs in constants is a violation of twelve-factor, which requires strict separation of config from code

IV. Backing services

Treat backing services as attached resources

- The code for a twelve-factor app makes no distinction between local and third party services
- Resources can be attached to and detached from deploys at will

V. Build, release, run
Strictly separate build and run stages

A codebase is transformed into a (non-development) deploy through three stages:

- The build stage: transform which converts a code repo into an executable bundle known as a build
- The release stage: takes the build produced by the build stage and combines it with the deploy's current config
- The run stage (also known as "runtime"): run the app in the execution environment

VI. Processes

Execute the app as one or more stateless processes

- Twelve-factor processes are stateless and share-nothing
- 'Sticky Sessions' are a violation

VII. Port binding

Export services via port binding

- The twelve-factor app is completely self-contained
- The web app exports HTTP as a service by binding to a port, and listening to requests coming in on that port
- Applies to other protocols as well

VIII. Concurrency

Scale out via the process model

- Processes are a first class citizen, and the unit of concurrency
- Due to this, adding more concurrency is a simple and reliable operation

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

- The twelve-factor app's processes are disposable, meaning they can be started or stopped at a moment's notice
- Processes should strive to minimize startup time
- Processes shut down gracefully when they receive a SIGTERM signal
- For a worker process, graceful shutdown is achieved by returning the current job to the work queue
- Processes should also be robust against sudden death

X. Dev/prod parity

Keep development, staging, and production as similar as possible

- The twelve-factor app is designed for continuous deployment by keeping the gap between development and production small
  - The time gap
  - The personnel gap
  - The tools gap
- The twelve-factor developer resists the urge to use different backing services between development and production

XI. Logs

Treat logs as event streams

- A twelve-factor app never concerns itself with routing or storage of its output stream
- Instead, each running process writes its event stream, unbuffered, to stdout

XII. Admin processes

Run admin/management tasks as one-off processes

One-off admin processes should be run in an identical environment as the regular long-running processes of the app