

"In the modern era, software is commonly delivered as a service: called web apps, or software-as-a-service. The twelve-factor app is a methodology for building software-as-a-service apps. . . ." <https://12factor.net/>

# I. Codebase

## One codebase tracked in revision control, many deploys

Tiki Wiki CMS Groupware ("Tiki") is tracked in two version control systems: Git and SVN (deprecated), currently Git is the default.

Tiki uses the same codebase with different releases. A major branch is done only once per major version (4.x, 5.x, 6.x, etc.) about 4-6 weeks before the planned official release of x.0. For minor versions, the work is done in the current branch.

# II. Dependencies

## Explicitly declare and isolate dependencies

Tiki uses [Composer](#) to manage the dependencies. All dependencies are declared in [vendor\\_bundled/composer.json](#) and to extend some Tiki features it is possible to install via [Packages](#) using the UI, these additional dependencies are also managed by composer.

# III. Config

## Store config in the environment

Tiki does not store config as constants in the code and uses different environment files.

The `db/local.php` file stores some config settings but, it's also possible to extend configurations between environments, see [System Configuration](#) for more information.

# IV. Backing services

**Treat backing services as attached resources**

Tiki makes no distinction between local and third-party services. Tiki is able to swap out, for instance, [SMTP](#) service without code changes and local MySQL database by a third party using configuration files.

# V. Build, release, run

**Strictly separate build and run stages**

Tiki strictly separates build and run stages storing new releases as <number>.x, for example, 22.x.

# VI. Processes

**Execute the app as one or more stateless processes**

Tiki has data that needs to be persistent; however, that data is stored in a stateful backing service like database, Memcached or expiring sessions.

# VII. Port binding

**Export services via port binding**

Because Tiki uses PHP, probably the code is being executed using PHP-FPM which exposes a binding port to communicate with

## VIII. Concurrency

### **Scale out via the process model**

Tiki does not rely on daemonize processes or write PID files, in fact it uses system processes to handle the requests and background tasks.

## IX. Disposability

### **Maximize robustness with fast startup and graceful shutdown**

For a web application like Tiki, this factor is achieved automatically, since PHP-FPM handles system signals like SIGTERM or QUIT out-of-the-box closing existing connections and refusing newer ones when the process stops.

## X. Dev/prod parity

### **Keep development, staging, and production as similar as possible**

Tiki uses continuous deployment, making the code being pushed and the production code almost identical and available to use by other developers. See: [Sync Dev-Prod Servers](#)

# XI. Logs

## **Treat logs as event streams**

Tiki does not have a single log file that can be used as a stream, but depending on some tasks log files are created (like index rebuild) or other operations/actions are logged in the database as action logs, used to track user activity on the basis of a single user or multiple users, groups or categories.

# XII. Admin processes

## **Run admin/management tasks as one-off processes**

Tiki offers administrative and maintenance tasks (see [console.php](#)), such as:

- Running database migrations
- Running a console command
- Running one-time scripts

The admin processes run in an identical environment as the regular long-running processes of the app, and they run against a release, using the same codebase and config as any process run against that release. The admin code is included in the version control system.