12 Factor App

Codecinella - September 2022

What is 12 Factor App?

A methodology for building software-as-a-service apps that

- **Minimizes time and cost for new developers joining the project** *by using declarative formats for setup automation*
- **Offers maximum portability between execution environments** *by having a clean contract with the underlying operating system*
- **Obviates the need for servers and systems administration** *by being suitable for deployment on modern cloud platforms*
- **Enables continuous deployment for maximum agility** *by minimizing divergence between development and production*
- **Can scale up without significant changes to tooling, architecture, or development practices.**

1. Codebase

One codebase tracked in revision control, many deploys

Codebase

There is always a one-to-one correlation between the codebase and the app

- If there are multiple codebases, it's not an app – it's a distributed system. Each component in a distributed system is an app, and each can individually comply with twelve-factor.
- Multiple apps sharing the same code is a violation of twelve-factor. The solution here is to factor shared code into libraries which can be included through the dependency manager.
- There is only one codebase per app, but there will be many deploys of the app. A deploy is a running instance of the app. This is typically a production site, and one or more staging sites.

2. Dependencies

Explicitly declare and isolate dependencies

Dependencies

It declares all dependencies, completely and exactly, via a dependency declaration manifest.

- The full and explicit dependency specification is applied uniformly to both production and development.
- One benefit of explicit dependency declaration is that it simplifies setup for developers new to the app
- Twelve-factor apps also do not rely on the implicit existence of any system tools.

3. Config

Store config in the environment

Config

The twelve-factor app stores config in environment variables (often shortened to env vars or env)

- An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc).
- Config varies substantially across deploys, code does not.
- Env vars are easy to change between deploys without changing any code; they are a language- and OS-agnostic standard.

4. Backing services

Treat backing services as attached resources

Backing Services

A backing service is any service the app consumes over the network as part of its normal operation.

- The code for a twelve-factor app makes no distinction between local and third party services.
- To the app, both are attached resources, accessed via a URL or other locator/credentials stored in the config.
- The twelve-factor app treats these databases as attached resources, which indicates their loose coupling to the deploy they are attached to.
- Resources can be attached to and detached from deploys at will.

5. Build, release, run

Strictly separate build and run stages

Build, release, run

The twelve-factor app uses strict separation between the build, release, and run stages

- The build stage is a 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”) runs the app in the execution environment, by launching some set of the app’s processes against a selected release.

6. Processes

Execute the app as one or more stateless processes

Processes

The app is executed in the execution environment as one or more processes.

- Twelve-factor processes are stateless and share-nothing.
- Any data that needs to persist must be stored in a stateful backing service, typically a database.
- Sticky sessions are a violation of twelve-factor and should never be used or relied upon.

7. Port binding

Export services via port binding

Port binding

The web app exports HTTP as a service by binding to a port, and listening to requests coming in on that port.

- Does not rely on runtime injection of a webserver into the execution environment to create a web-facing service.
- Nearly any kind of server software can be run via a process binding to a port and awaiting incoming requests.
- The port-binding approach means that one app can become the backing service for another app

8. Concurrency

Scale out via the process model

Concurrency

In the twelve-factor app, processes are a first class citizen.

- Using this model, the developer can architect their app to handle diverse workloads by assigning each type of work to a process type.
- For example, HTTP requests may be handled by a web process, and long-running background tasks handled by a worker process.
- The share-nothing, horizontally partitionable nature of twelve-factor app processes means that adding more concurrency is a simple and reliable operation.
- Twelve-factor app processes should never daemonize or write PID files.

9. Disposability

Maximize robustness with fast startup and graceful shutdown

Disposability

The twelve-factor app's processes are disposable, meaning they can be started or stopped at a moment's notice

- This facilitates fast elastic scaling, rapid deployment of code or config changes, and robustness of production deploys.
- Processes should strive to minimize startup time.
- Processes shut down gracefully when they receive a SIGTERM signal from the process manager.
- Processes should also be robust against sudden death, in the case of a failure in the underlying hardware.

10. Dev/prod parity

Keep development, staging, and production as similar as possible

Dev/prod parity

The twelve-factor app is designed for continuous deployment by keeping the gap between development and production small.

- Historically, there have been substantial gaps between development and production. These gaps manifest in three areas: time, personnel, tools
- Make the time gap small: a developer may write code and have it deployed hours or even just minutes later.
- Make the personnel gap small: developers who wrote code are closely involved in deploying it and watching its behavior in production.
- Make the tools gap small: keep development and production as similar as possible.

11. Logs

Treat logs as event streams

Logs

A twelve-factor app never concerns itself with routing or storage of its output stream.

- Logs provide visibility into the behavior of a running app.
- Logs are the stream of aggregated, time-ordered events collected from the output streams of all running processes and backing services.
- It should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout

12. Admin processes

Run admin/management tasks as one-off processes

Admin processes

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

- Developers will often wish to do one-off administrative or maintenance tasks for the app
- They run against a release, using the same codebase and config as any process run against that release.
- Admin code must ship with application code to avoid synchronization issues.
- Twelve-factor strongly favors languages which provide a REPL shell out of the box, and which make it easy to run one-off scripts.