Day 20 Task

The Twelve Factor App

# Introduction

In the modern era, software is commonly delivered as a service: called web apps, or software applications*.* The twelve-factor app is a methodology for building software-as-a-service 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.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

The twelve factors are: -

1. **Codebase**

* A twelve-factor app is always tracked in a version control system, such as [Git](http://git-scm.com/), [Mercurial](https://www.mercurial-scm.org/), or [Subversion](http://subversion.apache.org/).
* A codebase is any single repo, or any set of repos who share a root commit (in a decentralized revision control system like Git).
* There is always a one-to-one correlation between the codebase and the app i.e., there is only one codebase per app, but there will be many deploys of the app, there can be many branches of a repo.
* Deploys are running instance of an application, every developer has a copy of the app running in their local development environment, each of which also qualifies as a deploy.

1. **Dependencies**

* Most programming languages offer a packaging system for distributing support libraries, libraries installed through a packaging system can be installed system-wide or scoped into the directory containing the app”).
* **A twelve-factor app never relies on implicit existence of system-wide packages.**It declares all dependencies, completely and exactly, via a dependency declaration manifest.
* **A twelve-factor app** uses a dependency isolation tool during execution to ensure that no implicit dependencies “leak in” from the surrounding system.
* 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.

1. **Config**

* An app’s config is everything that is likely to vary between [deploys](https://12factor.net/codebase) can include resource handles to the database, credentials to external services or per-deploy values such as the canonical hostname for the deploy.
* Apps sometimes store config as constants in the code. This is a violation of twelve-factor, which requires **strict separation of config from code.**
* A test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.
* **The twelve-factor app stores config in**environmentvariables (often shortened to env vars or env). Env vars are easy to change between deploys without changing any code.

1. **Backing Services**

* A backing service is any service the app consumes over the network as part of its normal operation. Examples include datastores (such as [MySQL](http://dev.mysql.com/) ), messaging/queueing systems.
* Backing services like the database are traditionally managed by the same systems administrators as the app’s runtime deploy. In addition to these locally-managed services, the app may also have services provided and managed by third parties.
* It’s about treating that external services that your application depends on equally, regardless of whether you manage them, or whether another party manages them.
* To the app, both are attached resources, accessed via a URL or other locator/credentials stored in the [config](https://12factor.net/config).

1. **Build, Release, Run**

* A [codebase](https://12factor.net/codebase) is transformed into a (non-development) deploy through three stages Build (convert a code repo into an executable bundle), Release (combines build with the deploy’s current [config](https://12factor.net/config)) and Run (run the app in the execution environment).
* **The twelve-factor app uses strict separation between the build, release, and run stages.**

1. **Processes**

* The app is executed in the execution environment as one or more processes.
* **Twelve-factor processes are stateless and**[share-nothing](http://en.wikipedia.org/wiki/Shared_nothing_architecture)**.** Any data that needs to persist must be stored in a stateful [backing service](https://12factor.net/backing-services), typically a database.
* The twelve-factor app never assumes that anything cached in memory or on disk will be available on a future request or job, with many processes of each type running, chances are high that a future request will be served by a different process.
* A twelve-factor app prefers to do this compiling during the [build stage](https://12factor.net/build-release-run).

1. **Port Binding**

* Web apps are sometimes executed inside a webserver container.
* **The twelve-factor app is completely self-contained** and does not rely on runtime injection of a webserver into the execution environment to create a web-facing service.
* The web app **exports HTTP as a service by binding to a port,** and listening to requests coming in on that port.
* In a local development environment, the developer visits a service URL like http: //localhost:8080/ to access the service exported by their app. In deployment, a routing layer handles routing requests from a public-facing hostname to the port-bound web processes.

1. **Concurrency**

* Any computer program, once run, is represented by one or more processes. Web apps have taken a variety of process-execution forms. For example, PHP processes run as child processes of Apache.
* **In the twelve-factor app, processes are a first-class citizen.** Processes in the twelve-factor app take strong cues from [the UNIX process model for running service daemons](https://adam.herokuapp.com/past/2011/5/9/applying_the_unix_process_model_to_web_apps/).
* Using this model, the developer can architect their app to handle diverse workloads by assigning each type of work to a process type.

1. **Disposability**

* **The twelve-factor app’s**[processes](https://12factor.net/processes)**are**disposable**, meaning they can be started or stopped at a moment’s notice.**
* Processes should strive to **minimize start-up time.** Ideally, a process takes a few seconds from the time the launch command is executed until the process is up and ready to receive requests or jobs.
* Processes**shut down gracefully when they receive a**[SIGTERM](http://en.wikipedia.org/wiki/SIGTERM) signal from the process manager.
* Processes should also be **robust against sudden death,** in the case of a failure in the underlying hardware.

1. **Dev / Prod Parity**

* Your development environment should almost identical to a production one.
* **The twelve-factor app is designed for**[continuous deployment](http://avc.com/2011/02/continuous-deployment/)**by keeping the gap between development and production small.** Looking at the three gaps time gap, personnel gap and tools gap.
* **The twelve-factor developer resists the urge to use different backing services between development and production.**

1. **Logs**

* Logs provide visibility into the behaviour of a running app. In server-based environments they are commonly written to a file on disk (a “logfile”); but this is only an output format.
* **A twelve-factor app never concerns itself with routing or storage of its output stream.** It should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout.

1. **Admin Process**

* One-off admin processes should be run in an identical environment as the regular [long-running processes](https://12factor.net/processes) of the app.
* Admin code must ship with application code to avoid synchronization issues.
* The same [dependency isolation](https://12factor.net/dependencies) techniques should be used on all process types.
* Twelve-factor strongly favours languages which provide a REPL shell out of the box, and which make it easy to run one-off scripts.
* In a local deploy, developers invoke one-off admin processes by a direct shell command inside the app’s checkout directory. In a production deploy, developers can use ssh or other remote command execution mechanism provided by that deploy’s execution environment to run such a process.