UDC 004.02

PRINCIPLES OF DEVELOPING WEB APPLICATIONS USING THE "TWELVE-FACTOR APP" METHODOLOGY

KUPTSOV VYACHESLAV, OKSANA GOLUBEVA
Polotsk State University, Belarus

The article presents principles of developing software-as-a-service web applications using the "Twelve-Factor App" methodology. Solutions to problems that are often encountered in the development of modern applications are considered.

In the modern era, software is commonly delivered as a service: called web apps, or software-as-a-service (SaaS). Software as a service is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. SaaS is typically accessed by users using a thin client via a web browser.

The twelve-factor app is a methodology for building software-as-a-service apps that [1]:

1. use declarative formats for setup automation to minimize time and cost for new developers joining the project;
2. have a clean contract with the underlying operating system offering maximum portability between execution environments;
3. are suitable for deployment on modern cloud platforms obviating the need for servers and systems administration;
4. minimize divergence between development and production enabling continuous deployment for maximum agility;
5. can scale up without significant changes to tooling, architecture, or development practices;
6. are written in any programming language, and use any combination of backing services (database, queue, memory cache, etc.).

These factors are written by software engineers that have been directly involved in the development and deployment of hundreds of apps. They represent general conceptual solutions to problems that are often encountered in the development of modern applications.

Codebase. 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. Additionally, every developer has a copy of the app running in their local development environment, each of which also qualifies as a deploy. The codebase is the same across all deploys, although different versions may be active in each deploy. For example, a developer has some commits not yet deployed to staging; staging has some commits not yet deployed to production. But they all share the same codebase, thus making them identifiable as different deploys of the same app.

Dependencies. 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. Furthermore, it uses a dependency isolation tool during execution to ensure that no implicit dependencies "leak in" from the surrounding system. The full and explicit dependency specification is applied uniformly to both production and development.

Config. An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

1. Resource handles to the database, Memcached, and other backing services;
2. Credentials to external services such as Amazon S3 or Twitter;
3. 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. Config varies substantially across deploys, code does not.

The twelve-factor app stores config in environment variables (often shortened to env vars or env). Env vars are easy to change between deploys without changing any code; unlike config files, there is little chance of them being checked into the code repo accidentally; and unlike custom config files, or other config mechanisms such as Java System Properties, they are a language- and OS-agnostic standard.

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 or CouchDB), messaging/queueing systems (such as RabbitMQ or Beanstalkd), SMTP services for outbound email (such as Postfix), and caching systems (such as Memcached). 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.

A deploy of the twelve-factor app should be able to swap out a local MySQL database with one managed by a third party (such as Amazon RDS) without any changes to the app's code. Likewise, a local SMTP server could be swapped with a third-party SMTP service (such as Postmark) without code changes. In both cases, only the resource handle in the config needs to change.

Build, release, run. The twelve-factor app uses strict separation between the build, release, and run stages. For example, it is impossible to make changes to the code at runtime, since there is no way to propagate those changes back to the build stage. A codebase is transformed into a (non-development) deploy through three stages:

1. The build stage is a transform which converts a code repo into an executable bundle known as a build. Using a version of the code at a commit specified by the deployment process, the build stage fetches vendors dependencies and compiles binaries and assets;
2. The release stage takes the build produced by the build stage and combines it with the deploy's current config. The resulting release contains both the build and the config and is ready for immediate execution in the execution environment;
3. 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.

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. Some web systems rely on "sticky sessions" – that is, caching user session data in memory of the app's process and expecting future requests from the same visitor to be routed to the same process. Sticky sessions are a violation of twelve-factor and should never be used or relied upon. Session state data is a good candidate for a datastore that offers time-expiration, such as Memcached or Redis [2].

Port binding. Web apps are sometimes executed inside a webserver container. For example, PHP apps might run as a module inside Apache HTTPD, or Java apps might run inside Tomcat. 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.

Concurrency. 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. 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.

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. 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. Short startup time provides more agility for the release process and scaling up; and it aids robustness, because the process manager can more easily move processes to new physical machines when warranted.

Development/production parity. Historically, there have been substantial gaps between development (a developer making live edits to a local deploy of the app) and production (a running deploy of the app accessed by end users). These gaps manifest in three areas [3]:

1. The time gap: A developer may work on code that takes days, weeks, or even months to go into production;
2. The personnel gap: Developers write code, ops engineers deploy it;
3. The tools gap: Developers may be using a stack like Nginx, SQLite, and OS X, while the production deploy uses Apache, MySQL, and Linux.

The twelve-factor app is designed for continuous deployment by keeping the gap between development and production small. Looking at the three gaps described above:

1. Make the time gap small: a developer may write code and have it deployed hours or even just minutes later;
2. Make the personnel gap small: developers who wrote code are closely involved in deploying it and watching its behavior in production;
3. Make the tools gap small: keep development and production as similar as possible.

Logs. Logs are the stream of aggregated, time-ordered events collected from the output streams of all running processes and backing services. Logs in their raw form are typically a text format with one event per line (though backtraces from exceptions may span multiple lines). Logs have no fixed beginning or end, but flow continuously as long as the app is operating. 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. During local development, the developer will view this stream in the foreground of their terminal to observe the app's behavior.

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

Summarizing the factors described above, we can conclude that, despite some difficulties in the technical implementation of all methods, this methodology avoids many problems in the development of a web application. However, it should be noted that following some aspects of this approach does not always bring results at the initial stage of development. Often, some factors manifest themselves only when the application is scaled.

REFERENCES

1. The Twelve-Factor App. [Electronic resource]. – Mode of access: https://12factor.net/. – Date of access: 02.02.2018.
2. Приложение двенадцати факторов – The Twelve-Factor App [Electronic resource] / Хабрахабр. – Mode of access: https://habrahabr.ru/post/258739/. – Date of access: 02.02.2018.
3. 12-Factor Apps in Plain English – ClearlyTech [Electronic resource]. – Mode of access: http://www.clearlytech.com/2014/01/04/12-factor-apps-plain-english/. – Date of access: 02.02.2018.