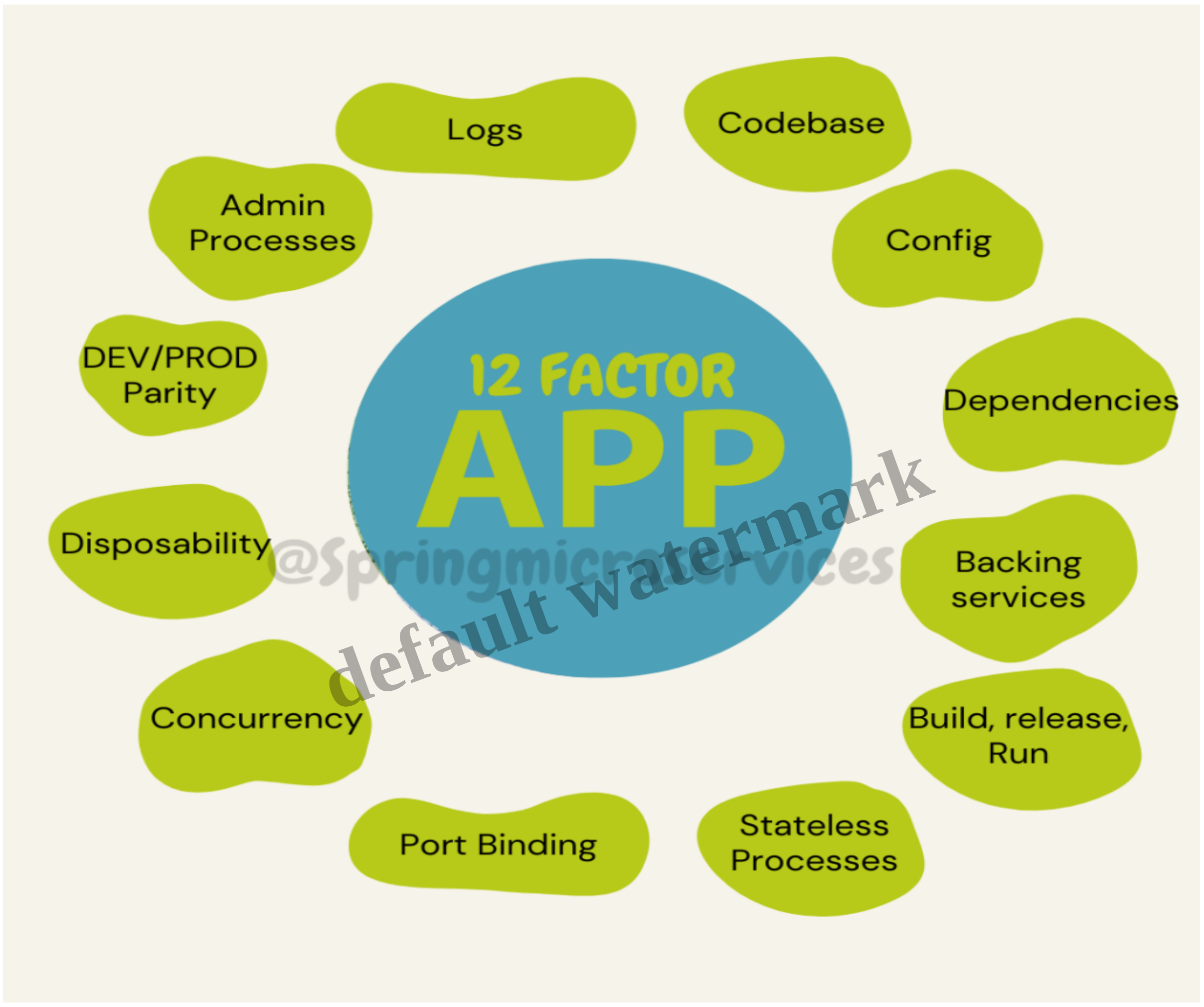
The 12-Factor App Principles

Description



Codebase

Every deployment should have a separate repository, i.e. if you have 3 different deployments then you will have 3 different repositories. There is always a one-to-one correlation between the codebase and the app. 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 theÂ 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. 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

The dependency management should be done by the application itself.

Config

Usually whatever code you deploy across environments like development, staging and production might be same but the only thing that might vary is the configs. Usually the configs contain below

- Resource handles to the database, Memcached, and other backing services.
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Config should be out of the repository, i.e. the code and the config should be altogether separate. If there is a change in the config then in that case you don’t have to redeploy the application. We can also store configs in the environment variables. Environment variables are easy to change without changing any code.

Storing configs as constants in the code is violation of 12 factor principles.

Backing services

A backing service is a service the your application consumes over the network as part of the operation. Examples are databases such as MySQLÂ or PostgreSQL, messaging/queueing systems such as RabbitMQ or ActiveMQ, SMTP services for outbound email such as Postfix, and caching systems such as Memcached. Each backing service that your application consumes is also called as a resource.

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 appâ€™s code.

Resources can be attached to and detached from deploys at will. For example, if the appâ€™s database is misbehaving due to a hardware issue, the appâ€™s administrator might spin up a new database server restored from a recent backup. The current production database could be detached, and the new database attached â€“ all without any code changes.

Build, Release and Run

Your application should follow a proper sequence i.e. Build, Release and then Run. It will never happen that you release the application without building it.

Build – A Build stage transforms your code into and executable unit called as build.
Release – In this stage the build produced by the build stage is combined with the deployâ€™s current config. The resulting *release* contains both the build and the config and is ready for immediate execution in the execution environment.
Run

– This stage runs the app in the execution environment.

Usually every release will have a unique release ID such as timestamp 2021-12-19-20:32:17. Any changes made to the code thereafter should have a new release ID. The deployment tools or the release management tools should have the ability to rollback to previous release in case there are any issues with the current release.

Stateless processing

You applications should not have any state maintained on the server. It should be stateless and share-nothing. Any data that needs to persist must be stored in a stateful backing service, typically a database.

The memory space or filesystem of the process can be used as a brief, single-transaction cache. For example, downloading a large file, operating on it, and storing the results of the operation in the database. The twelve-factor app never assumes that anything cached in memory or on disk will be available on a future request or job â€“ with many processes of each type running, chances are high that a future request will be served by a different process. Even when running only one process, a restart (triggered by code deploy, config change, or the execution environment relocating the process to a different physical location) will usually wipe out all local (e.g., memory and filesystem) state.

Some web systems rely on sticky session â€“ that is, caching user session data in memory of the appâ€™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.

Port Binding

The application should be responsible to tell which port the app should be running. In that way, one app can become the backing service for another app, by providing the URL to the backing app as a resource handle in the config for the consuming app.

Concurrency

When we scale the application we should ensure that the concurrency of the application is not compromised and we should make use of appropriate process to carry out the work.

Disposability

When the App starts up or shuts down it should happen quickly.

DEV/PROD parity

Whenever you are deploying any application to the DEV, it should be the same application you are deploying to the PROD. It should not have any parity between the environments.

So the main moto here is we should keep the development, staging and production as similar as possible.

Logs

The application logs allow you to view what's going on in your running application.

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.

The event stream for an app can be routed to a file, the logs are stored on multiple machines in the cloud so logs should be streamed or watched in realtime using terminal or Splunk or any other tool.

Admin Processes

When we are going to perform one time activity for example running some one-time script or running database migration, it should have a separate process rather than combining it with other process. Admin code or scripts must ship with application code to avoid synchronization issues.

Category

1. Design

Date Created
December 18, 2021
Author
kk-ravi144gmail-com