The Twelve Factors - 2022

**Remember**

🡺 **Codebase, Dependencies, Config Logs Backing services**

🡺 **Admin Processes Concurrency Disposability**

🡺 **Build, release, run via Port binding with Dev/prod parity**

**Codebase:** One codebase tracked in revision control, many deploys

**Dependencies**: Explicitly declare and isolate dependencies

**Config**: Store config in the environment

**Logs**: Treat logs as event streams

**Backing services**: Treat backing services as attached resources

**Admin processes**: Run admin/management tasks as one-off processes

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

**Concurrency**: Scale out via the process model

**Disposability**: Maximize robustness with fast startup and graceful shutdown

**Build, release, run**: Strictly separate build and run stages

**Port binding**: Export services via port binding

**Dev/prod parity**: Keep development, staging, and production as similar as possible

**Codebase: One codebase tracked in revision control, many deploys**.

As per 12-factor app, every service should have its own codebase. Multiple apps sharing the same code are a violation of the twelve-factor.

**Dependencies**: **Explicitly declare and isolate dependencies**

The **twelve-factor app should always explicitly declare all its dependencies**. We use dependency management tools like Maven and Gradle

**Config**: **Store config in the environment**

**Externalize the configurations from the application**. We can store all the application specific configuration in environment variable or we can use Spring Cloud Config Server. The safest approach to save and maintain an application’s configuration is to use env vars, which implies externalizing and storing the config in the environment

**Logs**: **Treat logs as event streams**

Twelve-factor app principles advocate **separating the log generation and processing the log's information**. We can use Tools like **Splunk** or **ELK** for log processing or ingestion.

**Backing services**: **Treat backing services as attached resources**

**A backing service is a networked attached service that your application consumes to do its job**. This might be a **MongoDB instance**, **PostgreSQL database**, NFS Broker Service, etc.  **A backing service is an application/service the app consumes over the network** as part of its normal operation. **Backing Services principle encourages to handle external components that can be provisioned and managed as attached resources.**

**Admin Process**: **Run admin/management tasks as one-off processes**

**What is one-off process**? This principle describes that **administrative or management tasks should be executed as separate short-lived processes or ad-hoc process**. **This factor discourages putting one-off admin or management tasks inside your microservices**. There is a number of one-off processes as part of the application deployment like **data migration**, executing one-off scripts in a specific environment. As our app is used by a vast number of end users, the visitor count has surpassed eight figures, and to ensure accurate tracking, we need to reset the visitor count to 0. This can be achieved by running a one-time database script. Identify tasks that are administrative in nature, such as database migrations, data imports, or configuration updates. Create standalone scripts or executables for these tasks, ensuring they can be executed independently of the main application.

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

**A twelve-factor app is expected to run in an execution environment as stateless processes.**In other words, they cannot store persistent state locally between requests. **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](http://memcached.org/) or [Redis](http://redis.io/)**.**

**Concurrency**: **Scale out via the process model**

This talks about scaling the application. Twelve-factor app principles suggest to consider running your application as multiple processes/instances instead of running in one large system. You can still opt-in for threads to improve the concurrent handling of the requests. In a nutshell, twelve-factor app principles advocate for **horizontal scaling** instead of vertical scaling.

Vertical scaling 🡺 Add additional hardware to the system

Horizontal scaling 🡺 Add additional instances of the application

**Disposability**: **Maximize robustness with fast startup and graceful shutdown**

The twelve-factor app's processes are disposable/transient, meaning they can be started or stopped at a moment's notice. When the application is shutting down or starting, an instance should not impact the application state. Graceful shutdowns are very important. **The system should not get impacted when new instances are added or takedown** the existing instances as per need. This is also known as system disposability.

**Build, release, run**: **Strictly separate build and run stages**

The application must have a strict separation between the build, release, and run stages.

* **Build stage:**transform the code into an executable bundle/ build package.
* **Release stage:** get the build package from the build stage and combines with the configurations of the deployment environment and make your application ready to run.
* **Run stage:** It is like running your app in the execution environment.

You can use CI/CD tools to automate the builds and deployment process. Docker images make it easy to separate the build, release, and run stages more efficiently.

**Port binding: Export services via port binding**

The principle of **Port Binding** asserts that a service or application is identifiable to the network by port number, not a domain name. The reasoning is that domain names and associated IP addresses can be assigned on-the-fly by manual manipulation and automated service discovery mechanisms. **This principle dictates that an application should expose its services through a well defined port**, making it easy to discover and access the service.

**In short, this is all about having your application as a standalone instead of deploying them into any of the external web servers.** Spring boot is one example of this one. Spring boot by default comes with embedded tomcat, jetty, or undertow.

**The essential idea behind the principle of port binding is that the uniform use of a port number is the best way to expose a process to the network. For example, the patterns have emerged in which port 80 is conventional for web servers running under HTTP, port 443 is the default port number for HTTPS, port 22 is for SSH, port 3306 is the default port for MySQL, and port 27017 is the default port for MongoDB**.

**Dev/prod parity**: **Keep development, staging, and production as similar as possible**

The twelve-factor methodology suggests keeping the **gap between development and production** environment as minimal as possible. This reduces the risks of showing up bugs in a specific environment. Now, technology like Spring Boot and Docker automatically bridge this gap to a great extent. A containerized application is expected to behave the same, no matter where we run it.