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 application should have its own codebase (repos). Multiple apps sharing the same code are a violation of the twelve-factor. In Microservices, every service should have its own codebase.

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

The **twelve-factor app should always explicitly declare all its dependencies**. We should do this using a dependency declaration manifest. Java has multiple dependency management tools like Maven and Gradle. **All the application packages will be managed through package managers like sbt, maven, Gradle.**

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

**Anything that varies between the deployment environments is considered as configuration**. This includes **Database connections** and **credentials**, **system integration endpoints**, Application-specific information like IP Addresses, ports, and hostnames. **Externalize the configurations from the application**. you can manage the configurations for your applications from a source control like git (spring-cloud-config) and use the environment variables.

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

Twelve-factor app principles advocate **separating the log generation and processing the log's information**. In Microservices, observability is the first-class citizen. Observability can be achieved through using APM tools (**ELK**, Newrelic, and other tools) or log aggregations tools **like Splunk, logs**, etc.

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

**A backing service is an application/service the app consumes over the network** as part of its normal operation. Database, Message Brokers, any other external systems can be treated as Backing service.

**12-factor app can automatically swap the application from one provider to another without making any further modifications to the code base**. Let us say, you would like to change the database server from MySQL to Aurora. To do so, you should not make any code changes to your application. Only configuration change should be able to take care of it. In a microservice ecosystem, anything external to service is treated as attached resource.

**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**. **One-off tasks** can be independent of any process or can be added to any active processes. **One-off admin processes should be run in an identical environment**. They run against a [release](https://12factor.net/build-release-run), using the same [codebase](https://12factor.net/codebase) and [config](https://12factor.net/config) as any process run against that release. Admin code must ship with application code to avoid synchronization issues. **Twelve-factor principles advocates for keeping such administrative tasks as part of the application codebase in the repository**. 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.

**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. Some web systems rely on [“sticky sessions”](http://en.wikipedia.org/wiki/Load_balancing_%28computing%29#Persistence) – 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](http://memcached.org/) or [Redis](http://redis.io/)**.**

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

Twelve-factor app processes [should never daemonize](http://dustin.github.com/2010/02/28/running-processes.html) or write PID files. 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, 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 must ensure the correct state.

**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. Let us understand each stage in more detail.

* **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 twelve-factor app is completely self-contained and doesn't 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 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.

**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.