

A-11) 12 Factors App:-

The twelve-factor app is a methodology used to develop SaaS apps or web apps with the help of Microservices. It serves as a practice guide to building apps when pressed for time or cost to hire new developers for a project and make them scale up without significant changes to tools or architecture. It allows organizing projects effectively and managing scalable applications.

This methodology was first introduced by the developers at Heroku in 2012 who created hundreds of appl^s and developed this methodology based on their experience. The methodology incorporates 12 principles each applying to a subset of an app and guiding developers in finding the best way to manage apps as a whole.

The principles it suggests ~~that~~ are not constrained to any particular programming language or database.

→ The 12-Factor Principles:-

1. Codebase (One code base tracked in revision control, many deploys)

There should be one codebase with different environments.

The guideline expressed in this first factor suggests the principle of not creating another codebase, just for the sake of setup in different environments. For inst. there shouldn't be 2 repositories (productⁿ & developⁿ)

for one codebase. A distributed version control sys can maintain & monitor the proj. source code. For developers, this makes accessibility easier. Diff. environ. represent diff. states. They all share the same codebase, whatever the state might be.

2. Dependencies (Explicitly declare and isolate the depen.^{ies})

All dependencies should be declared, with no implicit reliance on sys. tools or libraries.

Including the depen. into the codebase is a bad practice since it could create prob. such as the underlying dependency of the platform. For inst, if the uploaded modules were from a windows pc perspective, & the codebase was downloaded by a Mac OS developer, they would undoubtedly have prob. running the proj.

It's better to use a package manager - depending on the tech. stack - to download the depen. on your respective sys by reading a dep. declaratⁿ manifest containing the names & versions of the depen.

3. config (Store configuratⁿ in an environ.)

Configurat^{ns} are an imp. part of any appⁿ, especially when there is a need to support multiple environ or clients.

A configuratⁿ that varies betⁿ deployments should be stored in the environ. No one wants configuratⁿ info to be publicly available, such as database connectⁿ info or other sensitive data. The configuratⁿ is also large dependent on the environ. where it's based.

There should be a strict separatⁿ betⁿ config & code. code should remain the same irr. of where the applⁿ is being deployed, but config can vary.

4. Backing services (treat bac. ser. resources as attached resources)

All backing services are treated as attached res. & attached & detached by the executⁿ environ. Data-bases, external storage, msg. queues, etc, are ext. services & they can be considered as a resource. They may be accessed via some web-addresses / URLs or similar req. then specified in the config. That way, the sources of the ser. can be modified w/o affecting the core of the app.

5. Build, Release, Run (Strictly separate build & run stages)

The 12-fact. app ~~restricts~~ req. a strict separatⁿ betⁿ Build, Release & Run stages.

° First, the build phase. It's the assembly of the SC into an executable bundle while loading depend. & creating assents. The build phase starts every time a new code needs to be deployed.

° 2nd, the release stage: Here, the code produced during the build stage is combined with the deploy's current config. It contains both build & config & is ready for executⁿ.

° 3rd, the run stage: The final stage runs the applⁿ in the executⁿ environ. It shouldn't be intervened by any other stage.

SINRAN KUMARI
AOK

12AMUX 1132412

PAGE No.

DATE

/ /

G. Processes (execute the app as 1 or more stateless processes)

App's should be ~~per~~ deployed as one or more stateless processes with persisted data stored on a backing service, typically a db.

The key principle of this factor is that the process is stateless and shares absolutely nothing.

While many web sys rely on "sticky sessions", storing data in the session expecting that the next req. will come from the same service contradicts this approach.

7. Port Binding (Export services via port binding)

A twelve factor app should be independent of additional application. Every function should be its own process.

This factor refers to an application binding itself a particular port and responding to all the requests from that port. The port is declared as an environment variable and provided during execution.

Applications built on the basis of principle do not depend on a web server. The application is self-contained and runs standalone. The web server is packaged as a library and bundled with the application.

8. Concurrency (Scale out via the process model)

Based on the requirement, each process in your application should be able to scale, restart, or clone itself. Doing this can enhance scalability.

Instead of making a single process even larger, developers can create multiple processes and then distribute the load of their application among those processes. Using this approach, developers can build apps that can handle various workloads by assigning each workload to a process type. For instance, it is possible to delegate HTTP request to a web process and long background tasks to the worker process.

9. Disposability (maximize the robustness with fast startup and graceful shutdown)

A 12-factor app should be built on simple processes that start quickly, work quickly and finish correctly. This helps developers to scale up processes while still allowing them to restart them if anything goes wrong.

Building disposability into app means graceful shutdown. It should clean up all utilized resources and shut down seamlessly. An app can be easily relaunched when designed in this way. Likewise, when processes terminate they should finish their current request, reject any incoming request and exit.

10. Dev-Prod Parity (Keep development, staging, and production as similar as possible)

The twelve-factor article proposes keeping the difference between development and production environment minimal. This reduces the likelihood of bugs turning up in a particular environment.

Developers should strive to use the same third party service between development and production. To keep differences between development and production minimal, teams collaborating on a project should use the same operating system, backing services, and dependencies. As a result continuous development takes less time.

This also encourages the idea of rapid app development.

The process of continuous deployment becomes hassle-free when developers reduce the number of differences between the development and production stages.

When troubleshooting production problems or understanding user behavior logs become extremely important. Logs provide visibility into a running application's behavior.

A 12 factor app should not concern itself with routing or storage of its ~~data~~ output stream. Neither should it not attempt to write to a manage logfile. All logfiles should be written to stdout and the environment decides how to process this stream. This can be shipping to the console or saving and routing to a log indexing and analysis system.

SIMRAN KUMARI
A018

SIMRAN KUMARI

PAGE No. 00

DATE / /

12. Admin processes:

This factor states that any admin processes should be run in an identical environ. as the regular long-running processes of the app. In a local deploy, dev. use a direct shell command inside the app's checkout directory to perform one-time admin processes.

→ Advantages:-

→ Speed up efficiency

→ Besides, the time spent on learning & applying these guidelines can help companies save a great deal of money.

→ There are cases when it makes sense to diverge from some of the guid. such as logs, but it's better to follow the 12-Factor guideline.