12 Factor App Principles

Best 12 practices to develop application developed to run a service .

* Drafted by Heroku for deployed as service on cloud platform
* Design ,develop ,deploy and maintain software solution to derive business value from it.

Example – Calculating applicable tax is a generic function in many domains.

Subscribe to commercial service offering .

Service offering is known as software -as -a – service.



1. Codebase
   * Version Control System
   * Each application should have its own codebase
   * Avoid using multiple codebases. Can have multiple branches.
   * Multiple app sharing the same code are violating the 12 factor – Use Shared Libraries
   * Instance of App like production, staging ,QA and every developer should have their local instance environment which qualifies for deploy.
   * **Microservice** : Every Service Should have own codebase to make CI/CD process for your application
2. Dependencies (Isolate the dependencies)
   * Define all Project dependencies in XML file – Project object Model
   * Packages should be managed by sbt, maven.
   * In the non-containerized environment, you can go for configuration management tools like chef, ansible to install system-level dependencies
   * For containerized environment , go for docker file.
3. Configuration (Store Configuration is an Environment):
   * Anything that varies between the deployment environment is considered as configuration.

* DB Connections, credentials , system integration end points
* Credentials to external services like Amazon S3 or Twitter or any other External App.
* Application specific information like IP address , pots and hostnames.
  + Must be managed using environment variables
  + Spring-cloud-config and use the environment variables

1. Backing Services (Backing resource as an attached resource)
   * Database, message brokers, any external system – app communicates is treated as backing service.
   * Plugin based implementation to support multiple providers
2. Build , Release and run (Strictly separate build and run Stages)
   * Build stage : transform the code into a executable bundle

mvn clean compile test package

* + Release stage: get the build package from build stage and combines with configuration of deployment environment and makes application ready to run.

Packer+ansible to create docker images :

Packer build application .json

Run stage: running the app in execution environment

Use docker as a container to release our application.

Docker run –name <container-id) -it <image-id>



1. Processes (Execute the app as one or more stateless processes)
   * It is executed as process in environment. App can have more than once instances to meet customer demands
   * Use **Redis or Memcached** – cache provider based on requirements – highly scalable without impact on the system.
   * Stateless nature of REST -to make horizontally scaled as per needs.
2. Port Binding (Export services Via Port Binding)
   * Application may export multiple service s like FTP or Websockets and aceesible using http .
3. Concurrency(Scale out via process Model)
   * Should deploy more copies of application instead of making app larger.
   * Supports horizontal scaling of app instead of vertical scaling.
4. Disposability (Maximize Robustness with Fast Startup and Graceful Shutdown)
   * Application state should not get impact in case of startup and shutdown.
   * Graceful shutdown

* As of [Spring Boot 2.3](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.3-Release-Notes#graceful-shutdown), Spring Boot now supports the graceful shutdown feature for all four embedded web servers (Tomcat, Jetty, Undertow, and Netty) on both servlet and reactive platforms.
* **To enable the graceful shutdown, all we have to do is to set the *server.shutdown* property to *graceful*** in our *application.properties* file:
* server.shutdown=graceful
* Then, Tomcat, Netty, and Jetty will stop accepting new requests at the network layer. Undertow, on the other hand, will continue to accept new requests but send an immediate 503 Service Unavailable response to the clients.
  + Docker containers can be started or stopped instantly.
  + Storing request , state,session data in queues or in any backing service ensures request is handles seamlessly in case of crash.

1. Dev/ Prod Parity :
   * Keeping the gaps between dev and prod environment as minimal as possible.

Like deployment process.

* + Run using containerization techniques.

1. Logs( Treat logs as event Streams)
   * Logs provide visibility to behaviour of application
   * Observability can be archieved using APM tools (ELK,Newrelic and other tools) or log aggregation like Splunk etc.
   * Go to central dashboard of your tool and search for it.
2. Admins Process :
   * Containerization helps to run the one-off processes as task and shutdown automatically.