

Driving Efficiency and Agility

Exploring the Power of the 12-Factor App

by Evren Tan



About Me



12-Factor App?



★ A methodology for building SaaS apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

★ ➡ <https://12factor.net/>



And these factors are;

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release & Run
6. Processes

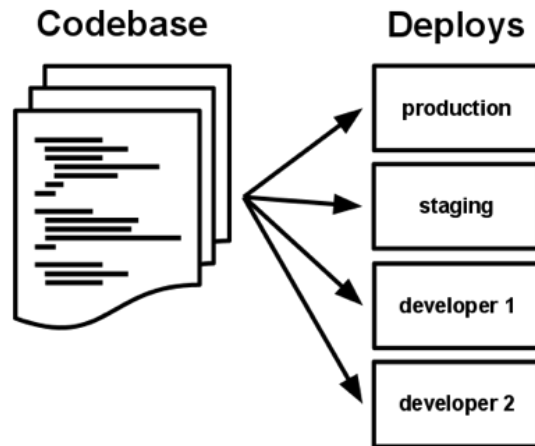
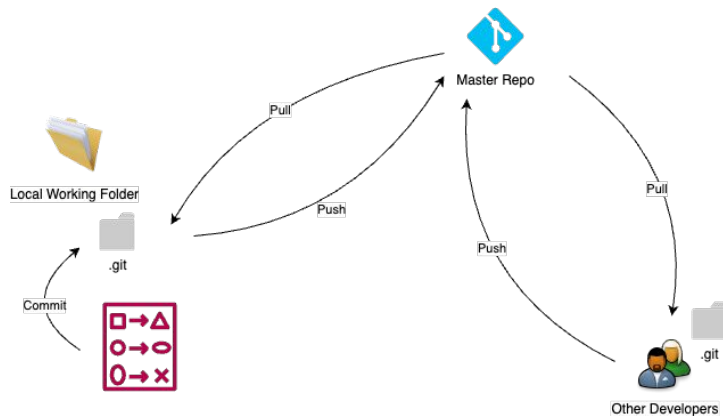
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin Processes

Codebase

❌ Multiple Codebases \neq One App ➡ A Distributed System

❌ Multiple Apps Sharing the Same Codebase

✅ One Codebase per App with Many Deploys



<https://12factor.net/codebase>

Dependencies

! A 12-factor app never relies on implicit existence of system-wide packages.

✓ It declares all dependencies, completely and exactly, via a dependency declaration manifest.



A dependency isolation tool

```
repositories {
    mavenCentral()
    maven { url "https://repo.spring.io/snapshot" }
    maven { url "https://repo.spring.io/milestone" }
    maven { url "https://repo.spring.io/release" }
}

ext {
    set('springCloudVersion', "2022.0.0")
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    implementation 'org.springframework.cloud:spring-cloud-config-client'
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'
    implementation group: 'javax.validation', name: 'validation-api', version: '2.0.1.Final'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.hibernate.validator:hibernate-validator'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
    implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'
    implementation group: 'org.springdoc', name: 'springdoc-openapi-ui', version: '1.6.4'
    compileOnly 'org.springframework.boot:spring-boot-starter-jdbc'
    compileOnly 'org.projectlombok:lombok'
    implementation 'org.flywaydb:flyway-core'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    compileOnly group: 'javax.xml.bind', name: 'jaxb-api', version: '2.3.0'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"
    }
}
```

Config

? Store configs static in the codebase ❌

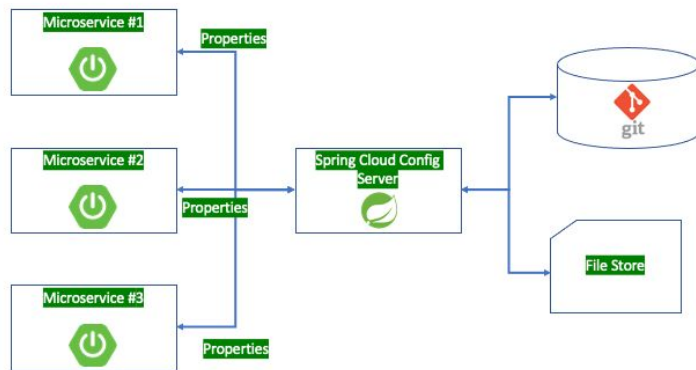
✓ Strict Separation Config from the Codebase

✓ The twelve-factor app stores config in environment variables

🔄 Group configs in the same batch, i.e., dev, test, staging, production



High-Level Spring Cloud Config Server Architecture



```
community-workspace - config-server-data - community-manager - C community-manager-dev-properties
└─ Project
   └─ community-workspace - /Volumes/Development/community-workspace
      └─ .
         ├── .github
         ├── .idea
         ├── api-gateway
         ├── community-manager
         ├── config-server
         ├── config-server-data
         ├── api-gateway
         └── community-manager
            ├── community-manager-dev-properties
            ├── community-manager-local-properties
            ├── service-discovery
            ├── user-manager
            ├── venue-manager
            ├── README.md
            ├── db-migration-manager
            ├── event-manager
            ├── postman-collection
            └── CommunityWorkspacesPostmanCollection.json
               ├── service-discovery
               ├── user-manager
               ├── venue-manager
               ├── gitignore
               ├── CODE_OF_CONDUCT.md
               ├── CONTRIBUTORS.md
               ├── LICENSE
               ├── README.md
               └─ community-manager-dev-properties
                  1 spring.application.name = community-manager
                  2 server.port = 8081
                  3
                  4 # Database Settings
                  5 spring.datasource.driver-class-name = org.postgresql.Driver
                  6 spring.datasource.url = jdbc:postgresql://localhost:5432/community-manager-db?currentSchema=community-manager
                  7 spring.datasource.username = admin
                  8 spring.datasource.password = admin
                  9
                  10 # Hikari Connection Pool Settings
                  11 spring.datasource.hikari.schema = community-manager
                  12 spring.datasource.hikari.schemaName = 5
                  13 spring.datasource.hikari.maximumPoolSize = 20
                  14 spring.datasource.hikari.idleTimeout = 30000
                  15 spring.datasource.hikari.poolName = CommunityManagerJPAHikariCP
                  16 spring.datasource.hikari.maxLifetime = 200000
                  17 spring.datasource.hikari.connectionTimeout = 30000
                  18
                  19 # JPA Specific Configs
                  20 spring.jpa.properties.hibernate.show_sql = true
                  21 spring.jpa.properties.hibernate.format_sql = true
                  22 spring.jpa.properties.hibernate.use_sql = true
                  23
                  24 # Enable logging
                  25 logging.level.org.hibernate.SQL = ERROR
                  26
                  27 # Flyway Settings
                  28 spring.flyway.enabled = true
                  29 spring.flyway.url = jdbc:postgresql://localhost:5432/community-manager-db
                  30 spring.flyway.user = admin
                  31 spring.flyway.password = admin
                  32 spring.flyway.schemas = community-manager
                  33 spring.flyway.baseline-on-migrate = true
                  34
                  35 # Discovery Server
                  36 eureka.client.service-url.defaultZone = http://localhost:8080/eureka
                  37
                  38 # Actuator Settings
                  39 management.endpoints.web.exposure.include = *
```

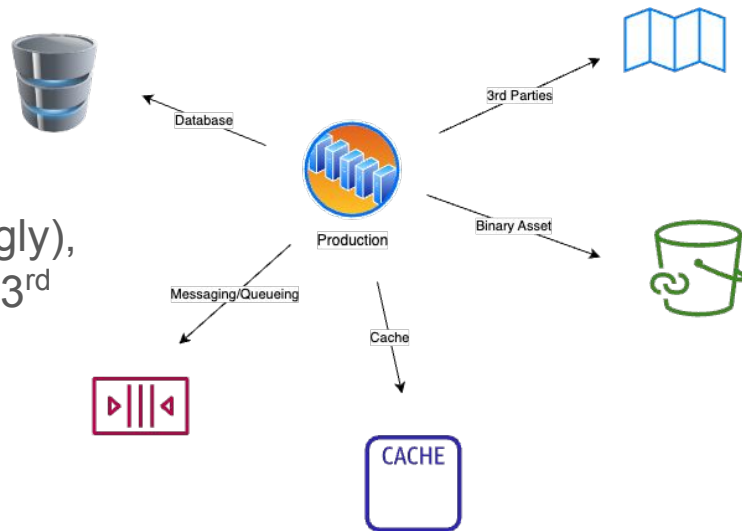

Backing Services

i A backing service is any service the app consumes over the network as part of its normal operation.

→ Databases (PostgreSQL, MongoDB),
Messaging/Queueing Services (Kafka, RabbitMQ),
Caching (Redis), Metrics Gathering (New Relic, Loggly),
Binary Asset Services (Amazon S3), API accessible 3rd
Parties (Twitter, Google Maps, etc)

★ Treat backing services as attached resources

★ Resources must be designed according to loose coupling pattern

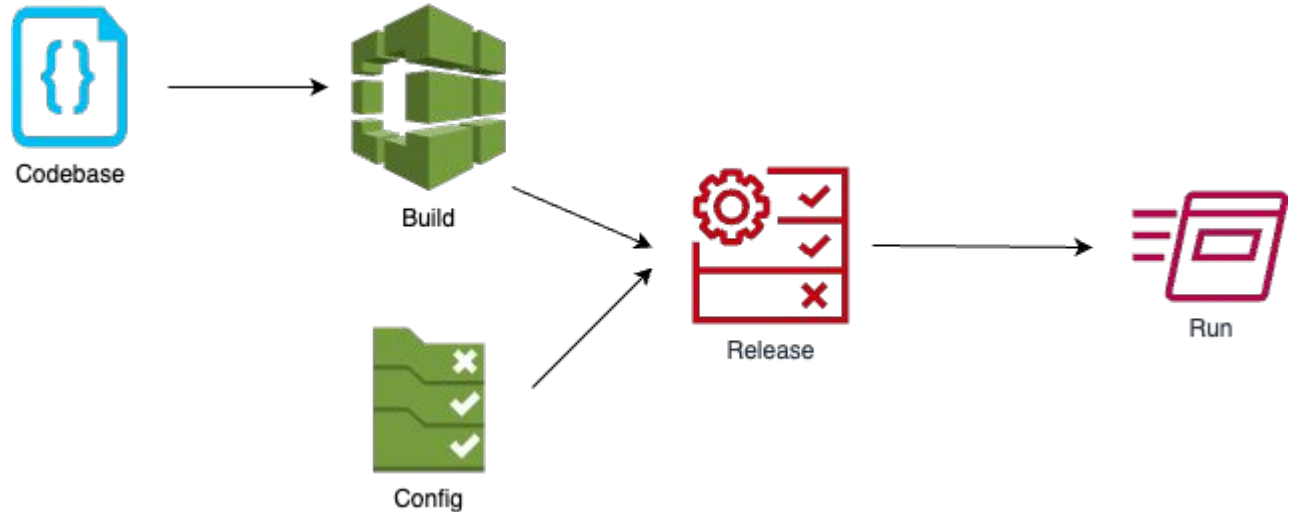


Build, Release & Run

! Strictly separate build, release and run stages

i Three stages;

1. Build Stage
2. Release Stage
3. Run Stage



Processes

- ★ Twelve-factor processes are stateless and share-nothing.
- ★ Any data that needs to persist must be stored in a stateful backing service, typically a database.
- ✓ Execute the app as one or more stateless processes

Port Binding

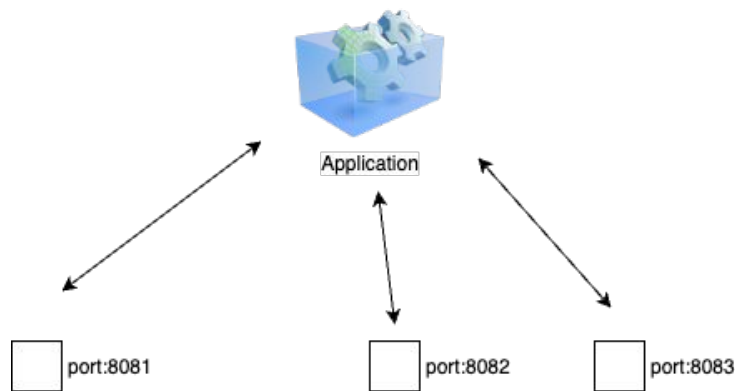
★ Export services via port binding

📘 Improves an application's portability and security

🤔 An App has three services and all binds to different ports ?

Scalability, portability, security, routing, load balancing
(container native load balancing)

? Treats like a backing service. Think about a Java app behind a Nginx web server



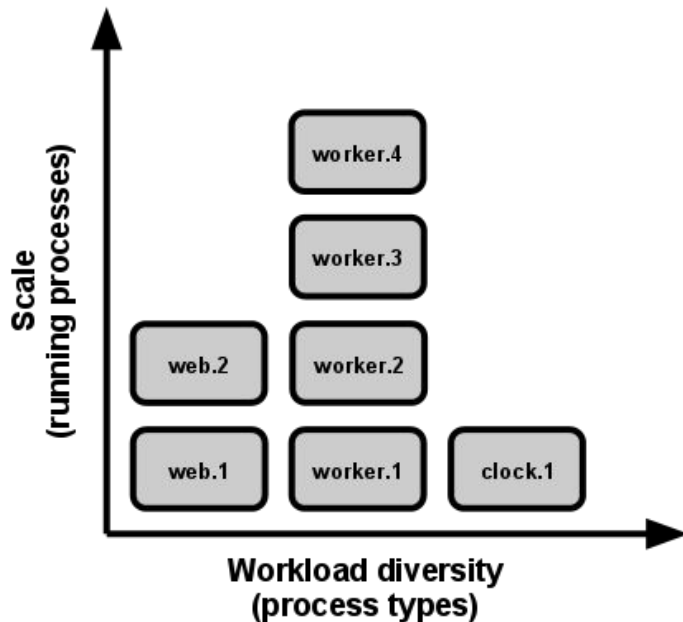
Concurrency

★ Scale out via the process model

🏆 Processes are a first class citizen for a 12-Factor App

🏗️ The developer can architect their app to handle diverse workloads by assigning each type of work to a process type.

- HTTP requests may be handled by a web process, and long-running background tasks handled by a worker process.



Disposability

★ Maximize robustness with fast startup and graceful shutdown

📘 Processes are disposable, meaning they can be started or stopped at a moment's notice.

⚖️ Fast elastic scaling

💻 Rapid deployment of code or config changes

💪 Robustness of production deploys

💣 A 12-Factor app is architected to handle unexpected, non-graceful terminations

➡️ Crash-only Software ➡️ A software that crashes safely and recovers quickly

Dev/Prod Parity

★ Keep development, staging, and production as similar as possible

i The 12-factor app is designed for continuous deployment by keeping the gap between development and production small → Smaller Feedback Loop

	Traditional app	Twelve-factor app
Time between deploys	Weeks	Hours
Code authors vs code deployers	Different people	Same people
Dev vs production environments	Divergent	As similar as possible

! The twelve-factor developer resists the urge to use different backing services between development and production

🐳 Containerization tools like Docker

Logs

★ Treat logs as event streams

📘 Logs are the stream of aggregated, time-ordered events collected from the output streams of all running processes and backing services

! A 12-factor app never concerns itself with routing or storage of its output stream

✗ Write to or manage log files in the local file system

✓ Write to STDOUT as unbuffered data

Admin Processes



What are Admin Processes;

- Running DB migrations
- Running reports
- Running one-time scripts committed into the app's repo



Run admin/management tasks as one-off processes



Each one of those processes should be run against a new instance having identical configs with the app



Ensures that those background tasks do NOT have impact on the standard process

Q&A



Quiz Time :)



<https://form.jotform.com/evrentan/the-12-factor-app-quiz>