

12 factor app


IIHT

Contents

▶ <https://12factor.net>

- ▶ The twelve-factor app is a methodology for building software-as-a-service 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.

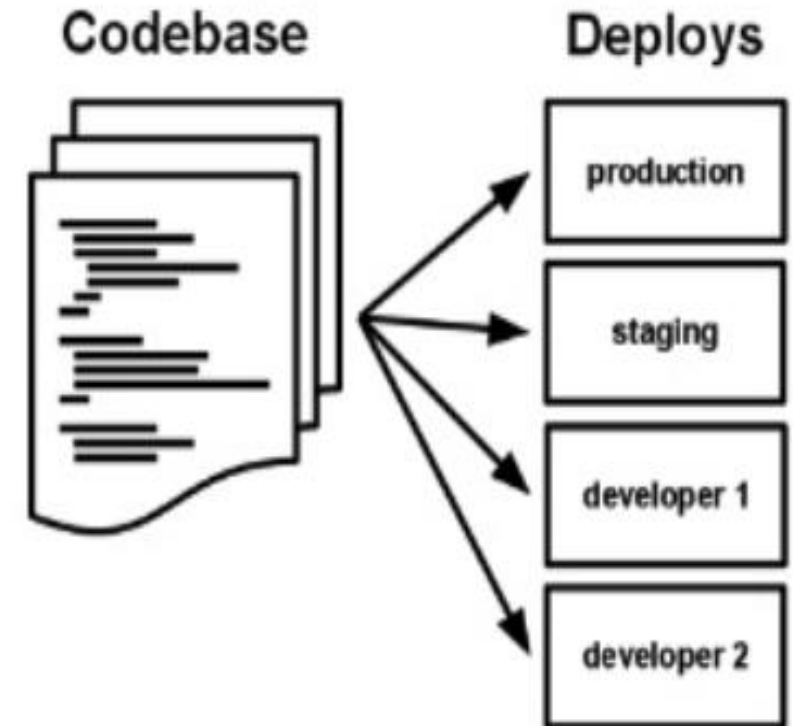
THE FACTORS

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
- 

1. Codebase

“One codebase tracked in revision control, many deploys.”

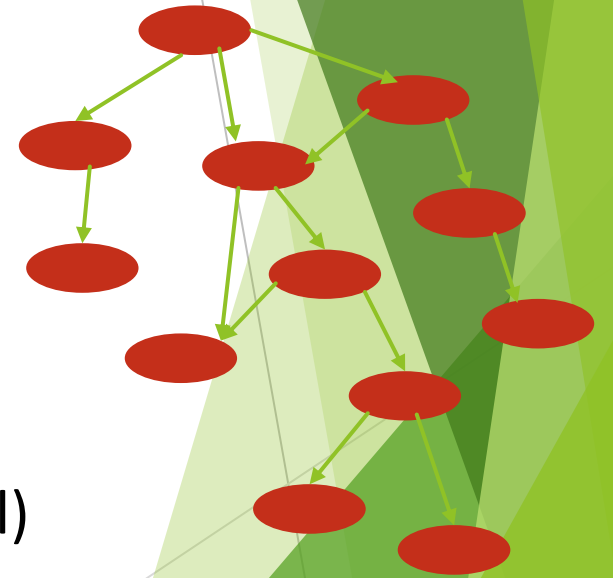
- Dedicate smaller teams to individual applications or microservices.
 - Following the discipline of single repository for an application forces the teams to analyze the seams of their application, and identify potential monoliths that should be split off into microservices.
- Use a single source code repository for a single application (1:1 relation). Deployment stages are different tags/branches
- i.e. use a central git repo (external Github/GitHub Enterprise also suitable)



2. Dependencies

“Explicitly declare and isolate dependencies”

- ▶ A cloud-native application does not rely on the pre-existence of dependencies in a deployment target.
- ▶ Developer Tools declare and isolate dependencies
 - ▶ Maven and Gradle for Java
- Each microservice has its own dependencies declared (e.g. pom.xml)



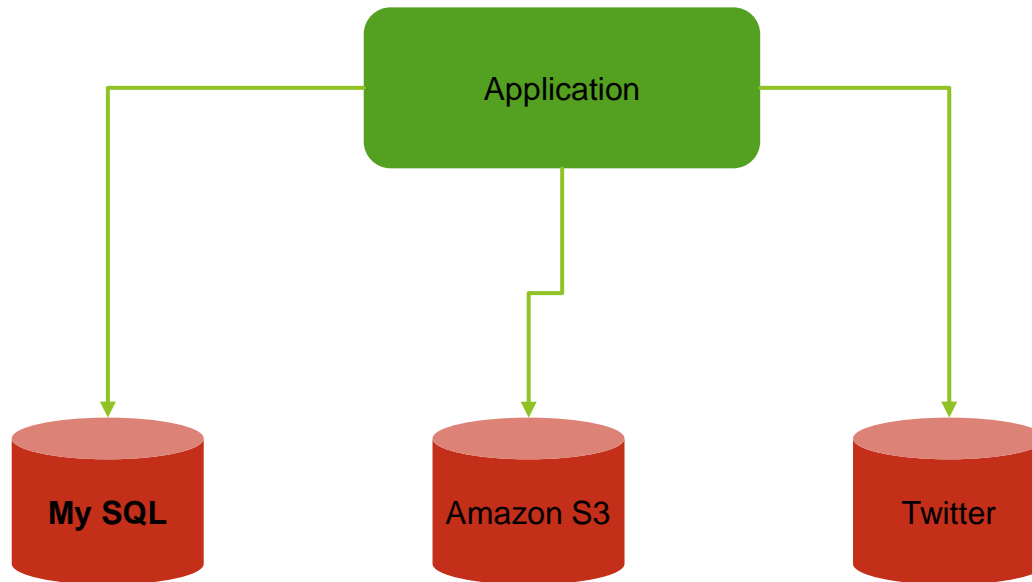
3. Config

“Store config in the environment”

- Changing config should not need to repackaging your application
- Use Kubernetes configmaps and secrets (rather than environment variables) for container services
- Use MicroProfile Config to inject the config properties into the microservices

4. Backing services

“Treat backing services as attached resources”



5. Build, release, run

“Strictly separate build and run stages”

- Source code is used in the build stage. Configuration data is added to define a release stage that can be deployed. Any changes in code or config will result in a new build/release
 - Needs to be considered in CI pipeline
- CD pipeline is independent from CI pipeline

6. Processes

“Execute the app as one or more stateless processes”

Stateless and share-nothing

Rest API



7. Port binding

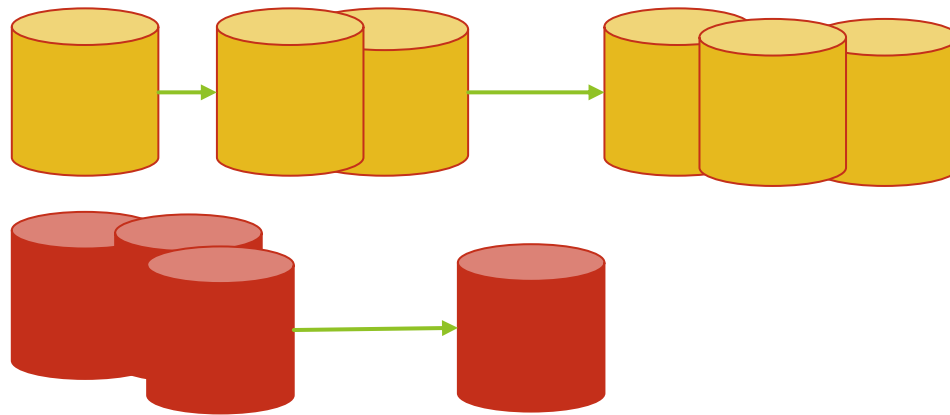
“Export services via port binding”

- Applications are fully self-contained and expose services only through ports. Port assignment is done by the execution environment
- Ingress/service definition of k8s manages mapping of ports
- Use MP Config to inject ports to microservices for chain-up invocations

8. Concurrency

“Scale out via the process model”

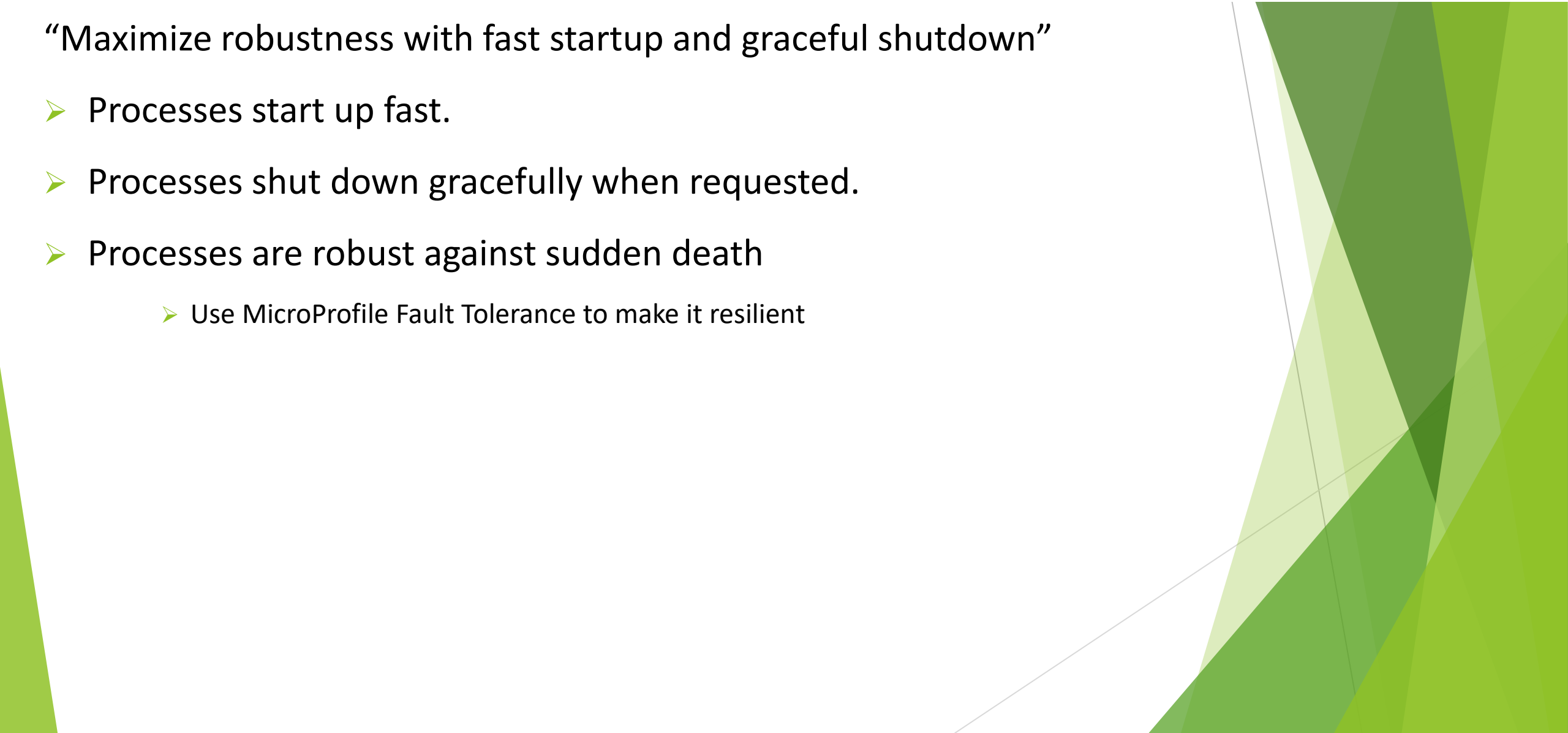
- Applications use processes independent from each other to scale out (allowing for load balancing)
- To be considered in application design
- Cloud autoscaling services: [auto]scaling built into k8s
- Build microservices



9. Disposability

“Maximize robustness with fast startup and graceful shutdown”

- Processes start up fast.
- Processes shut down gracefully when requested.
- Processes are robust against sudden death
 - Use MicroProfile Fault Tolerance to make it resilient



10. Dev/prod parity

“Keep development, staging, and production as similar as possible”

- Development and production are as close as possible (in terms of code, people, and environments)
 - Can use helm to deploy in repeatable manner
 - Use (name)spaces for isolation of similar setups
- Can use docker

11. Logs

“Treat logs as event streams”


- App writes all logs to stdout
- Use a structured output for meaningful logs suitable for analysis. Execution environment handles routing and analysis infrastructure

12. Admin processes

“Run admin/management tasks as one-off processes”

- Tooling: standard k8s tooling like “kubectl exec” or Kubernetes Jobs
- Also to be considered in solution/application design
- For example, if an application needs to migrate data into a database, place this task into a separate component instead of adding it to the main application code at startup

THE FACTORS

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
- 

References

- <https://www.12factor.net/>
- <https://slideplayer.com/slide/14275757/>