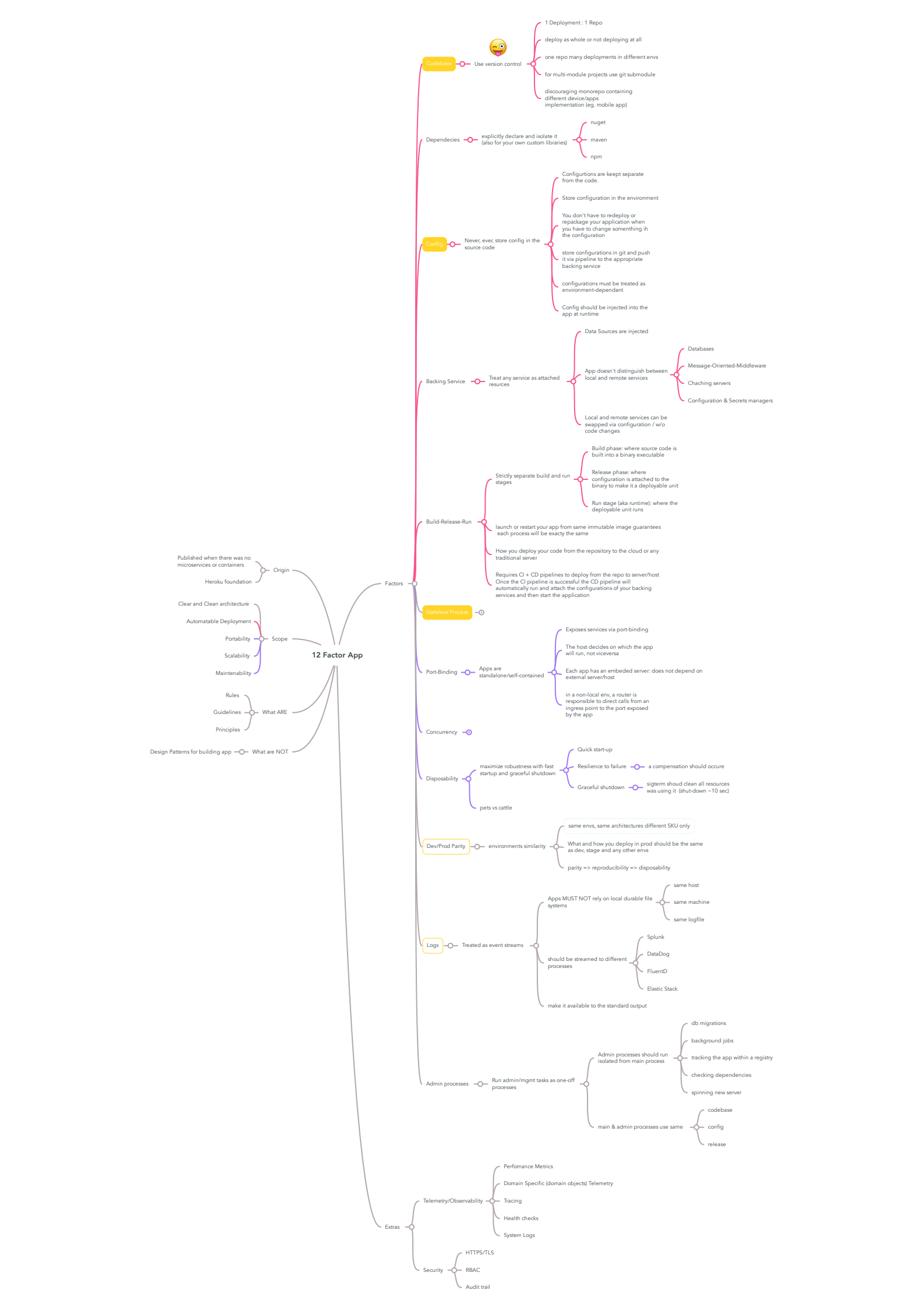
12 Factor App
Origin
Published when there was no microservices or containers
Heroku foundation
Scope
Clear and Clean architecture
Automatable Deployment
Portability
Scalability
Maintenability
What ARE
Rules
Guidelines
Principles
What are NOT
Design Patterns for building app
Factors
Codebase
Use version control
1 Deployment : 1 Repo
deploy as whole or not deploying at all
one repo many deployments in different envs
for multi-module projects use git submodule
discouraging monorepo containing different device/apps implementation (eg. mobile app)
Dependecies
explicitly declare and isolate it (also for your own custom libraries)
nuget
maven
npm
Config
Never, ever, store config in the source code
Configurtions are keept separate from the code.
Store configuration in the environment
You don't have to redeploy or repackage your application when you have to change somenthing ih the configuration
store configurations in git and push it via pipeline to the appropriate backing service
configurations must be treated as environment-dependant
Config should be injected into the app at runtime
Backing Service
Treat any service as attached resurces
Data Sources are injected
App doesn't distinguish between local and remote services
Databases
Message-Oriented-Middleware
Chaching servers
Configuration & Secrets managers
Local and remote services can be swapped via configuration / w/o code changes
Build-Release-Run
Strictly separate build and run stages
Build phase: where source code is built into a binary executable
Release phase: where configuration is attached to the binary to make it a deployable unit
Run stage (aka runtime): where the deployable unit runs
launch or restart your app from same immutable image guarantees each process will be exacty the same
How you deploy your code from the repository to the cloud or any traditional server
Requires CI + CD pipelines to deploy from the repo to server/host Once the CI pipeline is successful the CD pipeline will automatically run and attach the configurations of your backing services and then start the application
Stateless Process
Port-Binding
Apps are standalone/self-contained
Exposes services via port-binding
The host decides on which the app will run, not viceversa
Each app has an embeded server: does not depend on external server/host
in a non-local env, a router is responsible to direct calls from an ingress point to the port exposed by the app
Concurrency
Disposability
maximize robustness with fast startup and graceful shutdown
Quick start-up
Resilience to failure
a compensation should occure
Graceful shutdown
sigterm shoud clean all resources was using it (shut-down ~10 sec)
pets vs cattle
Dev/Prod Parity
environments similarity
same envs, same architectures different SKU only
What and how you deploy in prod should be the same as dev, stage and any other envs
parity => reproducibility => disposability
Logs
Treated as event streams
Apps MUST NOT rely on local durable file systems
same host
same machine
same logfile
should be streamed to different processes
Splunk
DataDog
FluentD
Elastic Stack
make it available to the standard output
Admin processes
Run admin/mgmt tasks as one-off processes
Admin processes should run isolated from main process
db migrations
background jobs
tracking the app within a registry
checking dependencies
spinning new server
main & admin processes use same
codebase
config
release
Extras
Telemetry/Observability
Perfomance Metrics
Domain Specific (domain objects) Telemetry
Tracing
Health checks
System Logs
Security
HTTPS/TLS
RBAC
Audit trail

12 Factor App

1. Factors

1.1. Codebase

1.1.1. Use version control

1.1.1.1. 1 Deployment : 1 Repo

1.1.1.2. deploy as whole or not deploying at all

1.1.1.3. one repo many deployments in different envs

1.1.1.4. for multi-module projects use git submodule

1.1.1.5. discouraging monorepo containing different device/apps implementation (eg. mobile app)

1.2. Dependecies

1.2.1. explicitly declare and isolate it (also for your own custom libraries)

1.2.1.1. nuget

1.2.1.2. maven

1.2.1.3. npm

1.3. Config

1.3.1. Never, ever, store config in the source code

1.3.1.1. Configurtions are keept separate from the code.

1.3.1.2. Store configuration in the environment

1.3.1.3. You don't have to redeploy or repackage your application when you have to change somenthing ih the configuration

1.3.1.4. store configurations in git and push it via pipeline to the appropriate backing service

1.3.1.5. configurations must be treated as environment-dependant

1.3.1.6. Config should be injected into the app at runtime

1.4. Backing Service

1.4.1. Treat any service as attached resurces

1.4.1.1. Data Sources are injected

1.4.1.2. App doesn't distinguish between local and remote services

1.4.1.2.1. Databases

1.4.1.2.2. Message-Oriented-Middleware

1.4.1.2.3. Chaching servers

1.4.1.2.4. Configuration & Secrets managers

1.4.1.3. Local and remote services can be swapped via configuration / w/o code changes

1.5. Build-Release-Run

1.5.1. Strictly separate build and run stages

1.5.1.1. Build phase: where source code is built into a binary executable

1.5.1.2. Release phase: where configuration is attached to the binary to make it a deployable unit

1.5.1.3. Run stage (aka runtime): where the deployable unit runs

1.5.2. launch or restart your app from same immutable image guarantees each process will be exacty the same

1.5.3. How you deploy your code from the repository to the cloud or any traditional server

1.5.4. Requires CI + CD pipelines to deploy from the repo to server/host Once the CI pipeline is successful the CD pipeline will automatically run and attach the configurations of your backing services and then start the application

1.6. Stateless Process

1.6.1. Execute the app as one or more stateless processes

1.6.1.1. App "share nothing"

1.6.1.2. App should not maintain any state in itself

1.6.1.3. Apps should not sharing anything between them: neither data or context.

1.6.1.4. Sharing should behappen through endpoints only

1.6.1.5. State of the application is stored only in a backing service

1.7. Port-Binding

1.7.1. Apps are standalone/self-contained

1.7.1.1. Exposes services via port-binding

1.7.1.2. The host decides on which the app will run, not viceversa

1.7.1.3. Each app has an embeded server: does not depend on external server/host

1.7.1.4. in a non-local env, a router is responsible to direct calls from an ingress point to the port exposed by the app

1.8. Concurrency

1.8.1. Easy to scale (up & out)

1.8.1.1. you can create as many instance you want of your app

1.8.1.2. scaling should be based on the load or the traffic that s coming to the environment

1.9. Disposability

1.9.1. maximize robustness with fast startup and graceful shutdown

1.9.1.1. Quick start-up

1.9.1.2. Resilience to failure

1.9.1.2.1. a compensation should occure

1.9.1.3. Graceful shutdown

1.9.1.3.1. sigterm shoud clean all resources was using it (shut-down ~10 sec)

1.9.2. pets vs cattle

1.10. Dev/Prod Parity

1.10.1. environments similarity

1.10.1.1. same envs, same architectures different SKU only

1.10.1.2. What and how you deploy in prod should be the same as dev, stage and any other envs

1.10.1.3. parity => reproducibility => disposability

1.11. Logs

1.11.1. Treated as event streams

1.11.1.1. Apps MUST NOT rely on local durable file systems

1.11.1.1.1. same host

1.11.1.1.2. same machine

1.11.1.1.3. same logfile

1.11.1.2. should be streamed to different processes

1.11.1.2.1. Splunk

1.11.1.2.2. DataDog

1.11.1.2.3. FluentD

1.11.1.2.4. Elastic Stack

1.11.1.3. make it available to the standard output

1.12. Admin processes

1.12.1. Run admin/mgmt tasks as one-off processes

1.12.1.1. Admin processes should run isolated from main process

1.12.1.1.1. db migrations

1.12.1.1.2. background jobs

1.12.1.1.3. tracking the app within a registry

1.12.1.1.4. checking dependencies

1.12.1.1.5. spinning new server

1.12.1.2. main & admin processes use same

1.12.1.2.1. codebase

1.12.1.2.2. config

1.12.1.2.3. release

2. Extras

2.1. Telemetry/Observability

2.1.1. Perfomance Metrics

2.1.2. Domain Specific (domain objects) Telemetry

2.1.3. Tracing

2.1.4. Health checks

2.1.5. System Logs

2.2. Security

2.2.1. HTTPS/TLS

2.2.2. RBAC

2.2.3. Audit trail

3. What are NOT

3.1. Design Patterns for building app

4. What ARE

4.1. Rules

4.2. Guidelines

4.3. Principles

5. Scope

5.1. Clear and Clean architecture

5.2. Automatable Deployment

5.3. Portability

5.4. Scalability

5.5. Maintenability

6. Origin

6.1. Published when there was no microservices or containers

6.2. Heroku foundation