
# Cloud Native Design

## Includes 12 Factor Apps

# 12-Factor Application

- http://12factor.net
- Outlines architectural principles and patterns for modern apps
  - Clean contract between applications and the platform they run on
  - Focus on scalability, continuous delivery, portability and cloud readiness
- Most of these principles are built in to the Cloud Foundry platform...

# 12-Factor Application

## I. Codebase
One codebase tracked in SCM, many deploys

## II. Dependencies
Explicitly declare and isolate dependencies

## III. Configuration
Store config in the environment

## IV. Backing Services
Treat backing services as attached resources

## V. Build, Release, Run
Strictly separate build and run stages

## VI. Processes
Execute app as stateless processes

## VII. Port binding
Export services via port binding

## VIII. Concurrency
Scale out via the process model

## IX. Disposability
Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity
Keep dev, staging, prod as similar as possible

## XI. Logs
Treat logs as event streams

## XII. Admin processes
Run admin / mgmt tasks as one-off processes

# I. Codebase

- An application has a single codebase
  - Multiple codebases = distributed system (not an app)
- Tracked in version control
  - Git, Subversion, Mercurial, etc.
- Multiple deployments
  - Development, testing, staging, production, etc.
  - Don't hardcode anything that varies with deployment

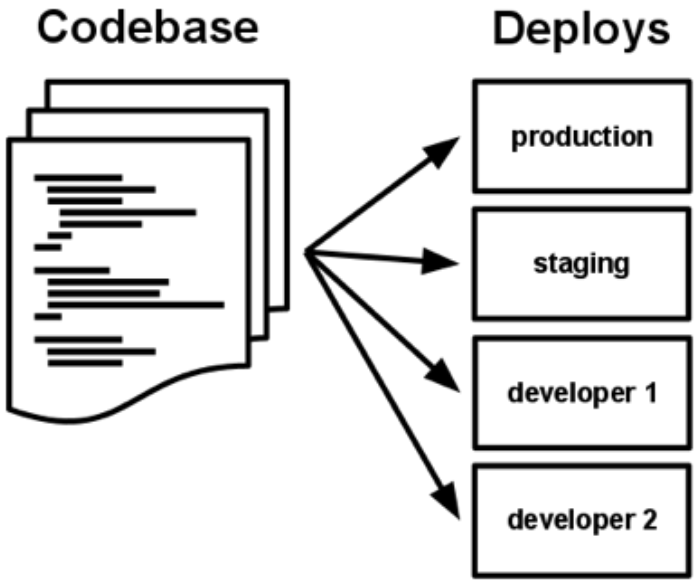


image from http://12-factor.net/codebase

# II. Dependencies

- Explicitly declare and isolate dependencies
  - Dependencies declared in a manifest
    - Maven POM, Gemfile, etc.
  - Use a dependency isolation tool (e.g. bundle exec) to ensure that dependencies don't leak in from the system
  - The dependency management approach is applied uniformly to development and production
  - Results in running the application after a deterministic *build* command

# III. Configuration

- Store config in the environment
  - Anything that varies by deployment should not be included in the code
    - Additionally consider separating config (e.g. database URL or feature flag) from credentials
  - Environment variables or configuration server recommended

# IV. Backing Services

- Treat backing services as attached resources
  - Service are consumed by the application
    - Database, message queues, SMTP servers
  - May be locally managed or third-party managed
  - Connected to via URL / configuration
  - Swappable (change MySQL to an in-memory database)
    - The app and backing services are loosely coupled

# V. Build, Release, Run

- Strictly separate build and run stages
  - Build stage: converts codebase into build (version)
    - Including managed dependencies
  - Release stage: build + config = release
  - Run: Runs app in execution environment
- In Cloud Foundry, these stages are clearly separated with **`cf push`**
  - Developer executes a build
  - Staging combines the build and config to create a droplet
  - Droplets are copied to a container and run

# VI. Processes

- Execute app as stateless processes
  - Stateless
    - Processes should not store internal state
    - Any necessary state is externalized as a backing service
  - Share nothing
    - Data needing to be shared should be persisted
      - “Sticky sessions” violate 12-factor methodology
      - Consider using Gemfire cache or Redis key-value store
    - Use local memory or the local filesystem only as a single transaction “scratchpad”
      - Use storage as a service if needed (Amazon S3, MongoDB)

# VII. Port Binding

- Export that app’s services via port binding
  - Apps are exposed via port binding (including HTTP)
    - Every app instance is accessed via a URI and port number
  - One app can become another app's service

# VIII. Concurrency

- Achieve concurrency by scaling out horizontally
  - Scale by adding more app instances
- Individual processes are free to multithread

# IX. Disposability

- Maximize robustness with fast startup and graceful shutdown
- Processes should be disposable
  - Remember, they're stateless!
- Should be quick to start
  - Enhances scalability and fault tolerance
  - Apps in containers start very quickly
- Should exit gracefully / finish current requests
  - Apps should be architected to handle unexpected terminations

# X. Development/Production Parity

- Keep development, staging and production as similar as possible
  - This enables high quality, continuous delivery
  - Use common tools and a clear separation of concerns
    - Application vs. operating environment/platform
    - Dependency management
    - Build, compile, release
    - Code, configuration, credentials
- Use the same services in development and production
- Minimize surprises in production

# XI. Logs

- App logs are streams of aggregated, time-ordered events
  - Apps are not concerned with log management
    - Just write to stdout or stderr
    - Do not write to logfiles
  - Separate log managers handle management, debugging, analytics, monitoring, etc.
    - Papertrail, Splunk …

# XII. Admin Processes

- Admin processes / management tasks run as one-off processes
  - Applies to developer admin or maintenance tasks like database migrations, clean up scripts, etc.
  - Run admin processes on the platform
    - Leverages platform knowledge and benefits
    - Use the same environment, tools, language as application processes
    - Admin code ships with the application code to avoid synchronization issues

# 12-Factor Application

## I. Codebase

One codebase tracked in SCM, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Configuration

Store config in the environment

## IV. Backing Services

Treat backing services as attached resources

## V. Build, Release, Run

Strictly separate build and run stages

## VI. Processes

Execute app as stateless processes

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity

Keep dev, staging, prod as similar as possible

## XI. Logs

Treat logs as event streams

## XII. Admin processes

Run admin / mgmt tasks as one-off processes