Agenda

The Twelve-Factor App

- What is Twelve-Factor App?
- The Twelve-Factors
- Benefits of Twelve-Factor App

What is Twelve-Factor App?

- The Twelve-Factor App methodology was created by Adam Wiggins and engineers at Heroku while building Heroku platform.
- First presented by Adam Wiggins in 2011.
- Later due to their generic and platform independent implementation they are released as fundamental guidelines for any cloud ready application.
- Include defined practices around version control, environment configuration, isolated dependencies, executing apps as stateless resources, working with backing services like database, queue, and much more.

The Twelve-Factors

I. Codebase

II. Dependencies

III. Config

IV. Backing services

V. Build, Release run

VI. Processes

VII. Port binding

VIII. Concurrency

IX. Disposability

X. Dev/prod parity

XI. Logs

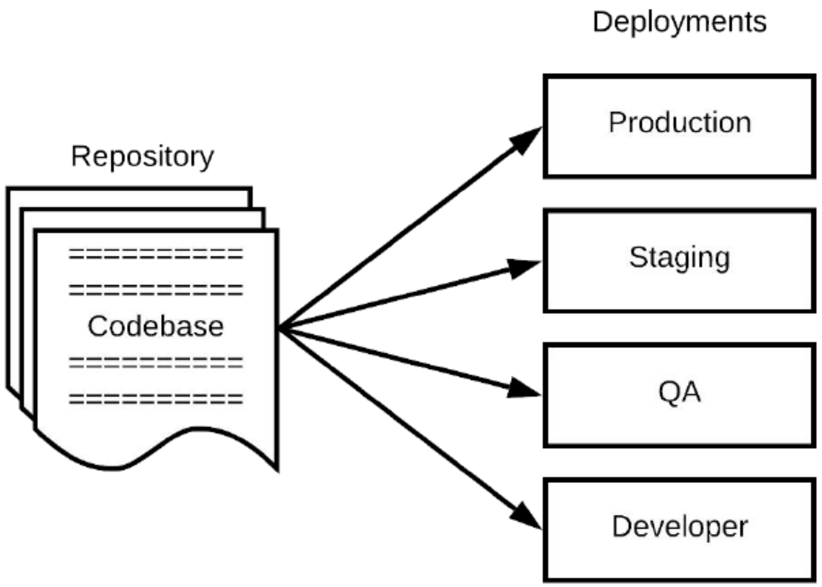
XII. Admin processes

1. Codebase

One Codebase Many Deploy

Codebase

- An app codebase must be stored in a repository managed by a VCS such as Git.
- Must be 1-to-1 correlation between the codebase and the app.
- A distributed app can have multiple codebases, one for each distributed module.
- Multiple apps cannot share code. Such code must be factored out as shared libraries



2. Dependencies

Explicitly Declare and Isolate

Dependencies

- Declare dependencies along with their specific versions (if required) in a manifest like package.json in npm.
- Use dependency isolation tool to prevent accidental import of unwanted dependencies.
- Use dependency manager i.e. packaging system like nuget or npm to fetch all required dependencies from their sources and maintain them in a local repository.

3. Config

Store Config in the Environment

Config

- A 12-factor app requires strict separation of config and code.
- Config is not checked into the app’s repository.
- A config contains secrets such as passwords, or db connection strings
- config should be stored in environment variables.
- App packaging, containerization runtimes, and orchestration systems provides facility to define config for the app thru environment variables based on the deploy type.

4. Backing Services

Treat backing services as attached resources

Backing Services

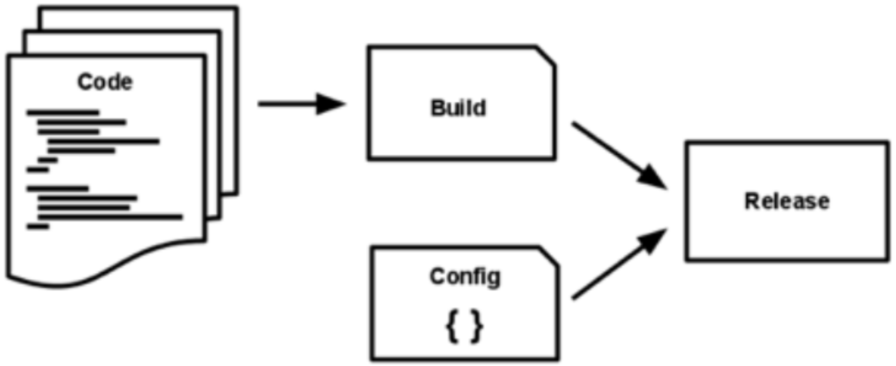
- Backing services are treated as attached resources, whether they are locally managed or third party services.
- They can be accessed easily via a URL or other credentials, and even they can swap to each other.
- Backing Services Examples are:
  - Data Store
  - SMTP
  - Caching Systems
  - Azure Storage

5. Build, Release and Run

Strictly separate build and run stages

Build, Release and Run

- BUILD = codebase + dependencies + assets
- RELEASE = BUILD + config
- RUN = run process against RELEASE
- There must be a separate pipeline for build and release process.
- The “run” can run multiple times from the same “release” on the environment.
- Also, the “release” can run multiple times from the same “build” with different configs.



6. Processes

Execute the app as one or more stateless processes

Processes

- An app process(es) must be designed to run stateless and share nothing.
- Any state that requires persistent must be handled by the backing services (e.g. database).
- The idea of stateless services, help us to scale them by creating multiple instances.
- App should save the sessions in a database rather than holding it in its memory.

7. Port Binding

Export services via port binding

Port Binding

- An app should be fully self-contained; means it does not requires runtime injection of a web server/container.
- An app should only bind to a TCP/UDP port rather than the complete address set i.e. IP address and TCP/UDP port.
- An app port binding should be configurable; not hard coded in the codebase.
- Apps with port binding bring flexibility of getting run within the same environment where all the other processes are bind to different unique ports.

8. Concurrency

Scale out via the process model

Concurrency

- To ensure the scalability of an app, more copies of the app (processes) should be deployed rather than making the app larger.
- The share-nothing, horizontal partitioning nature of twelve-factor app processes means that adding more concurrency is a simple and reliable operation.
- Tools such as Kubernetes can really help you here.

9. Disposability

Maximize robustness with fast startup and graceful shutdown

Disposability

- An app process should be design in a way that it can be tear down, terminated, and restart again in moments.
- An app process should have minimum startup time.
- An app process should shut down gracefully on terminate signal.
- An app process should also be robust against sudden failure, and should be architect to handle unexpected, non-graceful termination without losing the in progress workload requests.

10. Dev/Prod Parity

Keep development, staging, and production as similar as possible

Dev/Prod Parity

- An app's engineering process should be design to support CI/CD.
- The engineering process should minimize the dev and prod gap.
- Keeping dev, staging and prod similar will ensure anyone can understand it and provide releases.
- This ensures great development with limited errors, and also enables better scalability.

11. Logs

Treat logs as event streams

Logs

- Logs provide visibility into the behavior of a running app.
- An app should write its logs to its output stream that should be configurable from the environment.
- Don’t route or store logs in files.
- Use Splunk or Logstash/ELK Stack for logging.

12. Admin Processes

Run admin/management tasks as one-off processes

Admin Processes

- An app often comes with various one-off administrative processes for maintenance tasks like cleaning temporary or unused or malformed data etc.
- Admin tasks should run as separate process(es) against the same release. Hence any failure either in admin process or app's own process do not impact each other.
- Admin code must ship with application code to avoid sync issues.

Benefits of Twelve-Factor App

- Use declarative formats for setup automation. This minimizes the time and cost for new developers joining the project
- Have a clean contract with the underlying operating system, offering maximum portability between execution environments
- Suitable for deployment on modern cloud platforms, thus removing the need for servers and systems administration
- Limits differences between development and production, enabling continuous deployment for maximum agility
- Can scale up without any major changes to tooling, architecture, or development practices, hence performance is a priority