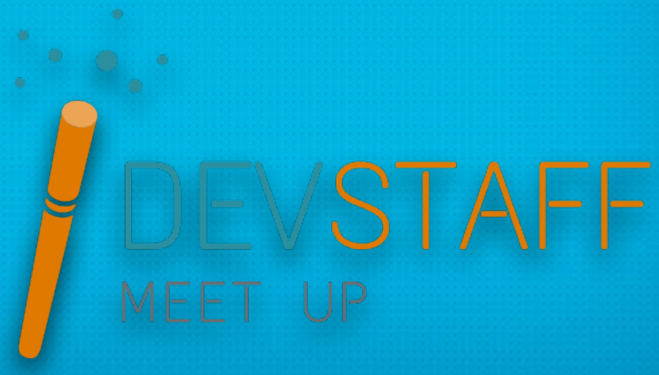
BUILDING

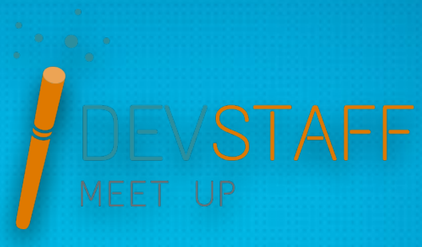
SCALABLE + MAINTAINABLE + RELIABLE

APPS

12-factors to consider...



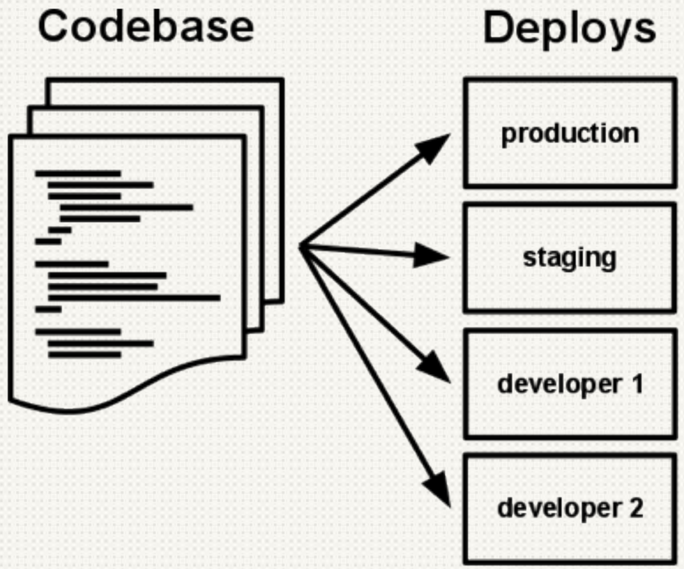
12FACTOR.NET



CODEBASE

I. CODEBASE

- 1 codebase, kept in a Version control system
- 1 codebase == 1 app. No cheating.
- Shared code is in libraries.
- Library sources are NOT included in app sources.
- One codebase can have many deploys (dev, test, staging, prod)



DEPENDENCIES

II. DEPENDENCY DECLARATION

- never rely on implicit existence of system-wide packages
  - e.g. curl does *not* exist everywhere your app will run!! (yes, docker, i'm talking to you)
- "declare all dependencies, completely and exactly, via a dependency declaration manifest"
  - completely: don't leave anything out. (this is not as easy as it sounds)
  - exactly: make sure to specify versions, or version upgrade policy
  - optimistic `>=` vs. pessimistic `~>` version constraints (semantic versioning)

II. DEPENDENCY ISOLATION

- Use some environment that allows you to ensure you are loading what you've declared.
  - And only that!
  - if your stack doesn't provide it, just start with as empty a box as you can find
- Examples:
  - Ruby: gemfile + bundle exec
  - Python: pip + virtualenv
  - Java: classpath

CONFIG

III. CONFIG

- IS: Everything that varies between deploys.
- (db url, credentials, hostnames, etc.)
- NOT constants in code. (Config varies substantially across deploys, code does not.)
- Avoid config files.
- All config in env vars.

BACKING SERVICES

IV. BACKING SERVICES

- Database. Obviously.
- Message queue (rabbitmq, SQS, redis, etc.)
- SMTP client (postfix)
- Cache (memcached, redis)
- Metrics gathering (new relic, loggly)
- Binary object stores (Amazon S3)

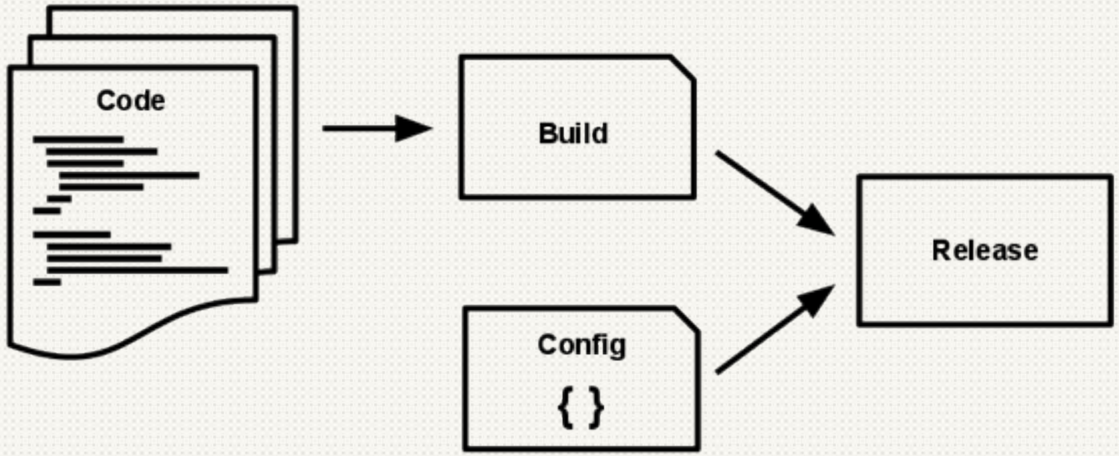
IV. BACKING SERVICES

- Make no distinction between local and third-party services
  - `localhost` shouldn't make a difference to `some_fqdn`
- Should be easy to swap simply by changing resource handle (URL) and credentials.
  - e.g. locally hosted mysql / postgresql <==> AWS RDS
- Attachable resources: i am connecting to some db, when that is available.

BUILD – RELEASE – RUN

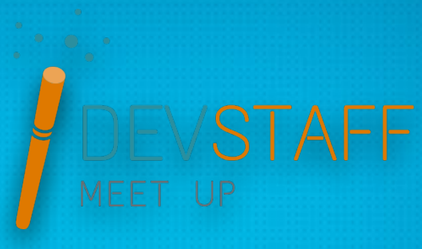
V. BUILD – RELEASE – RUN

- Strictly separate the below 3 build and run stages:
- build: create executable bundle, using some version, fetching all dependencies and compiling (where necessary).
- release: add config to bundle, to have something that's ready to run.
- run: (a.k.a. runtime) launch some process(es) that run the release.



V. BUILD – RELEASE – RUN

* No changes to code at runtime.
* Use deployment tools for release management:
    * deploy automatically,
    * but also **rollback** a release.
* Releases are immutable.
    * Yes, this means you are not allowed to manually make changes to production!
* Releases should have some unique ID
    * timestamp, number increment, hash, etc.



PROCESSES

VI. PROCESSES

- ☑ Execute the app as one or more stateless processes.
- ☑ Twelve-factor processes are stateless and use a shared-nothing architecture.
- ☑ Data is persisted in stateful backing services, typically a database.
  - ☑ Use memory or local filesystem ONLY as a brief, single-transaction cache.
  - ☑ Don’t expect the filesystem or memory to be the same.

PORT BINDING

VII. PORT BINDING

- Export services via port binding
- Self-contained services that listen to a specific port.
- webserver library is bundled into the app. Do not rely on server existing in system.

CONCURRENCY

VIII. CONCURRENCY

- Separate Workload type <–> process type.
  - (e.g. HTTP requests <–> web process, long-running background tasks <–> worker process.
- Many workloads, but also many processes of the same workload!
- Rely on some process manager or scheduler to handle lifecycle of your processes (don’t daemonize!)

DISPOSABILITY

IX. DISPOSABILITY

- Process must start (and STOP!!) fast!
- Minimize startup time (a few seconds!)
- Shutdown gracefully (on SIGTERM)
- Protect your state against sudden death.

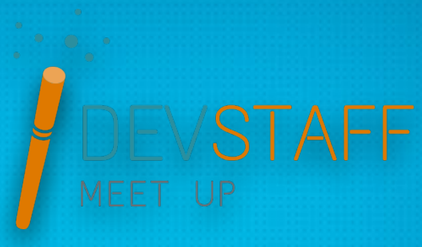
DEV / PROD PARITY

X. DEV/PROD PARITY

	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

X. DEV/PROD PARITY

- Keep development, staging, and production as similar as possible
- Use same backing services (same type of) between development and production
  - e.g. using in-memory dbs is an anti-pattern
- Design for continuous deployment



LOGS

XI. LOGS

- Use logs as event streams
  - a stream of aggregated, time-ordered events from all running processes and backing services.
- Not concerned with routing or storage of output stream.
  - Do not attempt to write to or manage logfiles (e.g. rotate). Instead, each running process writes its event stream, unbuffered, to stdout.
- In production, a log router / forwarder will take care of sending events from log stream to a centralized logging facility for long-term storage.
- where... find events (fast!), spot patterns (graphs over time), alerting!

ADMIN PROCESSES

XII. ADMIN PROCESSES

- one-off tasks:
  - in an identical environment as long-running processes,
  - running against a release,
  - using same codebase and
  - using same config.
- generally shipped together with production code, using same dependency isolation.
  - they become part of the release


That's all Folks!

THANK YOU!

Yorgos Saslis
@gsaslis
github.com/gsaslis