Some Advice and Ideas

Dev-Ops

Cliff McCollum <cliffmcc@gmail.com>

Dev-Ops

What?   Why?

What is Dev-Ops?

- “**DevOps** (a clipped [illegible] operations) is a cult[illegible]e, move[illegible] emphasizes the co[illegible]boration[illegible] both software deve[illegible]opers and technology (IT) pro[illegible]essionals [illegible] process of softwar[illegible] delivery a[illegible] changes.”

What **REALLY** is Dev-Op

- You write it.
- You deploy it.
- You run it.
- You support it.
- ***You*** get paged in the middle of the night.

Why would I want that?

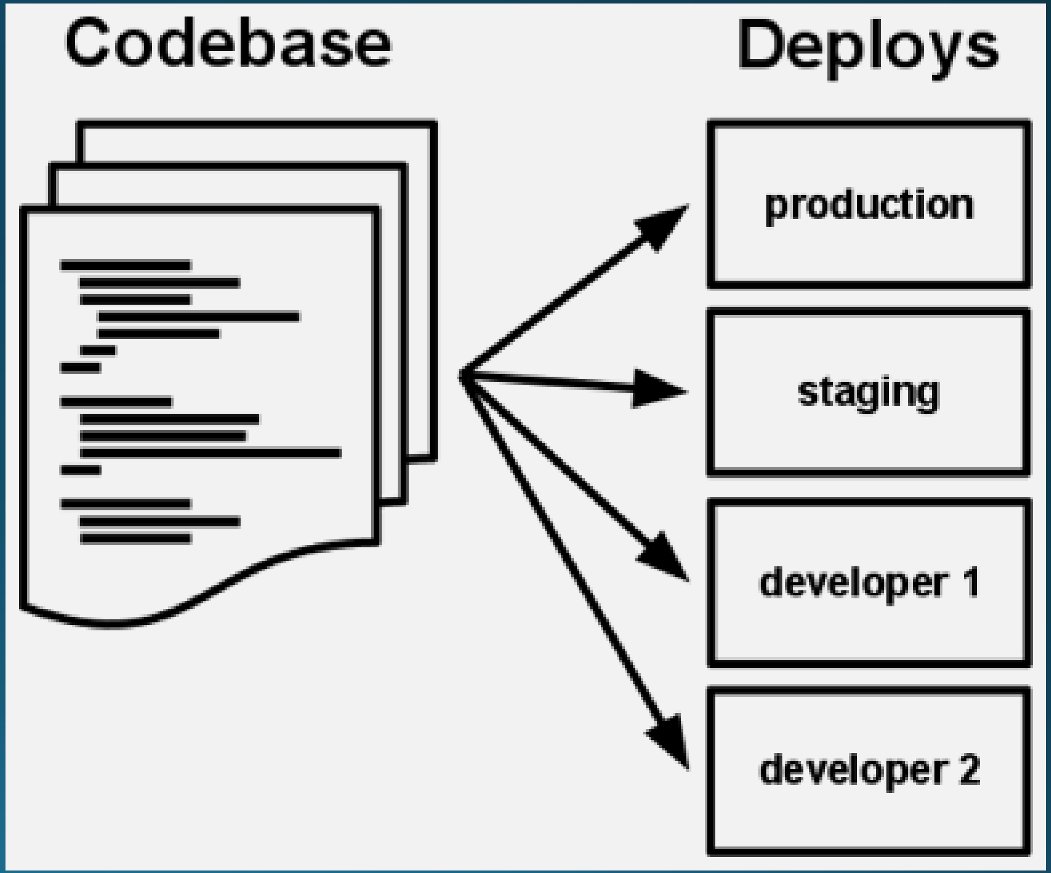
- Greater Ownership
- Better code
- Faster bug-fixes
- Quicker Deploys
- Happier Customers

One Model

12 Factor App

One Codebase, in revision control, many deploys

- If there are multiple codebases, it’s not an app – it’s a distributed system.
- Multiple apps sharing the same code is a violation of twelve-factor.



Explicitly declare and isolate dependencies

- A twelve-factor app never relies on implicit existence of system-wide packages.
- Declare all dependencies, completely and exactly, via a dependency declaration manifest.

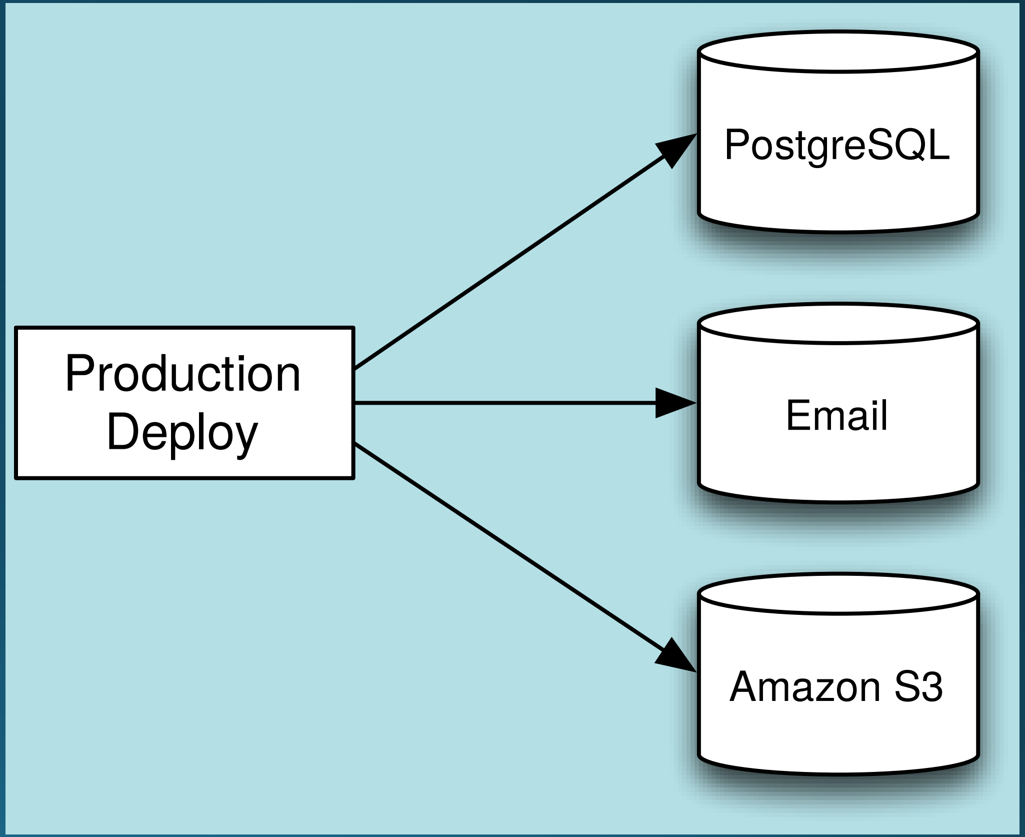
```
arrow==0.5.4
astroid==1.1.0
awsshell==0.0.1
awscli==1.9.15
bcdoc==0.15.0
boto==2.36.0
boto3==1.2.3
botocore==1.3.15
colorama==0.3.3
configobj==5.0.6
docutils==0.12
futures==3.0.3
```

Store config in the environment

- Never store App config as constants in the code.
- Does not include internal app config (like URL enpoint lists).
- Could your code be made open-source without compromising any credentials?
- Store config in environment variables.

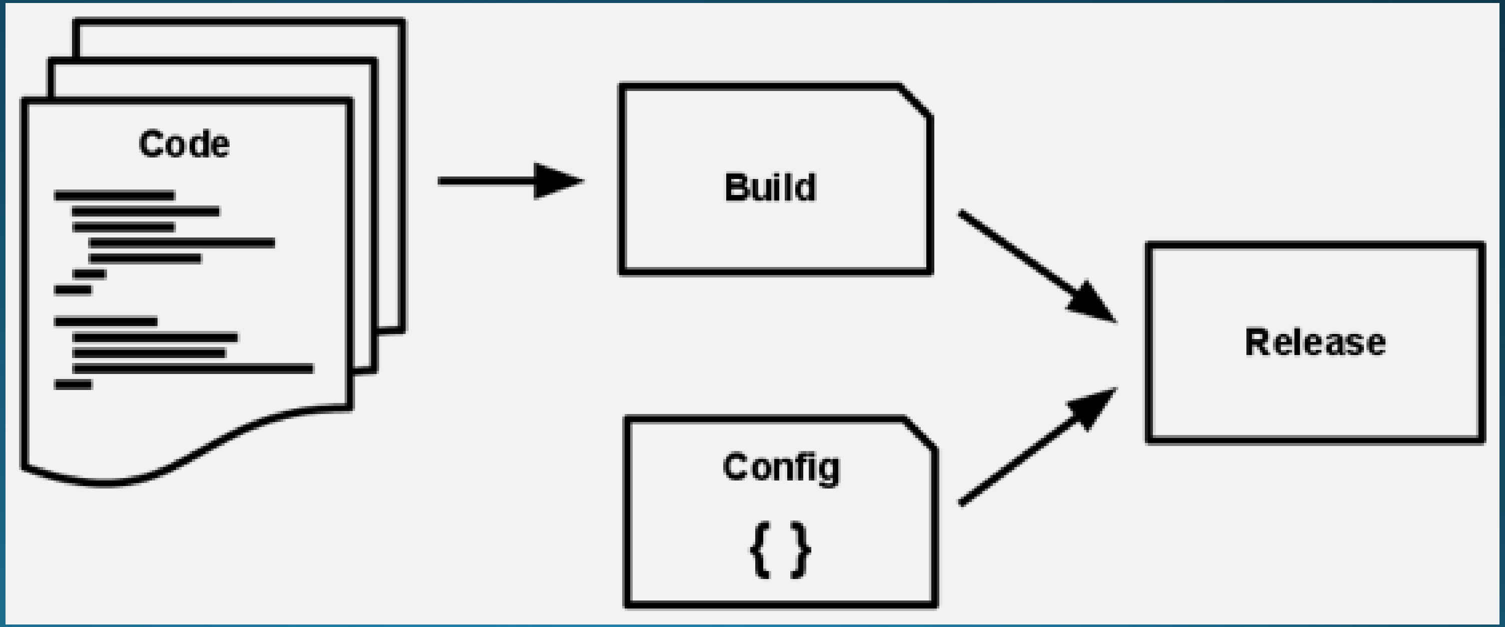
Treat backing services as attached resources

- A backing service is any service the app consumes over the network.
- Make no distinction between local and third party services.



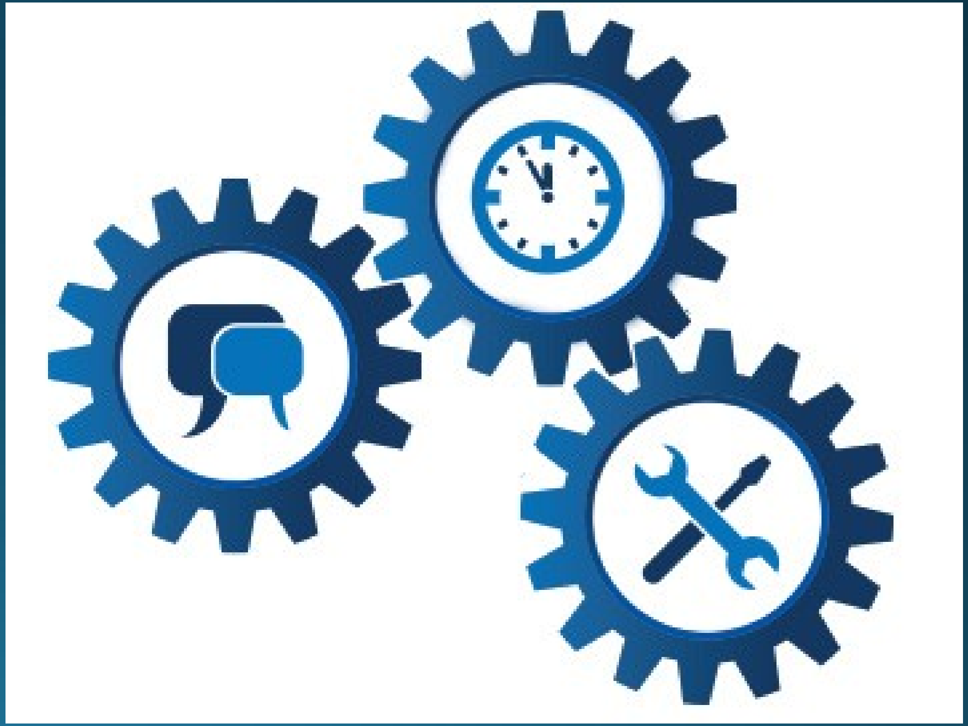
Strictly separate build, release, and run stages

- Every change creates a new release.
- Every release has a unique release ID.
- Developers trigger build and release. Run is triggered automatically.



Execute the app as one or more stateless processes

- Twelve-factor processes are stateless and share-nothing.
- Persistence must be stored in a stateful backing service (eg DB).
- Never rely on “sticky” sessions.
- Consider memcached or Redis.

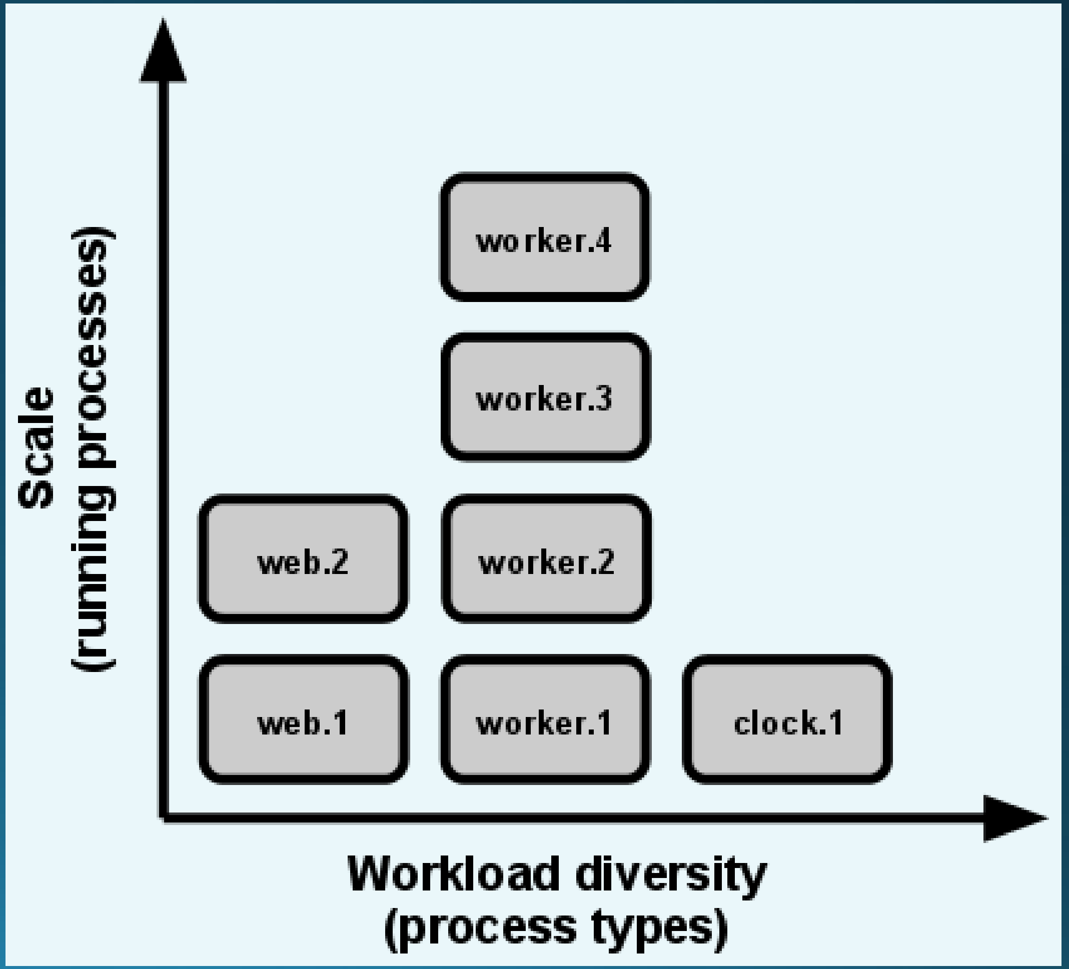


Export services via port binding

- Make your app completely self-contained.
- Build services as a process, binding on a port.
- Encourages SOA.
- Use in-language servers like Tornado, Thin, or Jetty.

Scale out via the process model

- Processes are a first class citizen.
- Never daemonize or write PID files.
- Use a process manager like Supervisor, Upstart, etc.



Make servers and processes disposable

- PROCESSES
  - App processes can be started or stopped at a moment’s notice.
  - Minimize startup time. Shutdown quickly to SIGTERM.
  - Ultimate version: Crash-Only Design.
- SERVERS
  - There should be no state on disk.
  - Don’t patch – just rebuild.

Dev / Prod parity

- Keep development, staging, and production as similar as possible
- Watch for “secret gaps”:
  - Personnel
  - Configuration
  - Time / Versions
  - Tools
  - Deploys
- Same backing services in all cases.

Treat logs as event streams

- Logs are a stream of all events from all processes and backing services.
- An app should never concerns itself with routing or storage of its output stream. Just use STDOUT.

Run admin/management tasks as one-off processes

- Run admin processes in the same environment as the regular app processes.
- Ship Admin code with the App.
- Use same isolation method as App (VirtualEnv, etc).

The 13th Rule: Infrastructure as Code

- Don’t rely on infrastructure you can’t specify in code.
- Check your infrastructure into your code-repository.
- Resist the urge to make manual changes to your infrastructure.
- You can’t ensure Dev/Prod parity without it.