

# The Twelve-Factor App



The twelve-factor app is a methodology for building software-as-a-service apps that:

Use declarative formats for setup automation, to minimize time and cost for new developers joining the project.

Have a clean contract with the underlying operating system, offering maximum portability between execution environments.

Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration.

Minimize divergence between development and production, enabling continuous deployment for maximum agility.

And can scale up without significant changes to tooling,  
architecture, or development practices.





# I. Codebase

- One codebase tracked in revision control, many deploys.
- If there are multiple codebases, it's not an app – it's a distributed system.

# II. Dependencies

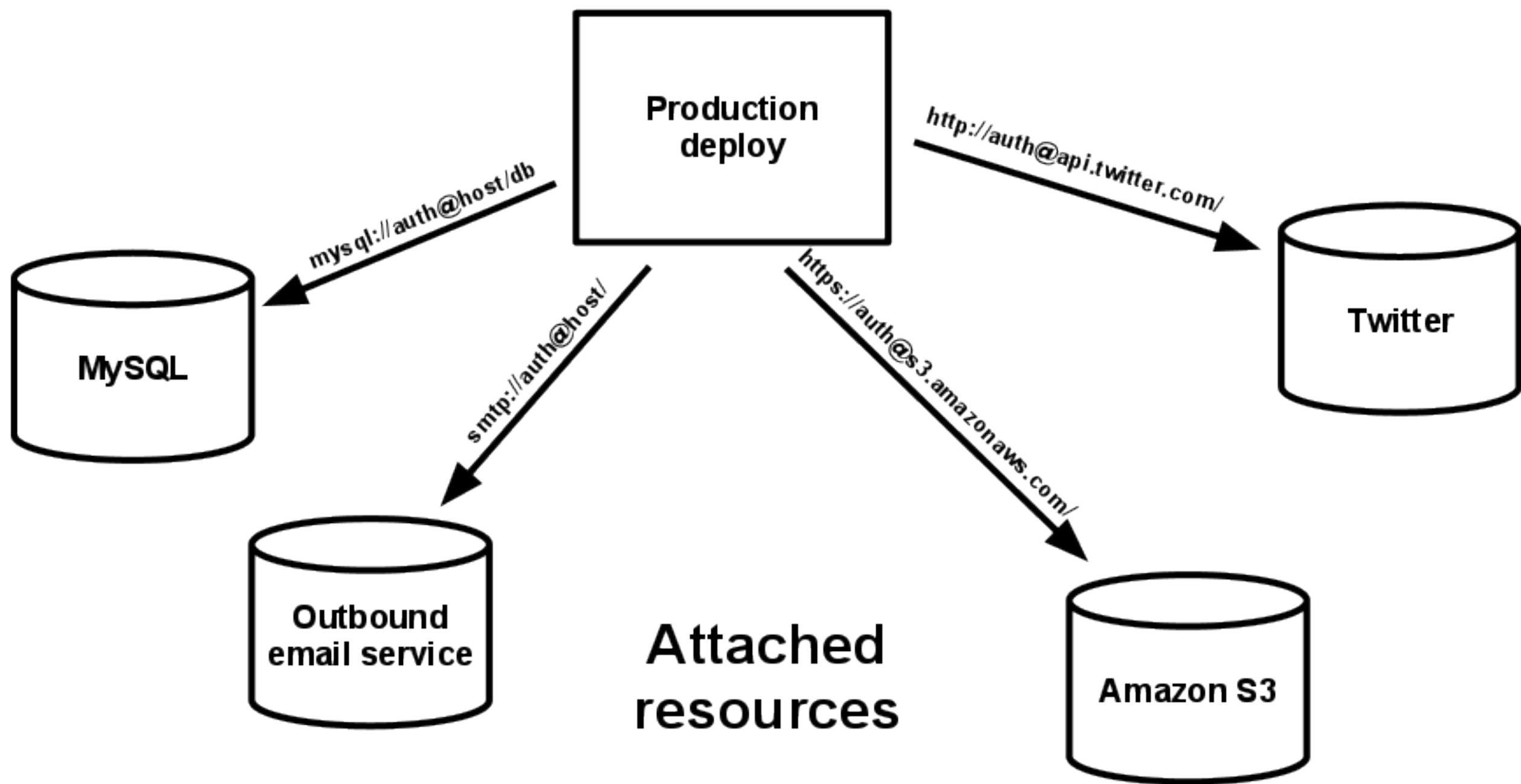
- Explicitly declare and isolate dependencies.
- A twelve-factor app never relies on implicit existence of system-wide packages.
- Do not rely on the implicit existence of any system tools.
- `pip + virtualenv + requirements.txt`

# III. Config

- Store config in the environment.
- Resource handles to the database, Memcached, and other backing services.
- Per-deploy values such as the canonical hostname for the deploy.
- Django & Flask make this simple.

# IV. Backing Services

- Treat backing services as attached resources.
- Make no distinction between local and third party services.



# V. Build, release, run.

- Strictly separate build and run stages.
- It is impossible to make changes to the code at runtime.
- This allows for rollbacks and other release management suites.

# VI. Processes

- Execute the app as one or more stateless processes.
- `$ python app.py`
- A production deploy of a sophisticated app may use many process types, instantiated into zero or more running processes.

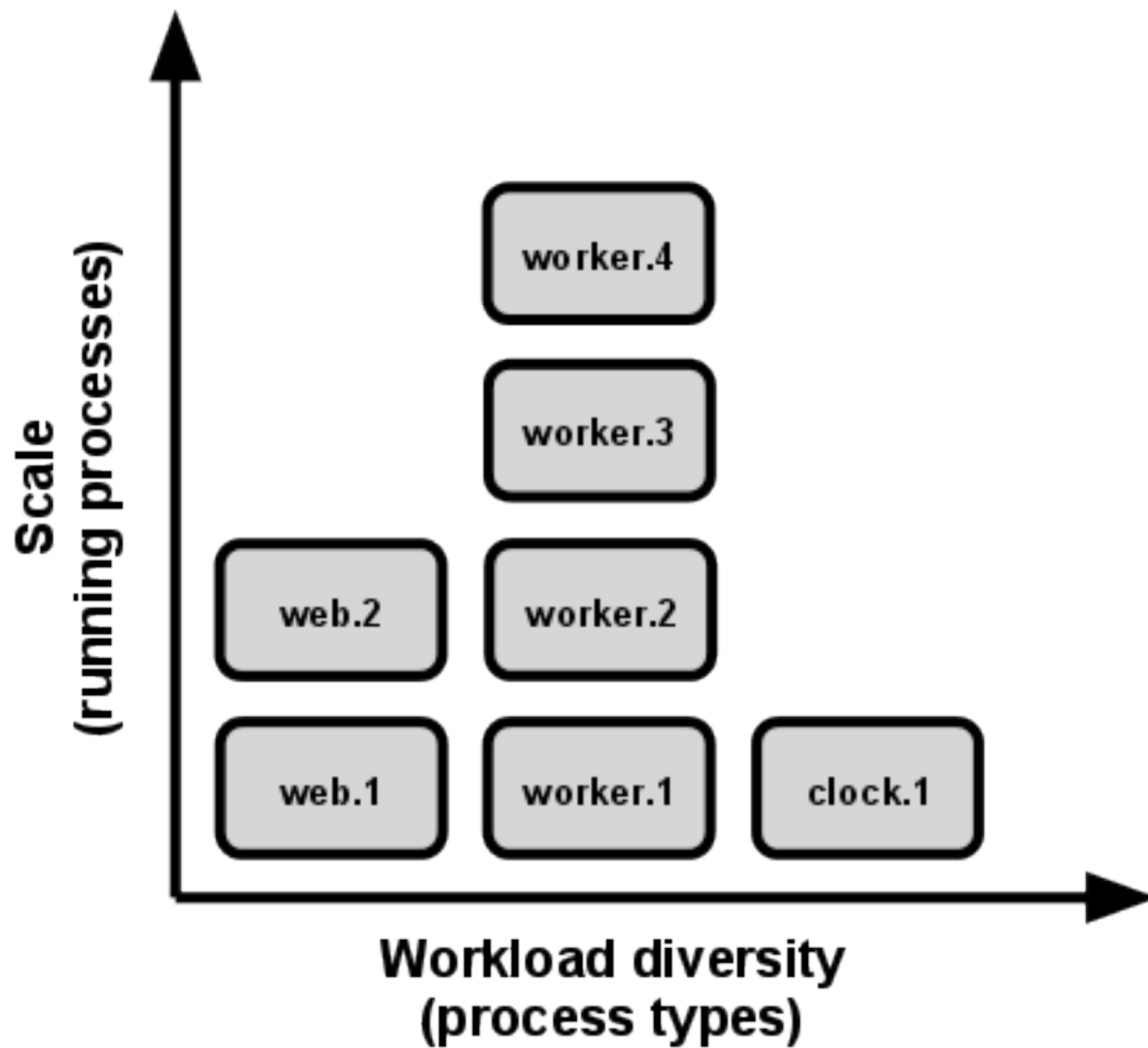
# VII. Port binding

- Export services via port binding.
- The web app exports HTTP as a service by binding to a port, and listening to requests coming in on that port.
- Gunicorn, Gevent, Eventlet.



# VIII. Concurrency

- Scale out via the process model.
- Using this model, the developer can architect their app to handle diverse workloads by assigning each type of work to a process type.
- Rely on the operating system's process manager.



# IX. Disposability

- Maximize robustness with fast startup and graceful shutdown.
- They can be started or stopped a moment's notice.

# X. Dev/prod parity

- Keep development, staging, and production as similar as possible.
- Failing to do so increases: the time gap, personnel gap, and the tools gap.

	<b>Traditional app</b>	<b>Twelve-factor app</b>
<b>Time between deploys</b>	Weeks	Hours
<b>Code authors vs code deployers</b>	Different people	Same people
<b>Dev vs production environments</b>	Divergent	As similar as possible

# XI. Logs

- Treat logs as event streams.
- Apps never concern themselves with routing or storage of the output stream.

# XII. Admin processes

- Run admin/management tasks as one-off processes in an identical environment.
- Run against a release, using the same code and config as any process run against that release.
- `$ manage.py syncdb`

12factor.net

