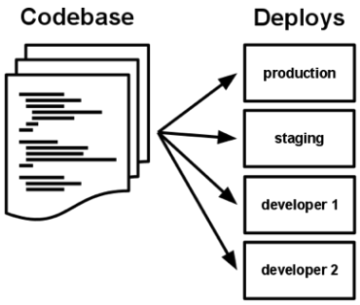
The Twelve Factor App

In the modern era, software is commonly delivered as a service: called web apps, or software-as-a-service. The twelve-factor app is a methodology for building **software-as-a-service** apps.

1. Use declarative formats for setup automation, to minimize time and cost for new developers joining the project
2. Have a clean contract with the underlying operating system, offering maximum portability between execution environments
3. Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration
4. Minimize divergence between development and production, enabling continuous deployment for maximum agility
5. And can scale up without significant changes to tooling, architecture, or development practices.

Twelve Factors are as below:



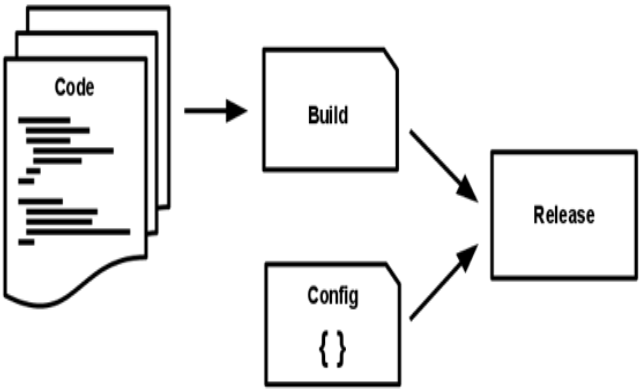
- **Codebase:** One codebase tracked in revision control, many deploys. A codebase is any single repo (in a centralized revision control system like Subversion), or any set of repos who share a root commit (in a decentralized revision control system like Git).

- **Dependencies:** Explicitly declare and isolate dependencies. A twelve-factor app never relies on implicit existence of system-wide packages. It declares all dependencies, completely and exactly, via a dependency declaration manifest. Furthermore, it uses a dependency isolation tool during execution to ensure that no implicit dependencies "leak in" from the surrounding system. The full and explicit dependency specification is applied uniformly to both production and development.

- **Config :** Store config in the environment. An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:
  - Resource handles to the database, Memcached, and other backing services
  - Credentials to external services such as Amazon S3 or Twitter
  - Per-deploy values such as the canonical hostname for the deploy

- **Backing services**: Treat backing services as attached resources. A backing service is any service the app consumes over the network as part of its normal operation. Examples include datastores (such as MySQL or CouchDB), messaging/queueing systems (such as RabbitMQ or Beanstalkd), SMTP services for outbound email (such as Postfix), and caching systems (such as Memcached).

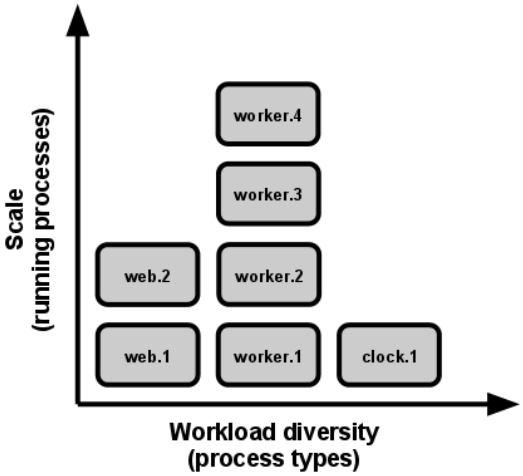
- **Build, release, run**: Strictly separate build and run stages.
  A codebase is transformed into a (non-development) deploy through three stages:
  - The build stage is a transform which converts a code repo into an executable bundle known as a build. Using a version of the code at a commit specified by the deployment process, the build stage fetches vendors dependencies and compiles binaries and assets.
  - The release stage takes the build produced by the build stage and combines it with the deploy's current config. The resulting release contains both the build and the config and is ready for immediate execution in the execution environment.
  - The run stage (also known as "runtime") runs the app in the execution environment, by launching some set of the app's processes against a selected release.



- **Processes:** Execute the app as one or more stateless processes. **Twelve-factor processes are stateless and share-nothing**. Any data that needs to persist must be stored in a stateful backing service, typically a database.

- **Port binding**: Export services via port binding. The twelve-factor app is completely self-contained and does not rely on runtime injection of a webserver into the execution environment to create a web-facing service. The web app exports HTTP as a service by binding to a port, and listening to requests coming in on that port.

- **Concurrency**: Scale out via the process model.
  In the twelve-factor app, processes are a first class citizen. Processes in the twelve-factor app take strong cues from the unix process model for running service daemons. Using this model, the developer can architect their app to handle diverse workloads by assigning each type of work to a process type. For example, HTTP requests may be handled by a web process, and long running background tasks handled by a worker process.



- **Disposability:** Maximize robustness with fast startup and graceful shutdown. The twelve-factor app's processes are disposable, meaning they can be started or stopped at a moment's notice. This facilitates fast elastic scaling, rapid deployment of code or config changes, and robustness of production deploys.

- **Dev/prod parity**: Keep development, staging, and production as similar as possible.

- **Logs**: Treat logs as event streams. A twelve-factor app never concerns itself with routing or storage of its output stream. It should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout. During local development, the developer will view this stream in the foreground of their terminal to observe the app's behavior.

- **Admin processes:** Run admin/management tasks as one-off processes. Twelve-factor strongly favors languages which provide a REPL shell out of the box, and which make it easy to run one-off scripts. In a local deploy, developers invoke one-off admin processes by a direct shell command inside the app's checkout directory. In a production deploy, developers can use ssh or other remote command execution mechanism provided by that deploy's execution environment to run such a process.