Factor 5 - Build, release, run: Strictly separate build and run stages

In a 12-factor app, the code is transformed during its deployment process by using three stages that are strictly separated:

- The build stage converts a code repository into an executable bundle that includes the code and all its dependencies. This bundle is known as a *build*.
- The release stage takes the build and combines it with the deployments's current environment configuration, which is described in "**Factor 3 - Configuration data: Store environment-specific configurations in environment variables**" on page 101. The resulting release is ready for immediate execution in the execution environment.
- The run stage starts the app in the execution environment.

The most important separation is between build and run to ensure that it is impossible to make changes to the code at run time and simplify rollback if it is required.

In container technology that uses modern language run times, the build stage can be as simple as placing product and language files on to the file system and creating from them a container image. That image is then the immutable executable that is deployed to each environment.

Factor 6 - Processes: Run the app as one or more stateless processes

This factor is really about the statelessness of the application process. Twelve-factor should not assume that any state is held between invocations. Any data that might be needed in the future must be stored in a stateful backing service such as a database.

There can be many instances where a process running for scaling has a high chance that a future request will be served by a different process. Even when running only one process, a restart (triggered by code deployment, a configuration change, or the execution environment relocating the process to a different physical location) wipes all local (for example, memory and file system) states.

You can see that this statelessness is embedded into the container model. Containers are ephemeral: When they are restarted, a new container is created, and all local memory and files are lost. We see the importance of this point when we talk about scaling in "**Factor 8 - Concurrency: Scale out by using the process model**".

Factor 7 - Port binding: Export services through port binding

Your application process should expose its function only over a defined URL scheme that is bound to a port, which commonly means HTTP-based APIs that are based on the OpenAPI specification, but other protocols also can be used. Any libraries that are required to perform this exposure should be included as part of the application.

The standardized port-binding approach means that one app can become a resource (see "**Factor 4 - Backing services: Treat backing services as remotely attached resources**" on page 101) for another app by providing the URL to the backing app as an environment variable (see "**Factor 3 - Configuration data: Store environment-specific configurations in environment variables**" on page 101) for the consuming app.

Factor 8 - Concurrency: Scale out by using the process model

Because you build our application as a stateless process (see "**Factor 6 - Processes: Run the app as one or more stateless processes**" on page 102), you can defer its operational management to the surrounding platform. The modern parallel here is with containers running as a single process, and container orchestration platforms such as K8s that provide agnostic operational management of the container replicas.