and production ending up side by side in the same Docker image, which requires an image rebuild for every change in either environment. All that shows us that environment variables are suitable for small sets of configurations only.

The patters *Configuration Resource*, *Immutable Configuration*, and *Configuration Template* described in the following chapters are good alternatives when more complex configuration needs come up.

Environment variables are universally applicable, and because of that, we can set them at various levels. This option leads to fragmentation of the configuration definitions and makes it hard to track for a given environment variable where it comes from. When there is no central place where all environments variables are defined, it is hard to debug configuration issues.

Another disadvantage of environment variables is that they can be set only *before* an application starts, and we cannot change them later. On the one hand, it's a drawback that you can't change configuration "hot" during runtime to tune the application. However, many see this as an advantage, as it promotes *immutability* even to the configuration. Immutability here means you throw away the running application container and start a new copy with a modified configuration, very likely with a smooth Deployment strategy like rolling updates. That way, you are always in a defined and well-known configuration state.

Environment variables are simple to use, but are applicable mainly for simple use cases and have limitations for complex configuration requirements. The next patterns show how to overcome those limitations.

More Information

- EnvVar Configuration Example
- The Twelve-Factor App
- Immutable Server
- Spring Boot Profiles for Using Sets of Configuration Values