Being a software engineer and architect I often think of ways to simplify and/or compartmentalize the development efforts of projects/products I work on. This is one of my favorite and most endearing aspects of Symfony that has led me so often to using it for custom application development. (It doesn't hurt that the recent release of PHP7 has been so fundamentally well regarded for its *dramatic* performance improvements).

I've written about this approach before in an article "Planning Your Tech Stack," but this in particular goes into more depth and a particular example use-case using Symfony.

One of the great "things" about Symfony is the concept of a Bundle or Component which can be reused on demand across any application thanks to Composer.

Take for example the setup below.

Here we've architected a `CoreBundle` with components we want to reuse across more than one application. The most obvious being the Entity classes and database handling, so each application shares the same source data. This example application is a basic API with an Admin Module; pretty typical setup for what I help clients with every day.

In this situation, the Api and Admin module want to work with the same data, but have separate authentication mechanisms, so `ApiBundle` and `AdminBundle` each have their own `Security` component that depends on the underlying `Providers` within `CoreBundle`. The API would most likely use JWT or Basic (use JWT, you'll thank me later) authentication, whereas the Admin module we generally recommend HTTP Auth with a traditional username and password.

Other components are shared in similar ways, but ultimately the point is that in a DRY situation you want to avoid repeating yourself, or your code. And furthermore, with each additional app or service that reuses the component, the model validates itself even further.