SymfonyCon 2018 Lisbon Conference Videos

With <3 from SymfonyCasts

Chapter 1: Back to the Basics (Symfony local Web Server)

Tip

SymfonyCon 2018 Presentation by Fabien Potencier

Keynote presentation covering Symfony's local web server.

Okay. Welcome. Good morning. Thank you.

So if you are in the stairs over there, I can see empty seats everywhere here in the front. So if you can come, as you like. Okay. So today's talk is about, it's a series of talks. I'm going to give over the next few conferences. Um, I talked about emails during SymfonyLive London a few months ago and today it's going to be another talk about getting back to the basics. And the topic is a local web server. Are you excited about that? No. Okay. Too bad anyway, Even if you're not excited about that, it's going to be what I'm going to talk about today.

Local Web Server?

Okay. So a few month ago I talked about and, and, and actually I asked people on Twitter about which a web servers that we're using on our local machines, uh, that's the results. So as you can see a lot of people are still using php -S which is a builtin a php web server. Um, and also using the symfony server:start, which is exactly the same because we understand that we are using php -S. Anyway, a bunch of people are using Docker, Nginx, Apache and Vagrant. So the problem with that is that php -S is great. It is very easy. You need to install the WebServerBundle and then you can get started really easily. The problem is that it comes with many limitations and I'm going to talk about that in minute.

Docker, well may be slow depending on the platform, especially on a Mac. I mean it is for me at least. Um, and it's kind of inconvenient and I'm going to talk about that as well. Um, Nginx and Apache, it depends on your skills on those um, uh... programs, but it needs some setup and it's not that easy. It's not really nice if you ask me. The great thing about using Nginx or Apache on your local server, it's probably the same that you're going to have on your production machines. So that's really nice and Vagrant is also not very convenient because it's, you know, something you need to install and then you have some kind of, virtual machines. So. Okay. So that's my setup.

Ideal Local Server Requirements

The setup I would like: something that is really fast and when I say fast, um, I want the web server to be really responsive. So I don't want to wait for something to happen. And convenient. And by convenient, I mean I want my files to be locally on my laptop. I don't want them to be on a Docker share or NFS or whatever. Or on the virtual machine because then it's going to be slow. I don't care about services so I don't care if my MySQL database or Postgres or Redis is on my laptop and actually I don't want to install anything on my laptop. So I like to be able to use remote services, it can be Docker, can be Symfony Cloud, whatever. And I want something that works on all platforms. Not that I have three different laptops, but just because I care about people in the room not using a Mac.

How many of you are using Windows? One, two, three, four, five, six, seven, eight, nine, 10 to 12. Okay. Linux? Much more. And a Mac? About the same. Okay, cool.

So I was talking about the PHP built in server, it's almost perfect, but it has many limitations. So the first one is that it can only do one connection at a time, which is kind of annoying sometimes when you have a bunch of ESI's for instance and things like that, your website can you kind of slow, even locally. No support for HTTP-2, no support for tls, which is kind of a requirement nowadays. And if you are using Symfony 4, with environment variables, you might know that it does not reload environment variables from one request to the next. So if you change your variables then you need to stop and restart your web server. And also it needs some tweaks to make it work with Symfony. So if you are just using php -S, like that is not going to work well. It kind of works, but we have a special router to make it work. That's why we have the WebServerBundle. But it's nice because it's local, it's using my local files. It's really fast and it just works.

Introducing the "better" local web server

So in the last few weeks I've been working on a better local web server and that's what are we going to talk about. And it's available via the the new symfony binary. I'm not sure if you are all aware about that. I Tweeted about that yesterday or earlier today. I don't remember. And we had a blog post about Symfony cloud a few days ago and everything is available via a new

symfony binary. So the new symfony binary is not just about Symfony Cloud, it's about a bunch of things.

symfony security:check

So I talked about two of them last week in my blog post about Symfony Cloud. The first one is security checks. So if you want to check your composer.json, for security issues, you can use security.symfony.com if you want and the API. Or you can use symfony security:check, which is nice because it does not connect to the API. So it's, everything happens locally. So it clones a GitHub repository but then everything happens locally. So that's a nice way to not depend on security.symfony.com.

symfony new

And symfony new is a way to create a new symfony project. It's just a thin wrapper on top of a composer, so you don't have to remind the composer create-project symfony/skeleton thing. You can just use symfony new. By default it creates a project with symfony/skeleton and then you can say I want a demo or a full website which is using the symfony/website-skeleton. And then it does a few other things. The first one is it does git init. It also configure everything for Symfony Cloud if you are interested in that. Okay.

symfony server:start

And of course, as of today, it's also about the symfony local web server. So that's how you can use it. You can just say symfony server:start and it should just work. So it uses the first available port. Instead of 8000 you can configure it if that makes sense for you. And then you have all the logs from PHP and, and a bunch of other things. You can also start it in background with -d and the great thing about that is that then you can just fire and forget. And the great thing - and that's very different from what you get from php -S - that you can just say symfony server:log to tail the logs of the server, which is not possible with php -S. And then you can say symfony open:local to open a website in the browser.

Symfony server Features

Okay, so some of the features from the symfony local web server. The first one is don't need to be root to run it. It runs on a port more than 8000. So that's not a problem.

It uses php-fpm, which is very important because it's more similar to what you would get in production. But it does more than that because I realized while working on that and I wanted to make it work on Windows. And now I realiz it was not really necessary because you are not that many. Um, it took me a lot of time really. But now I have a virtual machine on my laptop with windows, which is kind of nice. I mean, Windows is much nicer than it was 10 years ago, so that's great. But php-fpm does not work or it is not available on windows. So when you don't have php-fpm, it falls back to php-cgi, which is also fastcgi. And then php -S if there is nothing else. It supports HTTP-2, it supports tls, some push support. So if you have some link headers and - the support and we do have support in Symfony - then it's going to work with this web server.

It works with any PHP project, there is no, nothing special. So there's no configuration, nothing. It just works with any PHP framework or CMS's or whatever. You can also use it, use it for a plain HTML websites, if you have a nodejs thing, um, or something that is only on, on, on, on, on the frontend. And hopefully it's a great developer experience. And we are going to talk a bit more about that.

Managing Multiple local versions of PHP

So the first big feature I wanted to mention is that it also manages your local php version. So it does not install php itself, but if you have more than one version on your machine, then you can use them. And that's very interesting because it allows you to test a new version of PHP, uh, out of time. So if you want to test your website on php 7.3 for instance, that's a use case. Or if you want to test a different configuration with, with, you know, a different php.ini file or something like that. And of course if you're working on different projects, you might need more than one php version. Um, and actually when I started to work on that, I realized that most people do have several php versions on their machine even if they are not able to access them. So for instance, if you are using the Mac and you're using Homebrew to install your php versions, you do have more than one version because whenever you are upgrading to new version, the oldest ones are still there on your machine. And you can switch from one to the next one, but that's the global configuration. So I wanted something that is local and not global. So there is a command to do that. So if you run this command, you will see all the things that you have on your machine. That's what I have right now on my machine and it detects the kind of things that you have: php-cli, php-fpm, php-cgi and things like that. And you can see which one is going to be used for the web server. And a default one.

So if you want to change them, if you want in a directory for a project, if you want a very specific version of PHP, that's how you can do that. So you can create a .php-version file. And then, um, I can say I want php 7 or php 7.1 or php 7.1.13 or whatever, and it's going to use that specific version in that specific directory. So it's not going to be global, it's really local. Okay. So that's great. Um, so when you, when you run server:start depending on the current directory, it's going to check for a

.php-version file. If there is one, it is going to use the version from that, if not, it falls back to .symfony.cloud.yaml and by default it uses what you have in your path.

Um, it also, it's also about having the best backend possible. So again, php-fpm by default, if it is not available, then php-cgi, and as a last resort, it uses php -S. Okay, it also works on your terminal. So if you want to run PHP and have the same version of the local web server, that's how you can do that: symfony php and whatever. It uses the exact same algorithm as the web server of course. So that's very nice to be able to have the same version for the web server and also for the command line. Um, and you can also configure PHP per directory. I talked about that earlier. So if you have, if you need a special time zone, for instance, for a project, that's how you can do it. You can create a php.ini file when you have some kind of php configuration there. And then it's going to be loaded automatically by symfony php and also by the web server. So you have the exact same configuration for the web server and also for php in the command line. Okay. And of course, that's something you can do as well. You can just copy the symfony binary and name it php, pecl, php-fpm, php-config, and it's just a thin wrapper. And it's going to act as symfony php, which means that that's exactly what I have on my laptop, which means that whenever I type php something, it's going to use the right version for the, for the directory I'm in, which is kind of nice.

Logs, Logs, logs

Okay. I talked about logs. Logs are very important if you wanted to debug things, you need to have all the logs failure but very easily. Um, so that's how you can do that server:logs. And then it's going to tail all the logs from php, of course, from php-fpm, but also for from symfony. So as you can see here, we have some logs from symfony, uh, and as you can see, it's not exactly the same as the logs that you have from the var/log/dev.log file, because we actually parse all the logs, we convert them to semantic JSON, and then we kind of humanize them so that it's easier to spot problems. I don't know for you, but sometimes when I'm tailing logs, especially for the production environments and there's some kind of a problem I need to debug the problem, it's kind of hard to figure out where the problem is and where you got your exception. So here we make it very clear. So if there is an error like we have here, you have this red thing so that you can spot the problems really easily.

You can also use it even if you are not using the web server by passing a file directly to the command and it's going to read the file and parse them and, and you manage them.

Using TLS Locally

Okay. The next thing is TLS. How many of you have a website in production without https support? Ah, you can raise your hand. Okay, I'm going to raise my hand. Nobody really? Okay. The, so that's, just one, he's not here, he's somewhere in the room. So TLS is very important nowadays. You need to have https support so I think you need to have support for your local, local server. It's more similar to production. You can detect mixed content early on which is already, always a nightmare if you not detect that before going to production. And sometimes it's also a requirement for some JavaScript libraries. For instance, if you are using stripe, if you don't have https it's not going to work. Okay. So the first step is, and it works in the exact same way on the three platforms, you can use server:ca:install. So what it does, behind the scenes, it creates a local certificate authority. It is going to register it on your system trust store, which is different depending on the platform. It registers it on firefox because Firefox is using its on mechanism for that. So, uh, that's also something we do and then it creates a default certificate for localhost. Uh, so that when you actually access your website is https, you have a default certificate to work with. So it's everything is stored under your own directory, under the .symfony directory, and then certs and you can see, um, the, the local certificate authority and a default certificate.

And done. So now when you start a server and you have this tls support, you are going to be using https by default. And by the way, we have http and https on the same port. So you don't need to remember two different ports for that. And by default we redirect http to https.

Local Domain Names / proxy:start

Okay, so the next step is having local domain names. And that's also nice because of a bunch of reasons. So the first one is that, you know, some project depend on the domain name. That's the case for the symfony live conference website for instance, we have Lisbon, we have Paris, two different domain names. And depending on the domain, the website is not going to be exactly the same. You don't need to remember the ports, so don't need to remember that this website is on port 8000 or that one on another one. So you have something that is stable, it can be stable across machines as well. We can use the same domain names on two different machines and it uses regular ports. So it's, you're going to use port 80 and port 443 for https, which is again more similar to work you would have in production. And that's always a nice to have.

It's totally optional, so you don't need to use it. If you want to use it, you can start a proxy. So symfony proxy:start. you don't need to be root. And that's very important for me. You don't need to be root to use the proxy even if you are using port 80 and 443 just for a reason. And the reason is that we are using a proxy configuration that works in the exact same way on the three platforms. Uh, so you have nothing to install, you don't have to install DNS mask or whatever. You don't need to mess with your local DNS configuration, you don't need to edit your /etc/hosts file, for instance, and it does not intercept regular traffic

because that's the configuration. So if you want to have a look at how it works, you can curl this URL and you will see piece of, JavaScript, and if the domain name is wip, which is the default TLD we are using for the proxy, then it's going to use the proxy. If not, it's just, you know, doing nothing and, and, and, and not intercepting traffic.

Okay? So, so the first thing you do is you attach domain to a directory so you can do to your project directory. Then you attach a domain. So here I'm going to attach symfony.com On this symfony.com website, and then you start the server if it's not already done, and it's going to attach a domain name with the right port. So that's done automatically. You don't need to configure that. You can edit all the configuration under the proxy.json configuration file. So wip is the default if you want to use .sf that's also possible. I think that's kind of nice. Um, we have wildcard support, so if you need more than one domain for a project that's also possible and all the tls certificates are going to be created on demand locally, so we don't need to be connected to the internet to make it work. It works locally, so it's not using a, it's not using a letsencrypt, for instance.

Uh, the nice thing about the fact that you have a proxy for the whole tld is that if you are hitting something that does not exist, then you have, um, a good error message saying that, okay, it's not linked to any directory. If you want to do that, you can run this command to make it work. Um, and if you go to the proxy directly, you have the list of all the domains that you have in all the projects that you have on your machine that can just click on them. And it does work with curl as well, so you can just export HTTP_PROXY and HTTPS_PROXY with proxy url and it's going to work for curl or, or whatever you're using because of course, not of course, but curl does not use your system cert store. So you need to do that to make it work, if not, you're going to have an error.

Workds, Daemons

Okay, so now I'm going to talk a bit about the integration with Symfony because in the Symfony world and it's a trend that we can see over the last few years, I would say, where we have more and more long-running commands in the Symfony world, right? We have one for a dumping, the var-dumper, things that you might have in your project. We have one to consume messages from the Messenger component. We also have, Encore, if you are using Encore or yarn or whatever, you want, you might want to watch what's going on in your project so that I can compile their assets whenever they change. So we do have support for that as well. A symfony run, I'm not sure I mentioned symfony run before. So symfony run is a way to run a command with some nice features and I'm going to talk about them after the slides. Anyway, so symfony run -d daemon and then you can type whatever you want and it's going to run in the background for you and manage the background process.

The nice thing about that is that when you are using server:log, then you are also going to have the command log, so all the logs from php-fpm, Symfony project and also all the commands that you run via the symfony run command. And if you say symfony server:status, you can see that you have the web server and the command that is running on this PID. And you stop the server then it's going to also stop the command. So you can run any numbers of commands and they are going to be attached to your project.

Integrated with Symfony Cloud, Optimized for Symfony

Okay. Of course it's also fully integrated with Symfony Cloud and optimized for Symfony projects. So the previous one - you will be able to take a photo in a minute - at the previous one, so you know encore it's, it's not really linked to Symfony, so you can use it for any project. But now I'm going to talk about things that are really optimized for a Symfony project. So I talked about logs and Symfony logs that we parse. But we have more than that.

So the first one is that, and I talked about that at the beginning of the session, I want to be able to use my local server with my local files, but I don't want to install a database server or redis or whatever on my machine. So I'm going to show you how you can do that with Symfony Cloud. And I think that's a very nice scenario. So let's say that we have a project hosted on Symfony Cloud with the demo.com domain name. That's how you can get started. You have your project, you run project:init so that you generate a default configuration for Symfony Cloud. Then use symfony deploy it's going to deploy a cluster of containers, for php of course, for all your services. It's going to provision a tls certificate as well, and then when you deploy, it's going to be your master branch, which is the production environment in terms of Symfony Cloud. And then you can attach domain more or less like what we had before for the proxy and it's going to attach a demo.com domain to your project. And by default, of course, the environment is production and debug is set to false.

Now, let's say I want to work on a new feature or I have a bug in production and wanted to debug the problem. Instead of working directly in production and, uh, um, or working on my local machine with fake data, I want to have a replicate of what I have in production. So what you can do is, you can use the env:create command. It's going to create a new git branch for your code locally, of course. Then it's going to deploy a new cluster of containers, which is going to be the exact same cluster as what you have in production. So the same services, but also it's going to copy the data, the exact same data that you have in production, and then you can use symfony open:remote to get... the website is going to be exactly, exactly the same as production with environment production as well. Uh, let's say that you have a problem, you can use symfony:log to get all the logs from the remote web server, so that's the same as server:log, but for the remote services. But by default we want to replicate the production environment. So the environment is production, even for this branch. If you want to switch to the

debug mode you can use symfony debug and it's going to switch to the development environment and set debug to true.

Tunneling Services Locally

Okay, so that's remote. I have all my services remote now. I want to debug things locally on my machine. So I have attached my local web server to the demo.com.wip domain name. I'm running a local web server and if I opened the website I'm going to have an error because I don't have the database locally. Right? And I don't want to install the database locally. So what I want to be able to do is develop locally with my file on my laptop, but I want the data from the production environments.

So the way to do that is that you open a tunnel between your local machine and the remote services. And that's all, that's all. You refresh the page, don't need to restart anything and it just works. It just works because behind the scenes we are getting the information from the remote services and we are, we are exposing those information to the web server directly. And as you can see on the, on the right side of the screen here, we inject a piece of code in the web debug toolbar. It tells you that the tunnel is open for the some-bug branch. So that's what it is using. And if you have a look at the profiler. So that's my .env file here in my project. You can see I'm using the sqlite database, uh, and I've removed the file. So that's why I had an error. And here you can see that it's not in the .env, uh, on the right side, which is the profiler, it's not there because by default the local web server detected the tunnel and it exposed the service as environment variables in php-fpm automatically. And as Symfony and Symfony Cloud share the same conventions, it means that Symfony can read the environment variable and boom just works.

Um, it doesn't work by default if you are doing the same with the master branch because we are talking about production. I don't want you to mess up with the production data. So by default it doesn't work. You can make it work by setting a very long environment variable to be sure that that's really what you want to do. But by default it's disabled.

So let's recap. So you have your web server in production, on Symfony Cloud with your services. Here, I just have a database. I use a symfony env:create to duplicate my environment so it's going to duplicate all the files and the infrastructure in a local git branch. Then we open a tunnel which exposes the DATABASE_URL here for a database to PostreSQL, which is a link to the database for the some-bug branch. And then that's read by the server:start, which means that locally I have the web server, but it points to the db from my some-bug environment.

Which means that now I can debug locally with the production data, which is kind of nice, which is what we did 20 years ago when I started to work, uh, with the web. Whenever we had a problem, uh, we had to connect to the machine and mess up with the code directly in production to find a bug and, and fix that. That's not needed anymore. But the experience is exactly the same. And from time to time, if fixing the bug takes time, what you can do is you can use the symfony env:sync command, which is going to sync the production environment data back to your, uh, some-bug branch, which, which means that you will get that also on your local machine. Okay? So you work on the problem, you can fix the code, you can update a database schema if you want, you can change some services, so you can add service to fix the bug or add a new feature, we can upgrade to a new version of the database or whatever you commit. So when you commit, you're going to commit the infrastructure changes and also the code and then you deploy. So it's going to deploy on the demo.... uh, the some-bug branch. You can use symfony open:remote to check that everything is good. Um, and remember the big difference here is that remotely we are using the production environment. So it's nice to be able to check that in the production environment, it works in the exact same way as the local one. And when you're ready, you can checkout master, you merge your git branch and then you deploy again. And the nice thing about that is that it's going to be really fast because we are using the exact same container images from the some-bug environment. So when we deploy to some-bug environment, we created a container with all the information, the code and, and, and, and, and the configuration and everything, and it reuses this image when you deploy to production so you can deploy with confidence because you checked that everything worked well into the some-bug environment.

Okay. It also works in a terminal. So if you open a tunnel and then if you say symfony php, and I want to get the DATABASE_URL environment variables, you will get the one from the remote service. If you want to run something with symfony console, you can say just symfony console. It also works with some commands. So for instance, if you say symfony run psql, it uses the binary, the psql binary that you have locally, but behind the scenes, it sets all the environment variables needed for a PostgreSQL to actually connect directly to the symfony cloud database.

And that's, what I wanted. I wanted something that is fast, that is convenient. So I have everything locally, but the remote services. It works on all the platforms, including Windows. I spent, I spent *so* much time on that. You're welcome.

And you can test it today. You can go to symfony.com/cloud/download. So again, it's not just about Symfony Cloud, we will probably change the URL at some point. You can download it for all the platforms and you can test the web server. It's still an early version of that so any feedback is very welcome. And I think that's all for today. Thank you for coming again. Thank you for listening and enjoy the conference.

Chapter 2: Knowing your State Machines

Tip

SymfonyCon 2018 Presentation by Tobias Nyholm

Web development is not just about delivering a response. It is also about writing good code. The state pattern will help you move complexity from being all over your code to one or more state machines. This talk will introduce state machines, show how to identify uses of them and implement them in your Symfony application in an object oriented manner using the Symfony Workflow component.

So excellent, excellent. Thank you all for being here. Um, do you know this is a large conference and you know, when you're organizing a large conference like this, you do want to have the best speakers, right? So, and what do you usually do when you're having a conference? You put the best speakers upfront, like the keynote is obviously the most impressive one and then you have the best speakers and then they slowly decline. Um, as you know, Michelle is not here, Michelle was about to do this talk so and her kid got sick and you can't really leave a two year old home in Switzerland by itself. So she had to stay home. So the conference organizers thought, Hey, what should we do? Michelle is sick. So obviously they did the best thing they could. They call the second best speaker, right? Unfortunately that person couldn't make it. So they called the most knowledgeable speaker who knows the Workflow component the best, and that person didn't make it either. Um, however, like five or six calls down the line, they find my name and here I am and I hope I can give you. Yeah, thank you. At least one guy or girl is very polite. I hope I could give you like a, a talk that could match Michelle's talk.

So with uh, with that said, my talk is titled Knowing Your State Machines and I would like to do this talk because I believe web developers only care about printing out HTML as quick as possible. They don't care about fancy algorithms, they don't care about design patterns. They just want to print stuff really, really quick. And so I want to tell you about something that makes your code a little bit more elegant. So I didn't, I didn't bring any state machines. It's nothing I can have in my pocket. It's a state machine is just something that hold states, like any states. It could be the state of a pull request or this small washer on your - small lights on your washer. It could be, yeah, it could be anything. So it's basically a way of thinking, a way to relate to problems. And why state machines? Blah Blah. There's a lot of texts in this. If you work at a company where your manager needs convincing, this is a slide - just to take a photo or show him later. I'm sorry, show him or her later.

Tobias does a lot of Open Source

Quick about me. I do plenty of work on Happyr. We are a job advert platform - it matches people with confidence, blah blah. I do plenty of Symfony things. I run PHP meetups in Stockholm and I do plenty of open source. I started my real open source career in 2015 on a SymfonyCon, just like this one. The PSR-6 cache had just came out. So me and my friends were like, hey, why don't we implement this. Why don't we make an implementation of PSR-6. So we did. We created the PHP-cache organization and CacheBundle and I thought open source is great. So I started to get involved with HTTPPlug, which is kind of an abstraction over HTTP clients and when you do HTTP clients, you do API clients and then you do mail clients. And I even wrote like, Oh, this is how you should do API clients, this is the boilerplate for FIG. And currently I'm maintaining both Guzzle and Buzz. I've written my PSR-7, I've written PSR standards and stuff. I, I do plenty of things. The most interesting library here, which I like the most is this one, the NSA one. With NSA, you can totally violate class privacy: accessing private properties or methods is super easy with NSA, I do also contribute to a bunch of other libraries: these are only the most fun ones.

So how should we do this about state machine? How should we do this? I would like to start with a lot of theory and then I will put on some examples, examples, and then more examples. How does that sound? Excellent. Because I've got slides for this. How I can barely see you, but how many of you, how many of you feel like you know, state machines by heart? Excellent. It's like 12 of you, which, which is excellent. Then I guess like the rest of you will learn plenty. You are.. anyhow, let's get started.

Graph Theory

Let's get: first rule of state machines. We've got to talk about the graph theory and the first rule of graph theory is like these, these are not graphs. These are charts, these are diagrams, these are call them whatever. They are not graphs. A graph. It looks like this. A graph has a node. A more interesting graph got two nodes, and nodes can be connected with edges and edges can either be undirected like this one or it can be directed. An edge can also have a weight. Does this seem familiar for most of you? Excellent. Excellent.

This is a full-fledged graph. This is like the cool graphs can be. And it can represent pretty much whatever thing you like it to represent. If you slap on some numbers on here and maybe you can have a problem, maybe this represents cities and roads between them. Maybe it represents computers in the network, maybe, maybe they represent people and how well they know each other. And say we've got a graph like this and you wonder how much.... say they are computers in the network. How much data can you transfer from a to b? This might be a problem you're having, so this is called the maximum flow problem and there is a solution to this. And say that you want to find the shorter distance between a and b. You can solve this this by Birpartite breadth-first search. And say that you want to... Say that you want to find out what's the cheapest way of visiting all these cities, or these nodes, visiting only every node once. That's a traveling salesman problem and there is a solution to that as well.

Or say that these are cities and you should put out some electricity grid between them. This is the minimum spanning tree problem and there's a solution for that as well. What I want to tell you is if you've got a problem, if you can express this problem as a graph problem, there's already a solution to it. So you don't, you don't really have to solve these problems yourself, you can just solve the problem getting your problem to a graph problem. Does it make sense? Thank you. This, what I just did, I gave you like six months of university math of graphs. So this is basically the big graph theory I just gave you a quick introduction of.

A cool thing about graphs is that you can reduce them. Say, consider this graph. I want to reduce them by giving them a new rule. And say, the rule might be: let's remove all the loops. If I remove all the loops in this graph, I get a tree. And there's plenty of algorithms you can do in trees as well. One of the most difficult to implement is the red then black search tree, which basically is good way to store things if you want to search quickly in it. Anyhow, I want to show you this. You have the big graph theory and inside of it you have the theory of a tree. And you can continue reducing graphs with different set of rules and you will get a workflow.

What is a Workflow and State Machine?

A workflow also has nodes, but sorry, but we decided, oh also has nodes and it also has edges and we decided that every edge should be directed. We also decided that we call the nodes places and transitions. And a rule in this workflow is them saying: a place can never be connected to another place. All of this has to be a transition between them. I know I'm shooting a lot of rules at you now, but do you kind of feel comfortable? Excellent. And we decided, I mean just so it's easy to see, we draw these transitions like a square and a good example of a workflow is this: if I'm in place A and execute the transition t0, I'll get to place C. If I'm in place B and they execute the transition t1, I get to place C and D. Make sense? Good. If we continue to reduce this workflow, say we want to add rules saying that a transition could only have one input and one output and also the names has to be unique. Then we actually get a state machine that looks something like this. So workflow multiple one input, multiple outputs. A state machine only can have one input and one output, something like this. So if execute t1, if you are in B and execute t1 you get to D. And because I'm drawing these images by hand, I will choose to draw with this annotation. Fair?

Real-world Workflow Examples

So right now I have been talking for eight minutes and we discussed the big graph theory and we discussed the tree and we discussed what a workflow is and that a state machine is just a reduced workflow.

So let's start with some examples. I mean I know this part was the boring part, this part was the theory part. Let's start with some examples of recognize that we recognize. This is a simple state machine. We defined three states and we defined how we can move between the states, right? So it's easy to see that if you are in green we can easily move to yellow and then we can easily move to read. However though if we are in green, we cannot move to red. There's no arrow to red. It's impossible to go direct to red, which is which is a good thing. I've got to keep asking you simple questions until you give me some continuous feedback. Do you understand this?

I work at a company called Happyr and we deal with a lot of job adverts. Basically we put up adverts and we match people and people in common with soft skills, etc, etc. Anyhow, we have a lot of job adverts. So in a business meeting we decided like this, this is the state machine for a job advert. So we started draft and then we can send it to review. While in review, we can trash it, untrash it, and we can send it to review again if you'd like. And from review you can publish and archive it. It is impossible to archive something that is in review or that is something that's in trash. It's impossible. Excellent.

Since you are very confident now, what is this a workflow of? Someone said a pull request issue. Uh, I will say yes. Correct. So basically you start coding your pull request, you submit it to Travis, Travis runs for awhile. You may want to decide when you're waiting for review and then you may want to decide to update it. So we'll go back to Travis running. And maybe someone says, hey, we need some changes. So you go back to coding, then you update it, then send it to review and hopefully it gets merged or else it could be rejected and re-opened again. Seem straightforward?

So how about this? Oh yeah, pull request. So how about this? What is it the workflow of? What is the state machine of? People usually yell out like, like you, sorry, like you did. You said doors. I'm like no, no, not doors, it's elevator doors. But

you're equally correct. It's doors. So basically this is the four states that a set of doors can be in. Either the doors can be opened and then you press the close button and then be closing. While the closing we can press open button so they go to opening. It's impossible for doors to be open and then in one, the next instance be closed. They have to be closing, right?

Moore & Mealy State Machine Implementations

I think. I think we kind of know, know state machines now. Maybe business a dictates that... Maybe business a business dictates that you should have a form like this, like maybe a sign up form or whatever. So you can have a state machine to know where the user is in the form. So if you leave this process, you can, you can get back to where where he or she left off. And if your extra nice you can have back buttons so we can travel back and forth in this process. If you want to implement this, there's basically two implementations. I'm gonna can talk to them with them bot. No, I'm not going to talk *with* them. I'm gonna talk with you *about* the both of them - the implementations. The first implementation is the state pattern and that's using the Moore state machine. And a Moore state machine only cares about the current state. The next state only cares about, the next state is defined by the current state. The other state machine is the Mealy state machine and the next state depends on the current state and some input.

Moore State Machine Implementation

I just want to say this out loud now and then I'm gonna repeat it later. Let's start with the state pattern and the Moore state machine. This is basically what you easily read on wikipedia. You... a state. I'll click this. Say you want to implement, something like this. We want to implement a... something that reminds you to complete your profile, say a user signs up and you want to say like "Hey, you should add your name" and we remind them twice about adding their name. If in order to add your name, like we should, hey you should update your image or you should add your resume. So implementing this with a state pattern means that each of these five states have a class and that class itself only have one f unction. It has a Send function. And, and that calls itself decides what to do and where to go next. Makes sense? So the idea here is that we have, have, we have a worker like a cron job saying like: Hey, give me all the state machine from database somehow and start yourself. I don't really care what you're doing - you just send the email, send the correct emails.

So in the implementing this, I'm basically looping over all the states and if the, if the state machine says "I'm done", then we're done. So any given state just looks something like this. I, this is, I mean, you send an email and then I say go to next state. Makes sense? We should obviously do a little, we should modify this a little bit. We can maybe say like, if the user doesn't have a name, if the user *do* have a name, go to add your image state. So the state itself says where to go next. The sates itself decides everything what to do. And the state is a single class. We should also be a little bit more friendly. This would actually flood the user's email address, so uh, email inbox. So we should: you should be in this state for at least seven days until we send another email.

I know this might feel weird. I know this might feel like that's not really PHP. It feels more like Java or something else. It feels weird and I will agree with you. This feels a bit weird. Using this and, for the simple state machine I showed you the five states, you have a bunch of classes like this. And they all pretty much look the same. Each of this state just look, oh, they look like this.

So I... with this I would like you to get an idea how the state pattern works and some of you guys are nodding politely. Excellent. You obviously have issues with this. I mean you have issues: how do you store the state machines in this database? How do you keep track of the state? How do you store the createdAt entity... or data. Also it feels kind of weird that you have one class with state.

Mealy State Machine Implementation

So let's, let's do something different. Like now we know the Moore state machine, we know the state pattern, so let's look at something different. Let's look at something that feels more like PHP. And I'm going to stress this multiple times: the Mealy state machine, the next state depends on the current state and some input. So it's obviously a little bit more flexible. Let's try. Let's consider this state machine again, you all know this by heart now. If you would write an implementation of this, if we like PHP developers would write an implementation of this, how would we do? Well, what's the first? We obviously need a class, right? We need a state machine - traffic light state machine class. And we kind of want to have a function that applies new state. And I imagine it to be good to have like a function that "can I apply this state?" Good. So let's. So I create my state machine looking something like this. I define all my states as constants and I also choose to have a private property called state. And then I just, via a switch statement say: if someone say: "Can I do transition to yellow?" I check if I'm currently in state green and someone says "can (I go) to yellow?" Then sure, that's allowed. If I'm in state red and someone says "to green", then it's obviously no, I return false. And I do a similar, very similar thing for apply. I just do something like this.

So would this state machine. I can easily, I can easily implement my... This, this class is easily implement my traffic light state machine, right? There's an issue though: this state machine has a private state. So I need to store the state machine in the database somehow. This state machine is only for one traffic light. If I have multiple traffic lights in my system, I have to have

multiple state machines which each needs to be stored in the database, because of the state property. What if I made my state machines more stateless? Like I removed the private state and put that on the, on the traffic light entity? So I did something like this. Can this *specific* traffic light do this transition? So now I don't need to store the state machine in the database anymore. I can use this state machine for all the traffic lights.

It feels quite, feels quite good, right? Yes. It feels quite good. I diddothe same thing with apply function. I just say $trafficLight->getCurrentState() and I have a $trafficLight->setState(). Very similar, very small changes. And I feel pretty good about the state machine at this point and whenever I'm using this, it might be an idea of mine to have a function that allows more traffic. I mean if, if, if the state is currently in the red state and I want to allow more traffic, I just, whatever state they're currently in, I just move it towards green. So I might attempt to do something like this. So I know if I call allow more traffic twice. I will definitely be in green.

Making the State Machine Generic

You look confused and you should. Because this was like my trickery, but I didn't trick you. If I want to make this state machine more generic, more reusable, obviously this is a horrible idea. Right? So if I'm going to make it more reusable, I may want to consider instead of having the states all over the place, I may want to put them in a, uh, in a property like this. So with this weird array looking thing, it's actually me trying to be a bit clever again. So I can just say: can I do the transition? I was checking if this is, if this property is set, if this array key exists. I do the same kind of wizardry in apply. Do you feel confident with this? You kind of understand it at least that's, that's the most important thing. However, say you want to make this even more generic. I mean, I shouldn't, I mean this is only for traffic lights. I said, traffic light here. What if I do something different? What if I... Oh, sorry. Oh, look, the states, it should be states here. What if I inject the states instead in a constructor like this?

Then I have those states - I can do this for any kind of state machine and any kind of traffic light. And now, oh no, I have traffic lights over here. What if I did something like this to have the StateAwareInterface instead? And this StateAwareInterface just have a getCurrentState() and setState() method. With this class of 15 lines. This is actually a pretty good, pretty generic state machine, right? And it all feels (like) PHP, all in one class and I can configure it however like and use an object that I like. I hope we feel like we understand this. Does anyone have... oh no, I can't ask you for issues. Does everyone feel like you understand this class? Excellent. Excellent.

Hello Workflow Component

I'll tell you a little bit of a secret. This is actually how the Workflow component works. This is the workflow of the Workflow component. And it was very true like two or three years ago, this was exactly the Workflow component. And nowadays there's a lot of edge cases, but in principle it works exactly like this. The Workflow component all have some more functions we showed. I showed an implementation of "can we do the transition?" And "apply transition". The workflow component has two more methods saying "like what state are we currently in?" And what are my valid transitions?

Workflow Component Example

So let's show some examples. Same old job advert workflow again. In the Workflow component, this might look like something like this. The red circle is the current state, so I can say: give me the state machine for job adverts. And I say give me the current marking for this entity and give me the places. And marking here sounds weird. It's because it's basically the abstraction of getting properties from the, from the Advert. If you ignore that, it's, this is very simple, right? So I get an array back and I see I'm in Draft. I can continue showing examples saying: state machine, can I do publish? No, I can't, but can I do "to review"? Yes, that's true I can. So I continue. I go to review and when I execute this line, it actually moves the circle away and from here I can say go to trash, go to draft and go to review again. And in this state I can say which are my enabled translations, and I get an array back to say I can go to publish and trash.

I just, you just understand everything in the Workflow component. The Workflow component itself got like 15 classes and this is the most complex one. And you understand it fully now, which I think is quite... not impressive that you understand it, but it's impressive how simple Symfony components are. Uh, this is also, since this is Symfony, this is also available in Twig. Imagine, imagine when you're editing an advert on Happyr, imagine it looks exactly like, like Wordpress. You have this big text box in the middle and then you have a sidebar with like trash, publish and stuff button like this. This is the code for the sidebar. So I'm checking if the workflow "can" go to review, then show the link for review. If the, the, the, this advert can be published, then show the link for published. So the same sidebar will look differently on different adverts. And if you are, if you're smart with the naming of your work for transitions and you're naming of the routes, you can do something like this to print the sidebar. I want to show this because it's, it's possible, but you probably shouldn't do this. I do want to say that... I mean, I think this is pretty cool, right? You, sorry.

I need to reverse, blah blah, this is a sidebar, decisions... This is pretty cool, right? Your sidebar really, it changes depending on the states. And what's cool here though is on a business meeting, we sat down and someone asked, how come you can't trash something that's in draft? It's obviously no arrow here, but what's the reason? And I'm like, yeah, there is no real reason.

So let's, add this arrow. And here's, here's the kicker. I only changed the whiteboard. I want to change how the, how the state machine looks like. I did not change any of my models. I did not change any of my controller. I did not change any of my services, I did not change Twig layer, I didn't change anything but my configuration. And still with this change, I will see the trash in the sidebar when the advert is in draft. And that's pretty cool. A year ago, I did this presentation and when I said that's pretty cool. People like, yeah, that's cool. Good job, and they were kind of cheering. But you become the words, not only to be excited with me.

So I want to change the configuration And the configuration looked like this. It's super simple, This is my, this is my core of my business. This is my business logic. If you're really, really paying attention, you might spot a weird thing here. I told you that a workflow can have, no sorry, I told you the the state machine only can have one input and one output. And clearly I have two inputs here. This Symfony configuration is smart enough to know like, oh, I can make two transition of this. So it, it says, it looks like only one, but it's actually gonna to become two transitions.

Multi-Step Signup Form Example

And also quickly want to show you a multistep signup form. Business may dictate that this is the way we sign up to our service. So have an intro, you add your name, you add your email, you add your twitter, and then you're done.

And doing this workflow in the configuration is simple, you just do it like this. You just say that this is a state machine, this is my places, and this is how you move between places. And business might say this is good and all, but we want to have back buttons. We want to have a workflow like this where you go back. And we said, sure, no worries. We just do something like this. However, this is not valid YAML, right? You cannot have multiple keys on the same level which is called back. So what we need to do, we need to do something like this instead. We add arbitrary numbers or arbitrary names and then we have the name. So this how we get around the issue with YAML. So with this you can actually please the business to have back buttons.

Workflows: Modeling a Process

And I know I've been talking a lot about state machines now and I haven't really... This is called the Workflow component. And it says workflow like all over the place here, right? So what is the difference between a state machine and workflow? In the beginning I mentioned the state machine is a reduced workflow, but what is the actual, what is the actual difference? As an example, I want to show you this: this an example of a workflow. Previously we've only seen state machines. This is the example of a workflow. And the main difference in a workflow and a state machine is that in a workflow, you may be in two places at once. It might feel weird, but it's actually valid. A workflow more or less describes a process. A state machine describes traffic lights, when you actually go back to the beginning and stuff - how you move. So this is a workflow for a blog post: it's first a draft, then you request reviews and then you get in, you get in the places for waiting for spellchecker and waiting for the journalist at the same time. So there's an example where a transition that makes you to be in two places at once. And spellchecker is probably automatic, right? So that's pretty quick, but you have to wait for the journalist. And it might be tempting to try to publish this blog post right now. But you can't execute the publishe transition because both its inputs are not fulfilled: we don't have one of the inputs, right? So we have to wait for the journalists to catch up and then you can eventually publish it.

I will also say a workflow rarely contains loops. It's rare that you go back in your workflow. However, state machines, loops are encouraged... kind of. People used to ask me: when should I use workflow when I use state machine? And the answer is boring. It's obviously: it depends. I tend to use workflow when there's a clear process. I use state machine for everything else. It's just how I like it. The most important thing is that you know which one you're using. You need to know how to relate to them. It's like, for 99 percent of the problems, it doesn't really matter which one you're using. But you should know which one you're using. Does it make sense?

Workflow Events

Excellent. Excellent. So back to the Workflow component. If you have been around in the Symfony community for awhile, you might notice Symfony have like 30 components and the components are very decoupled from each other, but they play really, really well with each other. And the secret there is events. Events is the thing that makes them play well with each other. And the workflow component is no exception. We have events or, the Workflow component has events for everything. We, all the events looks like this: you have a generic event, say workflow.leave. Then you have an event, say workflow - the job advert - when they leave. And you have events for, workflow, job_advert, leave, draft node.

And all these, you have all these three types of events for everything. So we have workflow leave, when you're leaving a node. You have transition that would actually transition from one node to another. And when you enter the node, when you have been entered the node, and when a transition is completed. And then you have announce saying: Hey, I'm all done. It might feel weird that we have so many events. I mean these add up to like 21 events for each transition, which is a lot of events. However, there, there's a reason for all of them. Like, I know I've reviewed all the pull requests adding more events

and they all makes sense in this specific weird edge case. You should only care about the one you care about. I mean, you shouldn't... I only use like leave and enter... and that's fine.

Guard Events

So there's plenty of events. There's also this even more events. There's one event called the Guard event and the Guard event is something that you use for, adding extra logic, like external dependencies. So: can I publish this blog post? Like say we want to, we want to have a rule saying: you can only publish this blog post if it's between3 and 4 PM. Or if the database is online or if the manager is in the office or if you have the role ROLE_PUBLISHER. So a Guard, a Guard event is how you add this extra knowledge to your state machine. And I think you all are comfortable with events subscribers. I basically say at the bottom, I say: this is the event I'm listening to - workflow.job_advert.guard.publish - like the "got published" transition. Whenever that event is dispatch called the guardPublish() function and run this three lines in the middle. I would, I will assume that you are comfortable with this.

Normal Event Listeners - e.g. Logging

Except for the Guard event, we may, we can register normal event listeners. Say you want to log every time you move from one node to another. So I say, the workflow, when the job advert leaves any node - workflow.job_advert.leave - then I just to do a logging. I say advert with ID performed transition from one node to another. It might be useful to have this kind of log message.

More Workflow Examples

I'm checking - you're still with me? It looks like most of you are with me. I... no surprise: I love the Workflow Component. I use the Workflow component all over the place. If I have... the Symfony Security components have the concept of voters, which are basically: if someone wants to watch, view an advert, I got to make sure that advert is published. And I'm using the Workflow component to make sure I am in state published. So here's a simple security voter and I say: if someone tries to view then I check this: if the state machine got the marking for the advert and marking is published, like if I'm in published state, then you're allowed to view it. You should not be allowed to view something that's in draft. So I used my state machine all over the place.

I also use the state machine for domain events. I know for sure that no advert in my application will ever be published unless the workflow.job_advert.enter.published event is dispatched. I can trust that for sure. That's why I use this event for my domain knowledge. Like say business say: you should email all your users whenever an advert is published. And this is a good, is a good place to hook into this. So I never dispatch any events on my own, I just use the workflow events. Obviously we don't email all our users when adverts are published, I just wanted to say, but it's, an example how you use your domain logic with these events?

Rendering the Workflow as an Image

Um, whenever, whenever I'm in business meetings and we, the business people say we need this and we need that and we gotta figure out a new way to do stuff. We write the workflow on the whiteboard. And we change our minds, we go back and we add new states. We, because business people, they tend to like graphs and images, they tend to like, big circles and squares and arrows between them. So it's easy to work with, work with a whiteboard. However, when the whiteboard meeting is done, I want to make sure, I go back to coding and I got to make sure that I have configured this properly. So I add my configuration and then I run this command, the dump command and I'd get an image. I mean the dump command just prints out some text and I use this program called dots or this script called dot which converts, takes to an image, and I can get this image. So then I compare the whiteboard with my image and more often than not I made a mistake in my configuration.

So this is a good way to make sure you configure this properly. And so I usually dump two or three times and then I, then I know I got the right conflagration.

Hints on Finding State Machines in your App

And I know I talked a lot, and state machines they sure are great. I gives us, I've told you so many benefits, but how do... I mean where... so I want to give you some hints. I want to give you some: if you ever have, if you ever have code like this, this is code, I've written, I wrote a cCRMm or whatever it's called, basically for salespeople and they have a quotation and I defined the Quotation can be in these three states, four states. I even have a property called status. And whatever you see code like this in your application, you should know like, hey, maybe I should use a state machine for this.

Also, when you're looking at your database and you see something like this. Like have boolean for active, a boolean for ended, a boolean for closed. This is also a good sign, like "hey maybe a state machine will be better in this scenario" And

people asking me: "Oh, Tobias, should you really have state machines if you only have three or four states?" And the answer is obviously it depends. Are you sure you never ever would add more states? How are the rules between those states? How do you move? And I would say, if you want to make sure you don't have any bugs. Yes, a state machine might be a good idea. Obviously if you're only have two states, I mean it all depends and I understand, I'm sure you're getting the gist.

Identifying Processes in your App

So how about processes? If you don't have a quotation, but a process. What if you're doing e-commerce, it's obviously a process here. You obviously create an order somehow and items are being added to that order and then you need the user to fill in the delivery address and then you need them to add the payment method and then you need to call or somehow notify the delivery team to make sure that that order is delivered. And this is also like if you have this strict workflow, you should think of the Workflow component. You should think: "Hey, maybe, maybe I should use a workflow here." And then you start drawing on the whiteboard: you draw the order is created, and then somehow we add an item

- we have transition add item, so the item is added. And then you add the shipping and then you go to the checkout and after the order is confirmed. And you look at this. Yeah, sure, this makes sense. This is a clear workflow. However, this means a user can only can add one item, right?

Predictability and Less Bugs with the Workflow Component

So we need to add an extra arrow here. The user can add more than one item so it can buy more stuff from you, right? And this, it's pretty simple: you just go through the workflow and the item added, sure, add another item, add more items, add shipping, and then order confirmed. And the cool thing here is: this will, each of these steps will dispatch an event. And on this event I have an event listener because this is my domain logic. I can trust that this event listener is executed each time an order is confirmed. And I just asked the delivery service to take care of this order somehow. Makes Sense? So using the state machine, I also know that no order will ever be completed or confirmed unless I added the shipping address. It's literally impossible to complete an order without adding a shipping address. So what I basically want to say is, it all comes down to this: Using the Symfony Workflow component will make your write fewer bugs. And I hope this short presentation of mine have given you a new way of thinking, a new way to relate to your problems.

Thank very much. Thank you.

I will share my slides on joind.in. And so if you, if you're interested in anything, you please feel free. If you have any questions, I think we have a few minutes and I will throw this cube at you. Anyone want to have a cube? There's a question over there, please help me. Two rows back. Three rows back. Did you activate the cube? It was a mistake. Oh, okay, mistake question. Someone else - just throw it to someone waving over there. Whoever has the laser pointer in their eye.

Sorry, can you hear me? Okay, great. So a question is really simple: where is e-commerce here?

Where's the camera? I don't know where the camera is - oh there's the camera.

I'm saying this was callex using the workflow component for E-commerce.

Ah, where is the e-commerce? Yeah. I assume that you missed my five minutes beginning. Michelle sadly got sick. So I did a stand-in talk about Workflow component. So yes, I missed a lot of e-commerce stuff and we can only blame Michelle's three year old kid.

There's one more question and then I'm gonna let you all go.

You hear me?

Yes I do.

Uh, so my question is how to properly connect workflows with the security. So you have two ways: you can write a guard listener, uh, to prevent some transitions based on security. Or you can, as you showed, you can do a voter and there decide based on workflow whether you can do something or not. So what's the correct way?

It depends what you're trying to do. I show the voters, which, I will try to explain, I used for handling my app security. I used the state machine as a substitute, as a help. If you're going to add security, saying like you want to make sure... if you want to make sure that you have the correct role for doing such as this action, use the Guard event. Okay. So you do something like this.

So both ways are correct?

Both are correct depending on what you're actually trying to solve.

Okay. Thank you.

I will be around for the rest of the conference and I also told you the best speakers in the front. I'm doing a second talk at the very, very end of the conference for some reason. So if you're interested in Symfony 4 internals, please come and visit me then. Please review, rate all the speakers on joind.in

- it's very helpful for all of us. Thank you very much.

Chapter 3: Behat Best Practices with Symfony

Tip

SymfonyCon 2018 Presentation by Ciaran McNulty Behat is widely used as part of a Behavior Driven Development lifecycle, but it's also widely misused.

In this talk I'll explain the BDD process, and show the best practices for using Behat including: writing good scenarios, driving service development from scenarios, fast UI testing, and using Behat and the Symfony2Extension.

Hi everyone. Welcome. Let's get started. Maybe we can close the doors. Hi. I'm Ciaran, and the subject of this talk is Behat and the best practices around Behat. I'm a consultant around areas like agile, quality, communication in teams being productive and most of the time the thing I introduced to organizations that helps them the most is this thing called behavior driven development.

Business Problems vs Technical Solutions

So generally in software we start with some kind of problem maybe we express the problem as a project goal or maybe you express the problem as an epic. We've got some sort of business problem that we want to solve and we caught some ideas about how we're going to solve these problems and policies we're going to put into place some things we're going to let users do that they couldn't do before, some things we're going to stop people from doing.

We have we, we generate some kind of business rules. And then at some point we implement technical solutions that implement those business rules or enable those, those new policies and we can launch the product and real people can start using it. And as we go through this progression from that side to that side, when you're looking at it that way, the cost gets higher. So the cost of each of these things increases. Having a ... identifying a business problem is cheap. Coming up with some ideas for how we solve it. It's relatively cheap. We have to talk to people and then technical solutions and building stuff is expensive. So shifting left is about maybe trying a lot of BDD practice is about trying to test these ideas earlier, testing before we commit them into code. So a lot of it is about communication, so it's called behavior driven development, Behat is a behavior driven development tool. The name comes from behavior testing, but that's not documented anywhere that I know about. So it's a secret for you.

BDD: Talking about what the System should do

So, what's behavior Liz Keogh says BDD is when you use examples in conversations to illustrate behavior. Just about having a conversation. Number one, difficult for developers sometimes. So we need people in the teams who can enable this kind of process and help break down some barriers. We need to talk about behavior or define behavior in a minute. What, what should the system do, you know, not, not the technical details, how should it behave in different situations? We use concrete examples. We give very specific examples of how it should behave in different cases or talk about why that's important.

Examples

So what's an example? An example is something that illustrates a rule. So maybe the important thing is understanding business rules, but the idea is we give examples as a very efficient way of making sure everyone understands them the same way. So naturally in a conversation if you don't understand something, to ask for an example, a concrete real example. So, we've got these business rules we're going to implement and we're going to illustrate them with examples. Examples are going to help us understand the business rules, help us check that we all are aligned on the same business rules. They have some value in themselves, but mostly it's to make sure we all, we're all talking about the same thing.

So here's a business rule. We charge our customers sales at a tax rate of 20 percent. Quite common in different countries to do this. Put your hand up if you understand this rule. Someone wrote that in a user story, you'd kind of have no further questions, you can just implement it. So, but there's an argument that sometimes a business rule is so obvious that we don't have to really have a conversation. But an example is a really cheap way, even with a simple rule of just validating, yes, we understand. We really did understand it. So I might ask the developer who's from the US and they say, yeah, I understand this completely. So something's price is $10, we charge $10 and we had $2 tax. So we take $12 from the customer.

Put your hands up people if that's what you thought it meant, not everyone, right? So even in a simple business rule, we can

have a misunderstanding about how it works. We'll come back to that. So the person on the left has, has this idea in their head, this rule, and if they just expressed the rule, they explain the algorithm or the general policy that covers lots of cases it's possible for the person on the right to have a different, entirely valid understanding of the same rule.

If instead they give concrete examples. So this is the rule and that means in this situation, this is what will happen. It gives the other person, an opportunity to kind of run those concrete examples through their own understanding and say, that's not what I thought would happen in those situations. Maybe my model's wrong. Maybe I understand your model. And the best thing that can happen is then we have two way communication. They say, okay, so in this situation, this, you mean this is what will happen in this situation. So both this is a kind of back and forth, but both parties look at the concrete cases and they say those concrete cases really match with the way I understood this thing we're talking about. So that means I think the thing in my head is the same as the thing in your head. We can't validate that directly without telepathy, but concrete cases are giving us a way to sort of check this.

So, in my sales tax example, if the developer gives the example, I can say, actually, no, that's not what I meant. What I meant is if something's priced at 10 pounds, we charge 10 pounds. We just remember that some of it was tax because in the business where we're operating in the UK, all prices have to be inclusive of sales tax, not, not exclusive of sales tax. And I gave this example in another country and they had a different system where some stores showed it inclusive and some stores, some stalls showed it exclusive. So I built the system this way and invested my time and effort in the code implementing this business rule. Or maybe it would only find out when someone tests test the software, does some user acceptance testing because maybe my QA team had the same understanding I did. We can validate that earlier just by giving an example, give us an opportunity much earlier in the process to, to catch these understanding mismatches or, I'd say requirements mismatches. But we're not on the same wavelength.

So examples are about behavior and you can kind of, um, a really sort of computer sciencey way of looking at behavior is there's an input and an output. So something happens and there's an outcome, there's an action, something that causes behavior, there's an outcome that's the result of the behavior. I buy a pair of Levi 501's and I'm charged 32.99 in and out, right? So, then maybe there's something missing here. It's not obvious why that action leads to that outcome. It's not really obvious why that product leads to that price.

We need this other thing called context. Something that happened in the past is a good way of thinking about context. You can express context as like a state, the state the system is in. It can be really useful to think about context as what are the things that happened in the past that are affecting this outcome, especially if you're doing event driven stuff. So what happened in the past that means I'm charged 32.99. What happened was that they were listed it in the system. So someone administrator loaded that product into the system and gave it a price, and now that means when I purchase them, I get charged this money. Those three things are really all you need. There's some sort of triggering event. Triggering action is what should happen and there's, you know, the minimum necessary context to understand why that causes that. So when you're talking about examples, it's quite easy to capture them in, in something like this format, you know, little notes when you're talking about things.

Examples Lifecycle

And examples go through a kind of lifecycle. Early on when we sort of starting to think about our business rules, we can use examples as a way of discovering potential rules as a way of being quite concrete about the system. Coming up with different options, diverging, brainstorming. We can come up with lots of examples of how the system could operate. At some point we're going to start narrowing down which things are we actually going to implement, which things are we going to implement first? Then the examples get more detailed so early on in their lifecycle. You can think of an example like the guy goes into the store and buys a pair of jeans and it gets charged the right price. We just write that down or even something shorter, you know, Dan North came up with the Friends episode naming where it's the one where he buys the jeans and you can just sort of stick on a sticky note and keep it somewhere later on in their lifecycle when we're closer to implementing them technically we write them out in more detail and as we'll see, we're going to use them with Behat. We're going to write them out in a formalized way that the machine can read. But you know, think of that as an, as an evolution of an example as it becomes more likely that we're going to implement that example in our, in our project.

Context Questioning

So once you've got some examples, you can enrich them by asking a couple of angles of questions that Liz Keogh, again named in a blog post. I come back, I come back to this naming because it's a good reminder when you're having a conversation. Have I asked about the context, so is that, is there another context where this action leads to a different outcome?

So if I've got this example here, I can say, is there any situation where I could buy these jeans and I wouldn't pay 32.99? That's a great open question. In fact, that's not a very good open question. It should be what are the situations where I would buy these jeans and pay a different amount? Don't give an opportunity for no. What situations are there? So as soon as you

start having this conversation with a real person or a real group of people and they're thinking about, you know, because it's an example, we've got a real real product, a real amount of cash, and they think about the real situation. They're not thinking about specification land. They're thinking about a real store and then they can come up with loads of stuff. Yeah, maybe they're on sale, maybe the person buying them is a member of staff and they get the discount up to a certain amount. Maybe the jeans were damaged, and they're refunded to stock. So in those cases, what are the different outcomes? This is a whole discovery process of coming up with new examples, new requirements we didn't know we had and it's great that we're discovering them now rather than after delivery.

Outcome Questioning

The other kind of approach Liz advocates is outcome questioning. So we've got the context, we've got the action, we've got an outcome. Are there other outcomes we have to think about? So in this Levi's example, apart from me being charged some money, what else happens? Again, this is where the flood of future user stories, future requirements sort of cracks open the can of worms opens. I'm charged 32.99 and I get a pair of these jeans sent to me, dispatched through the packing system and the warehouse, has to know that we've got fewer jeans than we had last a minute ago and probably a load of other stuff, you know, an invoice gets generated and the marketing stats get updated and blah blah, blah, blah, blah. We discover the huge scope that we didn't know existed. This is the discovery phase is where you're having these conversations. So you're trying to do this early in, in a very informal way.

Example Mapping

And one way of having these conversations that I do recommend as a starting point is example mapping. It's a sort of workshoppy format involves sticky notes on a wall. It's easy to get people doing it rather than sitting everyone sitting around typing things into a Google docs. Or God forbid, a Jira ticket on a big screen, around everyone sits around the table. It's good to get people moving and doing stuff. So Matt invented an example mapping three or four years ago and blogged about it. There's lots of other ways of doing this kind of workshop, but this is a great formula. So you take the feature you're talking about and you stick it on the wall. In this case, it's calculating the price of a shopping basket. We've realized that customers like to know the price before they pay. So we're going to talk about that feature and you try and figure out what are the business rules and you lay them out in this horizontal dimension.

So we have to apply sales tax, above a certain amount you get free delivery and then as you're going through the rules, you try and think of really concrete examples of each rule. How many examples do we need to give for someone to understand how the rule works? If you have VAT being applied, I can probably come up with an example. If you buy this specific product, this is how much you pay in tax and maybe don't need any other examples. I don't need to generate lots of different examples because it's a simple rule, free delivery above a threshold. If you pay 50 pounds, you don't pay any shipping. If you pay 10 pounds, you pay some amount. I don't have to put all the detail when you're sticking these on the wall, but kind of identifying which examples we're going to have to go into detail on later. We might have some questions. So someone in this conversation says like, what if you're shipping to Belgium and our stores is in the United Kingdom? What happens then? We can capture the question as part of the example map and actually what happens, what happens with tax and what happens with delivery. We don't know.

If someone knows the answer for delivery, they can say, okay, overseas shipping is a fixed rate in each country and we don't have this concept of a maximum amount where you get free shipping. But maybe we end the session with a question, how does tax work across borders? Maybe we have to ask someone else who isn't in this session before we can proceed as a good example mapping.

I spend about 15 to 20 minutes talking about a feature. More than that then you're probably going into more detail than you need to. The questions at the end are, can we start work on it without this answer? Maybe that's something we trust we'll find out during the sprint, you know, or do we need to resolve it first? Do we need to get an answer before we say we can work on it. More interestingly, can we do part of this feature that doesn't involve the question and a nice way to split is against these rules so we can potentially take this overseas shipping and move it into a new feature.

I'm going to say, okay, well we can build the shopping basket pricing. It'll only work in the UK and if you select the different country, we just don't show you the total. We'll do that next week. This can be a really useful way of you know splitting down stories. Example mapping is one technique. You can do it one on one it's good to do it in a, in a group. I quite like doing it. If you've got a few features to talk about I quite like putting them physically in different places in a room and people rotate around them in a round robin. It should be quick. It should be scrappy. You're not getting into loads of detail on the examples. You're just kind of identifying hey we need to have an example where someone gets free shipping.

Feature Mapping

There are other techniques that are worth looking at. Feature mapping is very similar to example mapping, but it adds a kind of time line concept. So you have business rules, examples and then steps in the process. And of course if you're doing

event storming, they don't do event storming? If you're doing event storming off of the output of that as well as being super useful. You can zoom in on sections around the commands and say, given these events in the past when this command happens, these new events should happen. You can sort of extract scenarios from that.

BDD is not about Testing

So let's talk about BDD. That's some of the most important stuff in bdd is discovery. That's where you should start if you're implementing BDD. It's not really about testing, it's also not about one way conversations. I'm not suggesting that your product owner does this kind of example mapping session, identifies a load of examples and then types them into Jira and then delivers it to a team, who's never seen them before. It's about having multiple stakeholders inside those conversations.

Validating Examples

But where it comes into testing is that once you've got these examples, and we generated them so everyone understood what we're building, they are super useful for validating features. Once we've built the feature, we've now got a set of concrete examples of how the system should behave. Even if we're doing manual testing. That's really useful. I know that if I buy an item that costs 10 pounds, I should be charged two pounds sales tax. I can go into the system and buy something that costs 10 pounds and I check, I get charged two pounds sales tax. So once we started to build the technical solution, we're going to write tests and these examples are going to be very useful to create test cases.

This is where Behat fits in. We've had a lot of rich conversation conversations about what we're going to build. I'm going to write some automated tests and there's a, there's an overlap there where a tool that can take written examples and help you automate them into test is going to add value. First tool like this was called cucumber in Ruby. Every language has a tool like this. We tend to call them cucumbers, Behat is sort of retaining it's own identity because it's a completely separate code base. It was never like a port, but we collaborate a lot with the other cucumber tools or we make sure their syntaxes are interoperable and stuff like that.

Real-World Examples

So I wrote out an example. You don't need to be able to read this. I write out an example in a format called Gherkin because it's from cucumber. There is the same format we were capturing on the cards. You know, the three steps, the context, the action, and the outcome. It's just written out in a way that is machine parseable, but it's meant to be business readable. So a human can read it as if it was a document.

Let's zoom in. The problem I'm talking about is scheduling training. I do a lot of training and either we have a training course with not enough people, so we need to cancel it or we have too many people. So as a trainer I should be able to cancel courses or schedule new ones and I can identify the business rules. This stuff at the top of the feature file is just human readable text to explain to a person.

So the rules are, when I make a course, it has size limits, there's a sort of threshold where the course becomes viable, and when we reach a maximum class size, you can't sign up to that course anymore. And then I come up with some examples and I write them out. The Background is context that's common to all of the examples in the file. So we talk about context using the Given keyword:

> Given "BDD for Beginners" was proposed with a class size of 2 to 3 people

So that's the context. The first example is, a course doesn't get enough enrollments so it's not viable. When only Alice enrolls in the course, then this course won't be viable, easy to understand. Hopefully.

This is an example where the course gets enough enrollments. So Alice is already enrolled in the course, that's the context. When Bob enrolls, that's the action. Then the course becomes viable, the course will be viable because it was proposed to the size of two to three. And enrollments have stopped when the class size is reached. Given that Alice, Bob and Charlie have already enrolled in the course, when Derek tries to enroll in the course, he can't enroll.

It costs something to write them out in this detailed format and think about how much detail you're gonna include. So you have a kind of scrappy conversation about examples. When you're going to build the system soon you write them out in this kind of Given, When, Then format and you show them to the person you had the conversation with and says, this is what we talked about. So whoever learned the most in the conversation can write out these scenarios, so this is what we talked about in the room. Please validate that. Let's just double check. You don't do this, you don't write them out in this detail in advance for everything.

How Behat Executes Examples

The job of Behat is to take these multiple feature files which have steps - a step is: Given, When, Then - and somehow

execute them as tests. And yeah, that's hard to execute as a test. It needs some help. So the help is that we have a bunch of classes called contexts with methods in, called step definitions. Each one of these step definitions tells Behat when you see a line like this in the feature file, this is how you test it.

In the Gherkin file, Given a thing happens to Ciaran and then in the PHP file I'll write a method with a doc block that has a pattern in it. Then Behat will know, whenever I see a line like that, I should execute that function. And you'll see the function is parameterized. So given a thing happens to Bob, we'll also hit the same function. And I have to write the test code inside that function that's going to check the system.

Testing through UI vs Domain Model

The way a lot of users interact with your system is through a UI so it can be tempting to test through the UI. It's normally a much better idea to test through a domain model.

So why is that? Testing through the UI is going to be slow, you have to automate a browser, things break, web servers time out, that kind of stuff. And if I just test through the UI, all that tells me is that the UI supports all the business rules. That means the business rules could be living inside a controller or the business rules could be living inside a Twig template because we didn't put the button on that template and that means people can't do that thing. We probably want to check that the domain model is supporting these things. So let's see how that looks.

Testing Domain Models

So by driving PHP objects directly from my scenario, it's going to prove my php objects, implement the business rules correctly. It's going to give a bit of a pressure to use the same naming from the feature file in the objects because I'm looking at both at the same time. And I'm not going to pick a different word when I can use the same word. And it's gonna run fast. It's going to run almost as fast as your unit tests because it's just running PHP. So when I execute that feature file in Behat, you'll see they all get this nice, maybe it hasn't come across, nice yellow color and maybe you see at the bottom everything is undefined. So Behat is telling me, "you need to tell me how to test all of these steps.".

So I go through my feature, I start at the top, here, I look at the first one, this one. Given BDD for beginners was proposed with a class size of two to three people and then I write a method in my context class annotated with something that's going to match that pattern. And I think, what do I want this to look like? This is where I start, this is the TDD like loop I go through. I'm going to start here. So there's a thing called a course. So I'll make an object, I'll pretend there's a class called a course. I'm going to give it a constructor called propose() because the word propose was in the example I wrote. And it's proposed with a class size of between a minimum and a maximum. So, I kind of create, I guess course is an entity and class size is like a value object.

And I want to execute it. I get an error because none of these classes exist. I kind of started here, I'm using the domain language from the example in my code and I like to see the red.

If if the first time you run the tests, it's green, there's a lot of things you could have done wrong, so it's always good to run the test and see them red and then you write the code that makes them go from red to green.

So, I have to do some stuff here. It's a short talk so you have to make these objects and you probably write some unit tests for those objects as well. And you know if you're super advanced you write the unit tests first and all that good stuff, but at some point I've written a course object that has a constructor and I've written a class size object that has a constructor and I did some extra thinking about what do I need to check inside class sizes. Maybe I'd check that max is bigger than the min. So, there's extra stuff I have to think about as I'm building these objects, as soon as they exist Behat starts passing.

So, the next step only Alice enrolls in this course. In the previous step, I sort of stored the course that I was creating as a member variable in the context. The next step, Alice enrolls in this course, I'm going to enroll a learner on a course. Um, there's an example here of a transformation. So Behat does support very simple transformations from strings to value objects, especially if we're going to be using the same sort of values in a lot of your examples. So whenever I see a name, a learner, I'm going to turn the string into a learner object. The course, gets an enroll method. I have to then add an enroll method to the course that takes a learner object. I have to create a learner value object, write unit tests, blah, blah, blah. Then this step will start to pass.

Then the course will not be viable. So I need to write a step that checks that and I'm using an assertion here, I'm using a PHP assertion. Any step that doesn't throw an exception is considered a pass. So in the Then steps you have to check something, Then the course will not be viable. I have to tell Behat this is how you check the course won't be viable. You can use a different assertion library. You can use the PHPUnit assertion library, that has a bit of extra support than Behat because we know how to get the better error message out of those kinds of exceptions. Or you can have an if statement that throws your own exception. It is kind of up to you. So I'm asserting this is viable method that returns false. I have to then write some code, some logic when I've done all of that, my first scenario is passing. And actually that's quite a good point to commit.

I built all the objects and implemented one of the scenarios and working per scenario is actually very nice as a developer. It's a good way of not trying to think of everything at once. So what's next? I do all of the other scenarios, I work my way through in the same sort of way. So you can kind of imagine Alice is already enrolled in the course. I call the existing enroll method. Bob enrolls in the course I call the existing enroll method. The course will be viable. I assert something slightly different. I assert it's not viable. And, you know, the third scenario, Derek tries to enroll in the course. Maybe I catch an exception, then he should not be able to enroll. I maybe check the exception got thrown, or I check the course doesn't have Derek as a as an attendee. So I validated all of this inside my objects.

That doesn't take long. If you look at the execution time and the screenshot, maybe that's a lie. It's twenty it's still the same 20 milliseconds to run those three scenarios. That's not unrealistic. They're like large unit tests. There aren't, there's no infrastructure involved, nothing crazy is happening. We're just checking that these objects have the business rules in them and that's a super powerful place to start. Actually talking about what the business rules are, is a really powerful place to start. But checking that the object objects implement them on their, on their own and it's not, you know, in a stored procedure or a controller or somewhere else is really going to help when you have to write a command line, a cron job that also enrolls people, for instance.

Testing Services with Behat

The next layer to look at is the service layer. And I'm not specifically talking about Symfony services. Quite often you have some layer in between your domain model and your user interface, and in the Symfony case it is Symfony services. So when, you probably don't want to be interacting directly with these domain objects too much in your controllers you want to delegate it into services.

So, we, we might want to inject our services into the Behat context. As soon as you go into a Symfony app and you start thinking about your Symfony services as. So indeed, you would try and have your own concept of application services and not tie it into the Symfony service container, but a lot of apps, it is tied into the Symfony service container. So it's really convenient to have a way of accessing our services our defined services from inside Behat. And then trying to use the services rather than using the domain model directly. This is going to make sure that our service layer supports all of the use cases we want to support.

Even though the domain model is quite complex, we can have a defined set of use cases. These are the things that our system supports. And you can test the same scenarios in multiple ways using Behat, using a concept called suites. So we wrote one context that told Behat how to test every step directly against the objects. You can define a separate suite that tells Behat how to test the same steps against a different layer of your application, in this case services.

So it looks slightly different. It's the same process for me when I want to implement this first step, BDD for Beginners was proposed with a class size of two to three people. Instead of creating, instead of creating domain objects directly, I'm making calls to services. So I'm saying course enrollments, propose a course with a string name and the minimum and maximum attendees. So I'm kind of assuming that course enrollments is available. There's a service called the course enrollment service.

Symfony2Extension

Where's that come from? You sort of see it, it's arriving magically in the constructor. And this is then becomes framework specific. How is Behat accessing that service, the Symfony 2 extension, which needs to be renamed but it was a really good idea at the time to call it that, will inject your Symfony services. So, it takes care of finding your kernel, bootstrapping the kernel, telling the kernel that it's in the test environment. You can override these things. And then when we, instead of just listing the the object, instead of just giving the name of our service, our context object, we also can inject param, inject services as arguments to that context. So in this case I'm saying there's an argument, there's an argument called course enrollments in the constructor and this is the service you should get from our container. I'm very old fashioned so I'm not using the class name yet.

And this can be done in other, in other frameworks using different extensions. And now that PSR containers are standardized, there is an extension that will just let you directly plug any container in including the Symfony container. But it's best to use the Symfony extension for this still because it will do things like rebooting the kernel in between steps.

So I need to write a service and the service. This is, this is the service object inside it, it's using the domain objects I created earlier. So it's a way of defining a sort of course API for my domain model that the controller is going to use. The line in the controller is going to end up looking like this. It's just going to call a method on a service with some primitives, with some scalers. At this point you kind of need to get infrastructure involved. You definitely need to think about persistence. Using real infrastructure is super slow. Using real database is slow. So it's best to use fake implementations or infrastructure, like an in memory database that can lower your confidence because you think I'm not testing with a real infrastructure you can use a thing called contract tests to make sure your fake infrastructure behaves the same way as your real infrastructure to kind of mitigate that.

So I run through my other steps when only Alice enrolls in this course. So when enroll Alice, the course will not be viable. I'm not looking at the domain object directly. I'm calling a method on the course enrollment service, so they're kind of two different options. I wouldn't recommend testing the same scenario against the domain objects and the service layer, but something I often do is I will write my tests directly against the domain objects and then I'll switch and start testing against the service that wraps those domain objects, if that makes sense.

So I kind of deliberately tests at different layers because I want to be able to completely change my domain model without the tests failing. I want to decide, course isn't an entity anymore and actually it's all about learners enrolling in tracks and that doesn't need to be visible at the service layer. That doesn't need to involve any changes at the service layer.

Testing the UI with MinkExtension

At some point you want to test the user interface, but not too much. Don't start by testing the user interface. At the moment the best way of doing this is the behat/minkextension. Mink is a API for driving lots of different browser automation tools using the same common API.

We interact with the domain model via the UI and this checks, once we've got service layer that works, we're basically checking that the controllers are calling the right services. Think about it as just just validating that the UI works. Don't do this kind of stuff. If you start with the UI layer, you'll end up putting loads of UI crap in the, in the Behat scenarios, in the Gherkin scenarios. It's not the end of the world, but you're missing a lot of opportunities to, you can't have this conversation early. And it's very brittle, when you change the UI these things just break.

So to test, using mink, I enable the Mink extension. I already had the Symfony extension, I can tell it use the Symfony driver. This is gonna automate as if we're doing browser automation, but it doesn't require a real web server. It's just booting the kernel and sending a request and checking the response. You can also use it with PSR there's a PSR 7 driver they made. So it looks kind of the same. Again BDD for Beginners was proposed with the class size of two to three people, I'm still going to inject a service and I'm still gonna call that service, but maybe this is the service that's connected to the real database, the real, real test database, the real MySQL test database.

When only Alice enrolls in this course, then I need to write some browser automation code. So Behat checks Alice enrolling on a course in a different way. Instead of calling PHP methods, it's going to open a page and fill in a form and submit the form. The course not being viable, instead of checking a boolean on a, on an object, we go to the page and we say, does this element, this CSS element exist?

Now I don't need to do this for all of the scenarios. I'm kind of confident that if I've got 10 scenarios about different situations with different combinations of people sign up and then unregister then register again, and I've checked all of that at the service layer, I probably only need to check a few of them through the UI. Specifically the ones where the UI would do something different. So I need to check one case where the not viable warning appears. I need to check one case where the, this is oversubscribed thing appears I don't need to check all of the combinations of what could happen because I've got confidence that the system, the underlying system works. I'm just checking that the UI is plugged into the services correctly. So this can all be tested through Symfony directly. It runs on your laptop, it doesn't need any extra infrastructure.

Testing JavaScript

Some people automate real browsers, mostly when you've got loads of JavaScript and you've written a site that doesn't work without JavaScript and you have a frontend team who aren't doing any testing any automated testing. So it's very slow to automate a real browser. They crash and do weird stuff.

It's often better to use Cucumber.js and do a sort of layered approach like I've been describing in the JS application. It's better to do that then try and drive the entire system end-to-end with a real browser because that's how you end up with four hour test builds.

The best way I've got at the moment of doing this is using the chrome driver. You can pass some extra stuff to tell it to run headless. This is actually an experimental extension by this person, who I don't know, that seems to work fast. It talks to chrome across the debug port, it's faster than selenium for me, but it's a bit flakier.

And we used to have a lot of systems like PhantomJS that would pretend to be a browser. But if you've got a recent Chrome, you've got a good browser automation tool already. You can run Chrome in headless mode, you can open up a debug port and Behat can talk to it directly. You don't need to install selenium if you don't want to. Maybe into CI and stuff you can use selenium, but this will work directly and it's in the background and you don't really notice it's running or you know, the docker world now so you can just get a docker image that has a headless Chrome in it and it boosts itself and then you can kill it afterwards. So browser automation is much less painful than it used to be.

Summary

So start by, start by having conversations, try driving the domain objects directly. Refactor them into services when you feel like the domain objects have got to a good place and test with the service layer. Then test a few scenarios through the UI. If you can't avoid javascript entirely, it's good advice for life.

Future: Autowiring, Panther

So future: currently you can't autowire services into your Behat contexts. The API is there inside Behat for that to work. We just need someone, some brave person to make it work in the Symfony extension. I don't think it's impossible, there's stuff there in Behat for figuring out what to inject for specific argument, someone needs to hook that into autowiring in the Symfony container. So if anyone wants to volunteer afterwards talk to me.

Symfony Panther is awesome. Behat uses a thing called Mink, which is an API that's common to all of these different browser testing tools. But in Symfony land we already have a common API now across browserkit, Goutte and panther. It'd be really nice to have a way of like a convenient way of using that directly inside your Behat contexts and use the web test API that you've maybe already know instead of having to learn Mink. And there's a lot of convergence across cucumber tools so it's possible that it's possible, Behat will become more like the other cucumber tools by sharing code. There's a project to maybe switch a lot of it over to a shared go run time that we haven't signed up for. But a lot of the other cucumbers are doing it. So we might succumb to peer pressure.

Thanks for your attention. I went slightly over but we started late. So I think that's okay. I'm Ciarran. If you want to talk about any more of this stuff just grab me. I can come and help you do it at your company. I look after another tool called PHPSpec, which is unit testing. Check that out. There's two meetups I'm involved in look at them and if you want to see a working example of the code, I was showing in the slides. That's the URL on GitHub. Probably don't have time for questions, so enjoy the break and come and talk to me. Thanks.

Chapter 4: File Storage for Modern PHP Applications

Tip

SymfonyCon 2018 Presentation by Frank de Jonge

Many PHP applications need to store files. There are many different reasons to store file, and not all filesystems are created equally. How can we identify our needs? Which filesystem is best for your use-case? How can we ensure our future needs are not blocked by the code we write today? In this talk we'll explore how filesystem abstraction and general coding guidelines can improve our application's and make them future proof!

Hi everybody. Today I'll be talking about file storage for modern PHP applications. My name is Frank. I've got a couple more people dropping in. So thank you all for joining this talk. The other two talk slots are one improving workflow and the other is a good number of best practices for awesome, testability. But somehow you've managed to find the room where we're talking about file systems.

Who is this Frank Guy?

Uh, I'm a freelance and a free software developer. That means I work at companies now. I can't name the company that I worked for, but it's the biggest airport in the region of Amsterdam. So if that gives you any indication then uh, then you know where I work. Uh, so, uh, as in terms of free software development, I like to contribute to open source, for Symfony because this is the Symfony conference. I thought I'd highlight some of the things that I did. So you may be familiar with who I am. Who remembers their routing speed up in 3.2. I did, I had a lot of fun coding it up. And then Nicolas came along and basically ripped out all my code. Uh, so thank you Nicolas for that. Um, but then I thought, okay, I really liked this routing component. So for a 4.1 I added the internationalized routing. Who here uses that One?

Okay. So the rest of you are just stuck with code that I wrote that nobody uses. For other stuff that I did. Are there any people who like to do event-sourcing here, right? So if you like event sourcing, I created EventSauce in over the course of last year as sort of alternative way to approach event sourcing that's a little bit smaller and possibly more easy to understand. So if you are into that, maybe check it out.

Hello Flysystem

But for the majority of my time I've been working on Flysystem, which is a package of the PHP league. Who here knows the PHP league? Okay. Decent number of people here. Who here used Flysystem itself? That's the same people. All right. So, um, I started five years ago and I started working on this presentation like a couple months ago. And at that time we just hit the 50 million installs mark, which was pretty great, and now we're already close to 55 so it's going quite fast as you can see, it's a, it keeps on growing, growing, growing, and for the most part, working on Flysystem has been a pretty good experience, uh, but also you get to basically see all the stuff that people do with your open source work and also what they do in real life.

And sometimes it's a, it's a bit scary sometimes it's a little bit depressing and when it's depressing, it's mostly PHP because or FTP because FTP is not so nice. Um, but for today we'll be talking about file storage for modern PHP applications. A note of warning this talk may not be exactly what you expect it to be, but I hope you can keep an open mind in the things that I'm gonna propose because they're going to be a mix of different things, both code and not so much code things. Um, so, uh, if we're talking about file storage for modern PHP applications, what should we do? Well, it depends and that's the problem. Uh, so today I'll be talking about a few different subjects. One is just to get a generic gist of what file systems are, what types we have, so on, so forth, um, how they interact with modern applications, what is involved in the process of choosing a file system and how you consume filesystems in code.

The Problem: Who put these Files Here? Are they Used? What!?

So this talk is not so much how to do it right it's more about how not to do it wrong or even if we're doing it wrong, how to do it wrong the right way. So, okay, you're with me. Uh, so what's a filesystem? And I can already hear you say "lol don't care". It's something that we have everywhere and when I see this, when I see this, everybody looks at Buzz but I tend to feel a little bit like Woody, because filesystems is one of the things that we sort of all have problems with or something is always full disks or I don't know what to choose, but it's also one of the things like nobody really takes it into account as a decision process. It's like, oh, we have this, we'll use it. We don't tend to think about it anymore, but we do have a lot of common problems, common problems like the unstructured filesystem directory.

Who here has or has had one of those unstructured uploads directories. Alright. Uh, let's, let's keep the hands up. Let's see if we can get all the hands up at the end of the questions. Um, who's ever had the question like, are these files used? Okay, can I delete these files? All right. Where did these files come from? Like, okay, people already put their hands down. That's okay, let's go, let's keep them down from now. It's like if we ask ourselves all these questions, we're pretty sure we don't exactly know how to clean this stuff up anymore. And then you can get into a situation like this. It's like that's, that's basically what it boils down to. Right? So, and other questions like can we change filesystems? Like sometimes there's reasons to do that but it's not always possible. So for me it always looks like, okay, this is fine, but is it fine?

Well, I don't think so. And I think mostly it has to do with the feedback loop when we're consuming this stuff, we've got a timeline, and in this timeline we start out, we're going to create a feature. We've, uh, we've got it laid out, but all the knowledge that we really have is assumptions, then we create the feature and now we have experience with some tools and we put it in production and we get some initial usage. Uh, we get feature adoptions from our user and this is where the actual usage start, but also maybe when the actual problems start and then after awhile we detect the problems and we're going to solve them, then we're going to get actual experience. I really believe this is really a sort of a sort of depressing thing because it means that the information that we needed to have in order to make the right decision at the start is at the end.

What is a Filesystem?

So we're sort of doomed to fail. So, how can we make the wrong decision the right way? But before we go into that, let's first briefly go into filesystems. So what is the filesystem actually? Well, a filesystem is a system that controls how data is stored and retrieved. I've highlighted the word system there because it feels like there's something in there. So what is the system? Well, system is a collection of parts conventions and interactions forming a complex whole. Okay, so if we expand, that's what a system is into the definition of what a filesystem is. What did we get? Well we get this. A filesystem is a system that provides a collection of parts, conventions and interactions, forming a complex whole that controls how data is stored and retrieved. So in itself, although we take it for granted, a filesystem is already pretty complex. The definition itself already is pretty complex, so if we dumb it down a little, we have something you use to store and retrieve data. So it's something that we use, not something that we have. If we're talking about a filesystem, we're not talking about necessarily the storage units, for example.

Filesystem Types: Nested versus Linear

Then if look at the different types of filesystems that we have. There are many different types, but in a broad sense like how it affects us, there are two types, that we have to deal with. One is the Nested filesystem, that's what you have in your finder and if you're using or if you're one of the 12 people that use windows at this conference, you have something else. Um, but there's also a Linear filesystems. Who here has used the Linear filesystem? Okay. Who, who here has used anything in the cloud? Okay. This is same thing normally you see so more people then raising their hands. And most things in the cloud are actually Linear filesystems. So if you have something like S3 or Azure blob storage, that's basically a linear filesystem and what's, what's the difference? So Nested filesystems are very common.

Your desktop has one, it has directories which have files and more directories. On the other hand, Linear filesystems are like key value stores. So instead of directories, you just have full paths, they might have separators that look the same as a directory separators. But it's really just a sort of key value system. They only have files and if you know that, then, if you say what's common between Linear filesystems and Nested filesystem? So it's basically only the files, right? So what you can say is that directories don't really exist if you look at the common denominators. So, uh, they require a different approach and they're very cloud.

Filesystems in the Wild: NAS, CDN & More

So if we're looking at filesystem offering we see, or at least I saw something interesting. There's a bunch in the group that is file system, then that's a network attached storage. So if you're on AWS, you have a volume. If you're on a Digital Ocean, you have like the blob storage stuff that you can also mount into your VPSs. There's stuff like Gluster FS and Ceph and I think, I don't know if there somebody from the SymfonyCloud in here, but I think they're using Ceph. I see nobody from the Symfony. Okay. That's okay. I think, uh, that's the case. There's another group and that's basically filesystem and CDN. Then you get things like AWS S3, Digital Ocean Spaces stuff, uh, Rackspace Cloud Files and Google Cloud Storage, all those kinds of things. That is the filesystem plus CDN, and then you've got more and that's more filesystem plus sharing. So Dropbox, Box, Google drive, Microsoft OneDrive, those kinds of things. And then you have things where you can put files in but they're not really supposed to be filesystems but they happen to support it.

So like WebDAV, which is something different entirely. It's more the protocol of interacting with somebody er something over HTTP rather than being a filesystem. You have MongoDB GridFS, and I think I've got a MongoDB engineer in the room in the back... Oh, I've got two. And you can also use something like Postgres Large Object Storage. But if we flip this around, we see all those things and then we say, well, what's the inverse of this?

What is not a Filesystem?

What is not a filesystem? Well, a filesystem is not one of those CDN's. So a content distribution network is not a filesystem. It's something that uses a filesystem, but it is not a filesystem and the same goes for all these other things like a web server is not a filesystem, file managers like something that operates on. So there's a lot of things in this space that sort of overlaps with filesystems and they supply the same offering as filesystems, but they are not filesystems.

So for me that is sort of a problem. If we look at one thing and that one thing has two concerns, we tend to obfuscate those concerns and view it as one and that might induce misuse of a, of a product or in the use case that we have.

Stop Making your Filesystem do 100 Things

Ryan Singer in 2010 said this. So much complexity in software comes from trying to make one thing do two things. And I wholeheartedly agree with this. Uh, I don't know who knows the statement, but this is one that comes back at least once a month for me. It's like, okay, I should dumb this down. It'll be better.

So my version of this is stop trying to make a filesystem do 100 things and that's also sometimes what I see when you, when people use Flysystem, they want to use it for access control. They want signed URLs for different things. They want to expose it over HTTP. And all those things. They don't really belong with filesystems.

What a Filesystem Does: CRUDLI

If you look at filesystems, they're basically CRUDish, but I've let some space open at the bottom because you've got writing files, reading files, updating the contents of files or deleting file that's pretty CRUD. But there's also these two other things like you've got inspect and list. So inspect is more like getting the mime type of a file, listing is like getting what is in a directory. So it's basically CRUDLI not yet a term, but I've coined here now... So, if you use it now, you will have to pay me money.

Filesystems in PHP: woh

Um, so if we use, if you look at filesystems and PHP, I'm just gonna take a little sip water. Then if we look at the usage, it's, it started out it starts off pretty easy so we can very easily write to a file. We can read from a file, we can delete it, although now it's called unlink rather than delete and we can get the modified time of a file using mtime and that's an abbreviation of half the words. Maybe a bit confusing of them. We have things that are related to file info and that already gets really confusing because you now have to instantiate an object with the type of thing that you want to retrieve from it and pass it a location to the file methods in order to get the mime type. So pretty confusing already, but it gets more confusing because if you want to get the directory listing for a particular directory, what do you use well you've got glob, scandir and readdir, which takes a resource which it needs to retrieve from opendir.

Or, you can use the spl directory iterators which are awesome, but you get all these other types of objects back that you can't really serialize or anything. So you need to do some normalization. And like which one of these is better? As it turns out the last one is the most performant for the most cases in the most general thing. But who will remember typing that out every time? So like WAT. Oh, okay. So then, uh, if you're, uh, handling big files, you have to uh, use functions like fopen and fread and fwrite. Who here is comfortable using those functions? Okay. So definitely less than a quarter of you just raised their hands. Well these are pretty essential for, for the most part. And this is a relatively simple example where you basically, this is a copy, but then in php rather than just saying copy, but this does not handle all the error cases.

If you look at this and handling all the error cases, it actually looks like this. So it becomes very complex and very verbose and it's a lot different from a file_put_contents. Right? So, so WAT, why can't you just use copy? Well, copy, if we're looking at filesystem a copy just works locally. Those stream things, they might work for other things as well.

What does Flysystem do?

So as I mentioned for the last couple of years, I've been working on Flysystem. So what does Flysystem give you in order to help you with this? Well, it gives you a uniform API, which means this is how you interact with the filesystem using Flysystem. You create an adapter, you used that in a filesystem and then you use the filesystem to write. You never use the adapter directly. That's a, yeah, it's part of the pattern, adaptive pattern. Uh, it's really applicable here and you basically have basic operations for writing, reading, updating and many more things. Uh, but it also means that if you want to change, for example, to the AWS S3, all you have to do is change the configuration and all the usage stays the same. So that's pretty powerful. And if you are one of the people that have to use FTP, you don't need to deal with it anymore. As I said, I've taken the pain for you. You can just use this and get done with the job.

Streaming

The stream part in Flysystem allows you to remove a whole bunch of that conditional code. Basically you can, get a stream from a remote filesystem, uh, and pass that to a local one and if you'd need to, for example, change something that's on a remote filesystem, you can first download it like this, do some stuff, and then re upload.

This is a common pattern that you'll see when you're moving to remote file systems because a lot of the functionality that we need in order to operate on files require your files to be local first. For example, if you want to manipulate images, they need to be in your local dist. They can't use a stream from an external service. So this is something that you need to keep in mind. But that's, this is not so much code in order for you to get this working. So it's okay. Um, this also means that you can use two remote services to basically pipe so many information through. So here we're reading from Azure and we're writing to AWS and your thing to notice is because we're using string, we're only using a small portion of our memory at a given time, so this could be a re uploading from Azure to AWS, like a multi gigabyte file, uh, but even if your local php installation is limited to like the default amount of memory this won't run out of memory. So that's pretty powerful.

Configuration Root & Relative Paths

The other part that Flysystem gives is relative paths. Everything in Flysystem has relative paths. And what I mean with relative paths is the following. You have a configuration root, so if you have a filesystem and it's confined to stay in a root and every location that you provided is a relative path from that root. And it always needs to reside there. This means that if you change the root, the configuration changes, but you're how you're using it won't change. So this is a pretty powerful concept because it's a very strong encapsulation principle. So if you're using the local file system and you have that first path, so it's in /configuration/, that's where you store all your files and that's just basically how you set up your services. The configuration is there, and then when you use it in your other services or inside your services or maybe directly in your control or whatever you like, you can basically use the relative path and Flysystem will make sure that they're combined and actually written to the appropriate path.

This means if you for some reason need to change the location of your files. For example, if you're running out of disk space and they attach a new network attached storage that is bigger or automatically responding or whatever fancy thing they've got configured for you. All you do is change the configuration. This means all your consuming code will forever remain the same and that's pretty powerful, but you can take this even further because it even applies the same concept over multiple filesystems. So if you're moving from local to something that is in AWS, all you're consuming code will never have to change.

And this, this is pretty powerful because it prevents vendor locking. Basically, if all your code knows is that all the files are going to AWS and you want to switch to something else, but now you have to basically refactor half of your codebase. Uh, I don't know any manager who is like, yeah, yeah, sure, spend the hours on that, that's amazing. They're going to prioritize something else, but what if that expense is lower, close to zero, then you've got a better opportunity to actually push for that change and reap more benefits from that move. So it is better for isolation, better for encapsulation because all the responses are exactly the same from every adapter. It helps with predictability and that helps to stabilize software. Uh, and in the end that makes sure that everything is super portable.

When to use, or not use Flysystem

But don't use Flysystem for anything. If you're using local only, for example, for templating engine looking up files or a, if you want funky things with symlinks, then you will only ever do that on your local filesystem. So don't use Flysystem for that. Then it's more of an overhead, you'll be limited. Because the limitations are functional for file storage, but for at least local specific filesystem only things, you'll just be limited. So keep that in mind. So if we're then going into the process of choosing another filesystem, we often use the FEYNMAN problem solving algorithm and actually this is more than people do already. There the algorithms says write down the problem. Well mostly people just think of the problem and then they think very hard and then they come up with a solution. But I don't, I don't think that works very well.

If we have somebody who uses our application and they want to upload a file, um, we the browser has to file uploads through us and then the file is there. If you want to use the local file system for that, that's fine. Still many people who are developing or deploying applications, deploy to one machine only. So if you put a well configured nginx configuration for that one web server, you basically have a CDN and there's a lot less hassle and moving around, that you have to do so. It's always good to know if you're not restricted by this, you can use this option, so like the local file system stuff is the majority of the stuff and then we are in the cloud space. And when do you want to go to the cloud space? Well that's mostly when you have that image that you want to upload, but instead of one web server you now have three.

And if you upload the picture to one and somebody else wants to retrieve the picture, well how do they know they're going to get it from that one server? Probably there's a load balancer in between, but that should be not sticky and not knowing of the implementation of the web servers below. So the generic solution for this is pretty simple actually. You just make sure they share storage. So you can do this in various ways. Um, obviously the file goes there. Um, so you can do this either by using a service or using something that shares the filesystem across many things. So you can use a clustered type of file system, like Lustre fs or something else in order to orchestrate this. So you need to know that the storage now becomes a separate entity

that multiple, instances of your application are using this singular entity and whether, if it's a mounted filesystem or it's a service actually doesn't matter. So why do you want to do it as well if you want to scale horizontally, just as a quick poll. Who here scales horizontally their applications?

Okay. The other way around. Who deploys their application to one server?

Okay. So there's, I didn't see all the hands going up or down, but it's ok. Still a good number of people have that. If you don't need to do that, you may also not need to bother with the extra filesystem expenses because they do come in expense. But if you do need to do that, we want stateless apps or in some cases we don't want to serve the files ourselves. So we want something like a CDN maybe for geographical reasons or other, uh, or we maybe want to share the files across multiple applications. And I think when we then, going through the process of choosing a filesystem, we're not asking direct questions. Mostly, if we're asking the questions were asked like, do they need to be on premise or can they be in the cloud aka still somebody's computer. But I think more important things to take into account is, for example, the question of who will do the maintenance?

If you've got a file system locally, you also need somebody who manages that. If you're using a service that's can basically scale infinitely like S3 or Azure blob storage, you don't have to think about it anymore, but that might also come at a cost at a later time when you just keep growing in your expenses go up. But that's a different problem. Hiring An, uh, an operations person to manage a cluster of filesystems, uh, it's not an easy task. So you need to think about that as an organization and as a developer, do you want to spend your time doing that? Or do you want to spend your time writing code and just consuming. But also you need to think about how we will use the files. Do we have particular needs that the file needs to have? You can serve, um, for example, videos from your local web server, but there are also specialized services where you can upload your files and they are like a media service that will do all the bit rates stuff for you.

So you've got a specialized more capable file storage solution that you can just use. You can also ask why are we storing things in the first place? Uh, there are a lot of cases still where people upload CSV or excel files and they extract data from it and then they use that. They store it in the database, but they will also keep the file separately. And do you have rules about how long we keep that? How long are we going to keep those files? How big will the files be? All these sort of things, questions need to trigger you into finding out what the needs of your use cases. So it's really good to distinguish needs and capabilities. Needs is what you require from a solution and a capability is something that something can do. So for example, S3 can be used as a CDN, but if you don't have those needs, maybe you don't need to use S3.

Finding the right Filesystem

But now if we've asked all these questions, we need to somehow document this. So I think we need to write stuff down. But if you're then looking at the agile manifesto, we often see this, well, working software over comprehensive documentation. And what developer then thinks is, haha I don't need to write documentation. I would argue the difference. Um, and I had a conversation with Rafael Dohms about this and Rafael Dohms, is basically he's a Brazilian. That means he talks in Portuguese but then talks about barbecues a lot. Um, I think that's correct. Correct me if I'm wrong, and he reminded me of a technique called ADR. ADR is Architecture Decision Records. And in this format you can store all this collective knowledge. And this consists of a couple parts. One is context. The context is something that influences the decision, um, that is something that is there, that's a given.

For example, the team uses AWS. If the team uses AWS, it's more likely they will choose for S3 and if the team uses Azure, it's more likely that they will use something like Azure blob storage. And this is a logical decision, but it's good to keep that in mind. Um, for accountability purposes. Um, another context might be we expect massive storage growth. This might be something to think about, like, do I want a infinitely scalable file system? If you are expecting a large number of files, it might be a smart thing to think about that from the get go. Another requirement would be that we need to store things on premise or data must be stored in the EU. Uh, do, do you know this for the files? Do you ask these questions? Like you, we should be asking these questions and then we'll also document the decision.

So what is the decision that we made according to the context and the needs that we had? Well we'll use AWS S3 or Blob Storage or whatever, but then there's another part and the other part is the consequences. So if we're moving from local to something remote, we have consequences in our code. So we need to change our code accordingly. This is a consequence of the choice that we made and by having such a document, you also have something that you can communicate with your team leads, but also to management in order to say, well, we see these conditions and we made this recommendation. Now we as a company can make this decision. So you make people accountable with you for the decisions that you've made. So if later in time your needs have changed and now you're not the developer who made the bad decision, no, you and your manager both made an agreement that this was the right choice at the time. So you actually have, something to motivate your manager in order to make the new right decision with you without it coming at your expense. So I think that's a really, it's more political than anything, but it's really a good thing to think about.

Use-Case Driven Storage Models

So now we've chosen something and now we can actually create an implementation. So I would argue that the best thing that you can do for any application is to create Use-Case Driven Storage Models. And what do they look like? Well, they can look like an interface, so the interface could be ProfilePictureStorage and we'll just have a store function that will accept the user and their contents or in this case a resource containing the content. So profile picture or for example, a financial report. Same thing and how does this look internally? Well internally, it looks something like this. We've got an Filesystem Financial Report Storage which implements the Financial Report Storage interface and it uses the file system internally to dispatch. And this is a really powerful thing because this makes sure that the path created, which is a relative path, so easy to move around, will always be created here. We don't have multiple areas where we're operating on the same part of the filesystem. So we've got an encapsulation of our Use-Case in this thing. Just to reiterate, it's really good to isolate your use case. Um, so we now have consumed this and now what happens when stuff goes bad?

What Happens when Stuff goes Bad

So who here knows of the concept of, or the thing called (Blameless) Post-Mortem? Okay. Who has a meeting after something like a production incident went down? Okay. That's way too few hands. So like if we look at this, if we know that we need the information from the end in order to make the decision, uh, back, uh, or at the front. We need some sort of feedback mechanism. So what I propose is to document that as well. At least there's a sort of reference for the, for the future. So the next time you'll see, well we have these problems and instead of relying on gut feeling, you actually have a record of what happened. And what this needs to contain is what actually happened. For example, the storage was unavailable or what caused it. For example, a full disk, what did we do to fix it? Well, sometimes this can be the dirty thing you needed to do in order to go like ssh on the production machine and manually deleting files that you no longer need. Or even deleting files from a different part of the application in order to free up more time for more space for the, for, for the problematic area.

This is something that can happen. And, but more importantly, you can also document how you could have prevented it. If you're looking at your way of working, some of you might be working agile. If you're bothered by this a lot and you can give priority to these kinds of solutions, you automatically have a very good justification to tackle technical debt in your, workflow as part of your workflow and with probably with consent of management because they saw what kind of pain they and the business endured and what kind of solution you can provide.

Summarizing!

So to round things up, use Flysystem. Keep things simple. Use a filesystem for what it's for and use the other capabilities if you want but always keep that file storage concern, isolated. Document your requirements, document your decisions using ADR. Create specific storage classes per use case and create use generic solutions for generic problems and use special solutions for special problems. And just to finish up and see how weird things can be, um, for example, you can use ceph and ceph also has a way to talk over HTTP and it actually implements the S3 protocol so you can use your AWS adapter to use to talk to ceph and ceph can be installed locally so you can be your own S3 if you need to have stuff on premise.

This might be an interesting solution for you if you're heavily invested in S3, but need to move stuff back on prem. You can also use mongoDB as a file storage, but for very, very specific use case. For example, you can have a, geo distributed cluster of mongoDB nodes and then you can use the file storage component of that. So the files will be near where they are used. So for sort of file sharing, file transferring, geo where use cases, you could use something like that. For more cluster stuff, you can use Gluster FS and you can use or have a Gluster FS cluster running on Kubernetes and if you have Gluster FS running there, you can also create a persistent volume on top of that Gluster FS cluster and Kubernetes. And then you can have a service on Kubernetes that consumes a persistent volume on Gluster FS on Kubernetes. So that becomes really complex and as you can see, a, it's now a arrow up, which coincidentally also is the direction of where the complexity is heading or maybe not do that.

So last question, what do I do? Well, it still depends, but I think with all these questions and all these things that you now can think about, I think we are able to make the wrong decision the right way. I hope I have convinced you and if not, I hope you enjoyed this talk. Thank you for listening.

Chapter 5: Symfony Messenger: 6 months already and more to come

Tip

SymfonyCon 2018 Presentation by Samuel Roze The Messenger component brings asynchronous processing to Symfony: using message bus(es) you are able to decouple your application and route some (or all) of these "messages" to transports such as the built-in AMQP transport. It's been more than 6 months since we've merged the new Messenger component in Symfony. We will explore the different use cases it has been used for so far and how it will continue to involve in order to facilitate even more.

Hello, good morning. So, roughly six months ago we released the Symfony 4.1. And that was the first time that we actually introduced the Symfony Messenger Component. So today what I want to talk about is obviously this component. I want to quickly introduce it and that's going to be the first part of the talk. And then sort of like explain what's, what has changed in the, during the transition from the 4.1 and the version 4.2, that was actually released a few days ago I think.

Hey Samuel!

Before that, I'll introduce myself very quickly. I'm Samuel Rozé, I'm part of the Symfony core team, the ApiPlatform as well and I've created a bunch of open source things like ContinuousPipe and Tolerance. I don't have any time to present them but you should check these out, it's quite interesting.

Very quickly, the startup I'm working with, we are on a mission in trying to help the old people, the elderly to stay healthier and longer at home instead of going to retirement houses. It's, to me a very interesting project. You should check the website, it's birdie.care. Obviously as most of the technology companies we are actually hiring. So if you're interested, talk to me.

Symfony Messenger? The 5 Concepts

So Symfony Messenger. So what it is? There are five main concepts within the actual, within Symfony Messenger. The first one is messages. You could have guessed it. So it's your PHP object. It's a PHP class that you will create that you will dispatch somewhere and you will dispatch this message to a message bus. Then once the message has been dispatched to the bus, the bus's responsibility is to actually call some things that are going to contain the business logic, your business logic, and these things are called message handlers. It's a bunch of classes, no specific requirement, it's, um, what you want to execute when you receive, when the code receive the message basically. You can have zero to N handlers for a given message.

Just these three points are really super valuable. And if you're into, um, sort of design patterns or sort of like interesting things like CQRS if you've heard about it, the idea is that we can actually structure an entire application around the bus. And we can actually structure an entire application around *a* bus or multiple buses. But the point is that we can actually do really, really, um, decouple who is dispatching a message from who is actually handling the message. I won't go that much into the details. The important takeaway here is that just these 3 concepts, without talking about asynchronous, without talking about queue all of that, is already extremely valuable and you should actually try to play with them.

Turns out, that also we didn't have a solution for asynchronous things in Symfony. So one of the other concept in the Symfony Messenger component is the notion of transport. So we can actually say, well, instead of actually directly calling the message handler or your message handler, well, the bus can actually switch to something else. We can actually route the message to actual, to a transport. And the transport is really something a bit abstract, but something is responsible of talking to your, to a third party. The most famous one is RabbitMQ being a third party, so queuing mechanism. But technically it can be actually anything, can be Gearman if you want to do some distributed processing for your messages, can be an API, there is no real limit. And the last missing dot here is that if you actually dispatch the message and you route it to a transport, it's in the queue, but nothing will happen. So the fifth sort of concept is this notion of worker. So it's just something that is going to say, or ask the transport: "hey, tell me when I receive a message" And when a message arrives, it will actually call your message handler.

Messenger in Diagram

The same thing as a sort of a graphical representation. You've got a message, you dispatch a message to a message bus,

that is handled by one or zero or N handlers.

If you really want to, you can route this message to a subset of transports: to one or multiple transports, instead of directly calling the handler. These transports are responsible for talking to these guys, and we've got the worker that is responsible for receiving the message from the transport and dispatching them back to the message bus. Now the important thing here is that actually, if you actually route the message to a transport within your HTTP server - or your like main PHP thread - um, you actually dispatch the message and your handler won't be called. It will be super fast, you will hand it to the queue. And you need to run this worker somewhere else. So you won't be on your web server, might be somewhere else on another machine, but the code that your handler contains, or your message handler contains, is going to be executed outside of the PHP context or the web context. That's the main, that's the main concept.

Using Messenger!

So now I'm going to show you exactly how from a user perspective or developer perspective, how can we use it.

You all have seen, I guess what a Symfony controller looks like. Now it's fairly recent, but it's a few versions already, we can inject services directly in the action arguments. So in this example here, I'm going to inject a MessageBusInterface that is already configured and wired for you if you have the component installed on your Symfony framework application. What you can do, as I mentioned, you can actually dispatch using, calling dispatch method, on the bus, you can dispatch your message. There is no requirement. I mentioned that already, but there is no requirement: that's how your message looks like. It's your own php class. So whatever you want to do. If you want to actually send a notification, you can call it a SendNotificationMessage. If you want to, I don't know, buy pizza, it can be BuyPizzaMessage. You can put the properties you want, you can put the method, the getters, the setters you want it is completely your own code. There's nothing... to compare with the event-dispatcher for example, where the event needs to extend the Event class, well, we don't need that anymore. At least in this component.

Once you've dispatched your message, guess what? You need a message handler. Same idea: there is no specific requirement on the message handler. The only thing is it needs to be a PHP callable. So typically it can be an anonymous function. In our beautiful world of classes and all of that, it is a class that has a method, a magic method named __invoke(). If you type-hint your message, type-hint your message, as the first argument of the __invoke method, then Symfony will automatically understand that it is this message that you want to handle. And when you dispatch the message this handler is going to be called.

There is a slight tweak here. It's just that in order for Symfony to automatically configure your handler, and automatically add the tag required to actually discover your handlers, you can use this marker interface, and this marker interface is called MessageHandlerInterface and contains nothing. You don't have any requirements, but just to say to Symfony: "Hey, can you wire that handler into the message bus?" Okay, cool.

Now, if you run that code, your handler, whatever your domain logic is will be run synchronously. So that's the simplest way of registering a message handler. You've got another way which is very similar to the... what we've done with event dispatcher component. We've got a MessageSubscriberInterface. What you can say is that, you can say, well, this handler or this class will actually handle this message and this message and this message. You can handle multiple messages and you can also give it a very interesting configuration. So you can say that, so the syntax it's a bit different, it's using yield instead of returning an array. There is a technical reason to that. But the main thing here is the return value or the value of your yield can be a set of options. So you can actually say instead of the default __invoke method, you can say, well, let's call the doSomething() method. You can also configure the bus. So if you've got multiple buses, for most of, you are using an event bus or command bus or this kind of stuff, you can actually say, well, this subscriber or this handler will actually just listen for this message on this specific bus. You've got a notion of priority as well, etc.

An interesting point of using the yield instead of returning just an array is that we can yield multiple time the exact same key. So we can have multiple configuration for the same message, so we can actually, we can have the handler be called twice if we use the same message key but different priorities as the value. It's very interesting. Anyway, I'm not sure you're going to use that all the time, but that's another way to register the message subscribers.

Now, very important, by default, every single message is handled synchronously. And I think it's a pattern that starts to be, we can start to see if you've watched or listened to the Symfony Mailer, announcement from Fabien a few conferences ago, actually the idea would be that the more and more we ship the more, maybe, you would just be able to dispatch a message into the bus. And that would be a sort of a nice way to provide extension points from library perspective: here's all the messages you can dispatch and that is going to do the job.

Async & Transports

Now if you actually have a message handler that is a bit too slow - and that is the main use case. If it is too slow, what you can say is that either you wait. Or you can, well you can configure this message bus to say, I'm going to route this message

somewhere else. And to this message you're going to route it to a transport. So how are you going to do it?

Well, first thing you're going to create a transport. So transport is something that can be created from a DSN or URL here. If you.... Actually, who here is actually is using Doctrine? I was expecting more people. So half of the room actually admit they use Doctrine. So the idea is simple: you've got a database URL in Doctrine, so you can configure whether it's MySQL or Postgres database. Here is the same example. The transport: you give it a name, you give it the name rabbit, it's completely up to you give it the name you want, and the main configuration is the DSN. When you create the transport, you run the same code it's going to be as slow as before. Because creating a transport doesn't do anything. What *does* something is when you route the message... explicitly routing a message to a specific transport. So here you say your App\Message\YourMessage is going to be routed to the rabbit transport. Here I just put the actual class, but it can be a parent class or an interface, that works as well, so we can route a specific set of messages. And if you rerun the application, wow! Now super fast.

Workers: Handling the Async Messages

The only problem is... Nothing happened, obviously. Because this message has been sent to a queue, in this example RabbitMQ queue, but we didn't consume it, so we didn't run the actual handlers. And that's the fifth point, which is just the worker. The worker look like a... just a Symfony command. So you run bin/console messenger:consume-messages into another machine, on another machine, and this will listen for messages and consume them. That's a long-running process. It's got a bunch of options if you want it to be... to auto kill after like X time or X hours, etc. But that's the main thing you need to do. And here your handlers are going to be called for every single message. And that's it.

That's the one on one of the Symfony Messenger. That's how it works and that's mostly all we want to know about it.

What Happened During the Last 6 Months?

Now, I mentioned it's six months already, so I'm going to just do very quick rewind of like what happened and what we changed. So, six months ago Fabien tweeted this things like, yay! We merged it! So probably a year ago I started to create a Symfony Message pull request. We actually broke Github because it was too many comments. And finally somehow it got merged. Super cool. We actually renamed it to Messenger. And the ignorant me was saying: "yay, job's done!". Turns out that actually that was just the beginning of like a lot of work.

So since then, so what happened? Very quickly, um, we only had 33 issues. Super cool eh? It worked super well. But we actually had 150 merged pull request in the component. And if you mention the, sort of, not necessarily the pain but the complexity of getting pull requests and the review process and the Symfony code base. That's a lot of work. Out of these pull requests and other kind of side commits, more than a hundred of them were actually for the preparation of 4.1 release. And 91 commits for the release 4.2. Probably out of this 90, 91, is probably 50 of Nicolas because he destroyed the component recently.

But the good thing is it's actually coming from 21 different contributors and thank you very much to them. That is the beauty of the... Thank you. That's really the beauty of Symfony I guess is that everybody's just here to make it better. Super cool.

Improved in 4.2

So as of a few days ago, it's actually already 100,000 plus downloads. So it's actually been getting used. And now we are actually starting to have much more small tweaks and small features on this component. And I'm going to present three different things that have changed or have been improved in the version 4.2, the new one.

Envelopes

The first one is the notion of envelopes. So I didn't talk about it yet and this is the best moment to introduce it. The notion of the envelope is to say, well, you've got your message, we didn't want to put any pressure on your side - do anything special with the messages... although, when we started to go asynchronous, there is a lot of things actually we... that can go wrong and there is a lot of information we actually want to carry around.

Well it's the same as the beautiful analogy of the postal message. You wrap that into an envelope and you put a bunch of stamps. That's the exact same thing. We have envelopes within Symfony. It's a PHP object, it contains a message and guess what? It has some stamps. So before there were actually named Items, that's the main thing that we actually renamed to make sure that the analogy makes sense for everybody, but it's really that. So we can mark the envelope with a bunch of information. Built-in within the component, we've got three stamps. We've got the ReceivedStamp, we've got the SentStamp and we've got the HandledStamp. So we can actually know when we get an envelope, we can know whether the message has been received from a transport or actually we just sent it to a transport, or we actually, the handler has been called. That's the main, the really important information.

So how we are going to use it? I'm going to show you the... so that's the message, the interface, the main interface you're

using. You're going to use every day - the MessageBusInterface. That actually changed in the 4.2. So the main thing is you can dispatch a message. But actually, if you look at the param, param PHP doc, you can dispatch a message or an Envelope. So you can yourself, on your side, create... wrap your own message into an Envelope and add a bunch of stamps, if you want to. But main thing is it actually returns an Envelope now. Before, it was actually returning the return value of the actual message handler. Um, and that guy, Nicolas did... So basically he said, well: Symfony Messenger sounds very interesting, I'm gonna actually look at it. And he found a bunch of things to fix. And one of them was actually a long, long, long, long, long discussion over multiple weeks: how can we manage this return value of the message bus. Beginning was a mixed return value. That sort of made no sense because it's super unpredictable, when you dispatch a message, you don't know what's going to come back. And that is a real problem. So then, we decided to say, okay, let's return the Envelope. And that kind of makes sense.

Now by doing that, what you can do is when you dispatch the message, you get the Envelope back. And on the Envelope, you can get a, you've got a bunch of methods. And the one that is used here is the last() method. So you can get the last stamp of a specific type. So first, obviously it implies that we can have multiple stamps of the same type.

But the first example is *if* the last stamp of type SentStamp is not null, well it actually means that we sent the message to the transport. Just right now when we called $bus->dispatch(), it has been sent to a transport. And each stamp contains a bunch of information. The second example is that we actually use the same same thing: we get the last stamp of type HandledStamp. But you actually get the stamp into a $handled variable. And what we can say is that, well, the message has been handled. So a message handler has just been called. And we can, on the stamp, call the getResult() method to get much more information. So we can get the result of the handler. Actually, because multiple handlers can be called, you can have multiple HandledStamp and then you can get all the results. Same idea for if you send it to multiple transport at the same time, you can have multiple SentStamp. So that's how you would use the Envelope within your code base.

That's the advanced track. So the stamps themselves, they are automatically added, they are added by the Symfony Messenger - the few of them that I've mentioned. Now, you can very easily create your own. So you can just create a class, implement the empty StampInterface and create your own thing. So you can carry much more information if you want to have a unique identifier for that message, then you can carry it across your application. That you can do. If you want to carry the user context, you can completely do that outside of the message. You can carry much more information.

And the very important thing is the stamps will survive the serialization. So that's, so if when you start to do asynchronous processing, that's where the real troubles comes in, is the serialization process. So your message, your php class, when it's within the php memory, super cool, we can put whatever we want. Now, if we want it to be consumed by another php binary somewhere else, we need to serialize it. We need to put it in a way so that we can put it somewhere as a string and deserialize it into the other php process. So these stamps are going via the serialization process.

So very quickly, that's the detail of the AMQP transport, the default serialization, or the way your message is actually sent, is using the Symfony Serializer using the JSON format.

Tip

In Symfony 4.3, the default serialization has been changed to use PHP's native serialize() and unserialize() function. It *is* still possible to serialize to JSON, but the new default serialization is much easier to use for most use-cases.

So that is how a message would look like within the RabbitMQ. This payload is literally just your message. So within the payload of... which is in the message of RabbitMQ is just your thing with no constraints. No Symfony, neither php related things. So what it means that you can very easily use the messenger to also communicate with the other types of systems, other language, other frameworks. Now, to carry a bit more information, we use the headers. So we basically put the type of the header to add the type header, to say, well that is the type of message, so we can deserialize the right class. And also we can... another set of headers for each stamp. That's how we carry this information.

Middleware

Very, very related: the second concept is the concept of middleware. And actually there's a moment where I said that the message bus itself, the class MessageBus, which is the only implementation of the MessageBusInterface, is I think it's 10 lines of useful code. The rest is just comments and stuff like that. The *real* logic of every single bus actually comes from the middleware. I cannot say middlewares because in english we don't say that, it's just middleware, there is no plural apparently. So that's the middleware, multiple of them. You may have seen this kind of stuff. Um, that's the graph that comes from CakePHP that describes how they use middleware, the middleware mechanism to handle the request and the response, the HTTP request and response. The idea is very simple: every single middleware is in a stack. So there is an ordered list of middleware and each middleware is responsible for calling the next one. And they can do specific actions before or after. The Symfony Messenger bus as the exact same way of working.

So there is our middleware, the Symfony Messenger one, and there are two main important middleware. The two or... the two in the middle. The first one, which is basically because... the one in the middle, the last of the stack, so the last to be called is the HandleMessageMiddleware. So that's the middleware that's responsible of saying: okay, I got this message, I am going to find out which handlers are registered for this message and call them.

There is a middleware above that, which is SendMessageMiddleware. And this middleware is responsible for sending the message to a transport. So if this middleware decide that, well I need to send the message to a transport, it won't call the handle message middleware: it will just return the Envelope. So the entire behavior of a message bus is actually within the middleware.

And you can create your own. So in order to create your own, that's just a class that implements a MiddlewareInterface. It has just one method. I kind of mentioned it: it takes as an input an Envelope, and as an output another Envelope.

It also takes the stack as - the middleware stack as an argument so that it can call the next one. So an empty middleware that's is completely transparent, will look like that. The main thing is return stack: from the stack and get the next middleware, and call the handle method again. I just call the next middleware. And here, I can actually add some behavior, I can add some things, things that are going to happen before and some things that are going to have to happen after. After what? Or before what? After and before the next middleware. And if you look at the previous slide, it actually means it is going to be running something before or after either the message has been sent or the message has been handled.... your message handler has been called. Yep?

So, I'm gonna show you an example middleware, and I'll walk you through. The main thing here is that line at the bottom, which return is, return $stack->next()->handle(). That's super important: as a middleware, I need to call the next ones. Here, the concept is to say: well, I can create a middleware that's responsible of auditing what are the commands or what are the messages that went through. So the first thing that I'm going to do is to get the message from the envelope. So that's your php class. Get the message so that after, I can display a bunch of messages that contains the class of the message. So it's going to say: well, either I received the message or I started to dispatch a message. To get the class at the end of the line, just get the class of the message. Now there is one thing that if - and you've seen that already with the two other stamps, but it's the same mechanism here - if from the envelope we've got the stamp that says ReceivedStamp, it actually means that we are in the consumer. We are under, bin/console messenger:consume-messages side. So in the logs, I won't put a "started with message", I would just put, like, "received message" because that's the context.

So when you look at your log a few days later or something, the future you will actually thank you very much - that you didn't put the same message and that you don't know when things have happened. We don't know exactly, um, what was the context of the message execution. Because the middleware they were called when we received the message from the transports or, when we actually dispatched it from the web application. Because the messages all the time go from the bus.

Now, there is this notion of stamps and this custom stamp. So here is a simple example where you can create your custom stamp - here I have named it AuditStamp. It will just contain an identifier. So we can actually stick an identifier to this given specific envelope, and you can stick this identifier so that in your log you can correlate the log that have happened on the web server: when you created the message, when you dispatch and send the message and the logs that are happening in the CLI, in the background process, because you're going to have the same identifier for this specific message. Because this stamp, if it doesn't exist - that's the first line - if the AuditStamp does not exist, you add the stamp on the envelope. And because these stamps are like, carried around with the serialization process, you're going to have your AuditStamp when you consume the message. And that's this middleware. So this notion of stamps and envelopes - coupled with the middleware - in my perspective are super, super valuable because we can do such things.

When you create your own middleware, well the only thing you need to do is to register it. So you need to explicitly say this bus is going to have this middleware. This example is saying you can create multiple buses, this is just an example of creating multiple buses. And here one of them that we called command_bus will have this AuditMiddleware that we've created. So every message that we dispatch into this command_bus, will go through our new AuditMiddleware. while the rest, or if the message is going to the event_bus, won't go through this middleware. That's an example a bit more complex because we've got multiple buses, but imagine and replace, just remove the event_bus and that's the normal behavior of just having one bus in your Symfony application.

Now there's just a small clarification: the stack. Because that's the main thing as well that's changed in the MiddlewareInterface. Before we had a sort of magic second parameter. Instead of the stack, there was "next", so we could just call $next() parenthesis and call the next middleware. Well the problem was, by doing that, the next, the function that was behind the next variable was actually within the message bus class. So what it meant is that the stack trace was super, super horrible. And if you're using a profiler or something like that, you will always see the middleware going back to the message bus, going back to the next middleware, etc, etc. That way it's actually much more explicit as well: you get the stack and you get directly the next middleware within your own middleware. So you just go: $nextMiddleware->handle() and that's it. It directly called the next middleware. The stack trace is very good or very simple when you get exceptions. And that's very useful.

So that's the sort of, um, details of what's changed and how, about the middleware.

Transports

Now, the last thing I want to talk about quickly are the transports. The first thing, again, and I think it's really important to reiterate that, is that you don't need transports to benefit from the Messenger component. By itself, the structure that it offers you is really interesting for you to use. So we don't need transports. And I think it's been built in a way so that we can have very nice or an easy migration path. As in, we do synchronous as much as possible. The reason is, asynchronous, for a lot of reasons, including the fact that you might have inconsistencies between when you run the message and when the message has been dispatched, with the fact that you may have something that actually needs to return the value from the synchronous background process. We need to have sort of like asynchronous notifications via email or whatever. There is much more complexity.

So you can start synchronous. And *then* you can configure the transport and then you can write a message to be asynchronous. That is sort of like the great path, as has been planned. So the transport, we have a built-in AMQP transport. So if you've got, the um, the only requirement is the AMQP php extension. If you've got that, you can actually directly use RabbitMQ, and that's it.

Now there is an amazing php library is called Enqueue , which has 1+ transports. So if you're using AWS, SQS transport, if you are using google, you've got Google Pub/Sub, so using more exotic things like Kafka, you can actually use them. Even just a file as well, this kind of stuff. There is an integration with Enqueue there is an adapter so that you can directly create using the EnqueueBundle all these transports and use them directly in the messenger. So you can directly route the messages to an Enqueue transport. So it means that already right now you've got dozens of actually transport you can use.

One thing that we changed, which is very important for you if you want to create transports, is that we simplified the serialization. Before we had the notion of encoder and decoder. Now it's very simple. You've got just one class, that is responsible of serializing and unserializing the messages. You can, so I mentioned by default, using Symfony Serializer, if you want to use protobuf, if you want to use csv, you can easily create a class that implements the SerializerInterface and roll your own.

What Now? What's Next?

So what about now? So now the Symfony Messenger component, it's still experimental in 4.2. So what it means is that maybe in 4.3 we're going to change things as well. So the idea is to say for new components, because we - and especially this one which touches more than the scope of just the php process - there was potentially things we didn't see coming. And maybe there are design or structure in the component that we need to tweak to make sure you actually benefit for everybody. I think it's very unlikely, but it might change from 4.2 to 4.3. It doesn't mean that it is not usable, it just means that if you, when you will upgrade from 4.2 to 4.3, you might have some slight things to change.

It is much better than before. And really I think the massive work of the community has been great in a way that: it's super easier. The error messages are like, make much more sense, and you've got this notion of stamps that actually to me seems very, very simple to grasp.

Now, the documentation deserves a lot more love. It's a super hard problem, especially for components moving super fast like this one. If you feel you can help, that's an amazing contribution to do, and it will help a lot of people. So on that. Thank you very much.

Chapter 6: Going crazy with Varnish: Caching pages of logged in users

Tip

SymfonyCon 2018 Presentation by David Buchmann

You know how HTTP caching works but need more? In this talk we look into ways to cache personalized content. We will look at Edge Side Includes (ESI) to tailor caching rules of fragments, and at the user context concept to differentiate caches not by individual user but by permission groups. A big help to leverage this concept is the FOSHttpCache in combination with either Varnish or the Symfony HttpCache reverse proxy.

Aright! Welcome to my talk on Varnish. Today I want to talk about how to use Varnish, not just for caching static information kind of things that are the same for everybody, but caching when the content that you show depends somehow on who is viewing that content.

HTTP Caching For Scaling

If you've heard some caching in general, I really like this quote to explain why do we do the kind of HTTP caching, like the caching where we cache the whole page. In the end, like all that we do, if you do a visual website or if you do an API, the output is some kind of text content and if somebody else, it's exactly the same text content. The HTTP cache is the most efficient because you'll see, okay: he's asking for, he's asking the same question, he gets the same answer and that's really, really easy. You don't even have to touch your actual web server

Sometimes people start looking at caching because they realized, "Oh our page is slow, so if we have a cache, it's going to be fast." It's not exactly right or it's a bit... like the main reason why you want to do caching is to allow scaling because you will always have cache misses: like moments when the cache doesn't know the response and you have to hit the application and if the application takes five seconds to render a page, some of your users will have a shitty experience. Cache can't fix that problem. It will have... it will lower the number of people that have a bad experience but it won't make the bad experience go away. So the actual like the good motivation for caching is to say we: we want to be able to scale. We want to scale with less servers because if there is many people asking for the same thing, it doesn't take us a lot of resources. But the application behind should still be in a reasonable shape to deliver pages.

What is a Reverse Proxy?

Now varnish or any kind of caching proxy on that level, they don't know about the application. They know about HTTP, about the transfer of the data and they see it as a proxy between the Internet and your actual web application. So you have the requests coming through the proxy. It checks, "oh do I have that in my cache?" If it doesn't, it goes to the backend, gets the response. And if the response is cacheable, if it, if it is, um, there is ways for the server to tell: "Hey you can cache this response" or "Hey you shouldn't cache that". If it's cacheable it will store it at that point and then return the response. And then the next time somebody asks for the same thing, it will see: "oh I have it in the cache" and immediately return. And with varnish that's like a single digit millisecond time that it needs to, to handle a cache hit, that's like really, really fast and it doesn't take much resources.

Real World Caching: User-Specific Data

So the easy case of caching is when you have some kind of page that is the same for everybody that's really easy. But in real world for example, let's look at this newspaper thing which has here, there is the weather, it is localized, it's talking about Zurich. Maybe if you come from Berlin it will show you the Berlin weather. There is something related to your session, like here you can log in and I assume if you logged in it would show your name, stuff like that. So this depends on, on if you're logged in, this depends on where you access the page from and then there is some kind of things that are updated repeatedly. Like all the time they edit and update this and then there is some kind of headliner a thing that doesn't change that much. And there is a news ticker which maybe is even fed from the user profile because it knows that you like sports, so the news ticker is full of sports or you like international politics, so that's what they put in there. And such a page will have a high load so caching would be very beneficial, but you can't cache this with the naive approach because then everybody would see exactly this page of whoever happened to access it, which doesn't work.

Another example, I guess most of you have seen GitHub and use the GitHub interface. Again, like most of this stuff is the

same for everybody. You have a issue, you have the state of the issue information. Again, you have this login information. Then you have some something that is like, this is specific to you, but it's going to be the same on every page, but then this is specific to you and it depends on which repository page you're on. You can watch this repository or unwatch it. And then the green, like all these buttons to edit is about your permissions on the systems. It's not really personal. Like, if two people are allowed to edit, they will see the same thing there, but if you're not allowed to edit, you will see something different.

Dealing with User-Specific Data

So after that introduction, let's look at some options, some alternatives we have when when we have this kind of content and we want to cache. So first one is kind of the, if you can do that and do it because it's much easier: avoid sessions or if you have a kind of CMS thing with a form somewhere, maybe just not cache that one page with the form and you have the session for handling that form. But once that page is done, remove the session and the session, remove the session cookies again so that the rest of the page remains cacheable.

I call that cheating with JavaScript when you have, you said some information and then you deliver static pages and have JavaScript for changing little bits of the page. Then the kind of tricky approach of caching, even though there is cookies or caching per user - we're gonna come to that later - and finally the user context idea of where you cache, not by individual user, but somehow define groups of users that have the same shared cache.

Avoiding Sessions

So avoiding sessions. The like the the thing maybe to say that explicitly, as soon as there is a cookie, a normal cache will say: "I will stop caching" because the cookie is individual. It's probably meaning that everybody gets something different so I shouldn't cache. And with this approach you would leave that behavior. So if there is a cookie, there is no caching, but if, if you don't need the session, you would go and delete the cookie immediately so that there is no session. One thing on, on Symfony, if you use the flash messages, that triggers a session just if you do app.flashes, that thing in it initializes the session. So if there is no session, the response will set the session cookie and you stop caching. So you could check like does the request... did have a session? So should we do that? And if there is no session there will be no flash messages because the flash messages are stored in the session. And like in general if you do this you will have to check your application if it happens to somehow start sessions and then go figure out how to not make it do that. If you don't need the session.

Another thing you will need to do, so this is the Varnish configuration language, you would want to clean up the cookie so that it only has the php session id in it and not any other stuff. Like typically if you have Google Analytics on your page, there will be some random cookie things that actually only the client side cares about, but that can be sent to the server and the... Varnish or other caches don't try to guess what exactly the cookies mean. So it will see, "oh there is some cookie" and it changes all the time so I'm not going to cache that. So what you do is have this regular expression to extract only the session id out of the cookie and keep that. And then if there is no session id, you would unset the cookie.

Moving Logic to the Frontend

So far so good. Cool. Then if we move the logic to the frontend, so the cheating is not nice to say. It's not really cheating, it's just a solution where you can still cache even though there is some things different. So basically when you log in you would set some kind of cookie that the JavaScript and knows oh this user is logged in now. And with the thing that I showed before you remove the cookie in the requests, you would even remove the session cookie for this approach. So the user is logged in, but then on most routes the session cookie is removed and everybody gets the same page. But then, uh, in the frontend, in JavaScript, when the page is loaded, you would change the page. So the rendering would include all the editing buttons. For example, if you, if we take that GitHub example, we would include all the editing buttons and then have JavaScript show them or not show them depending on what's going on. Or for example, if you have this, say you're logged in - "Hello David" - you could store the information that I'm called David in a cookie and then whenever the page is loaded, have JavaScript replace the login button with "Hello David" and the link to the profile. And then if I click the link to the profile, the server would not ignore the cookie and have an actual session and not cache that because that's my profile editing - there is no point in caching that part.

I think that's really useful when you have these like small bits of a page that change or you have the login on a otherwise pretty static page, that's a useful approach. You get fine-grain control over what to show. You can cache a lot of things. You don't have network overhead with this, uh, and the backend, doesn't have any additional load from it and you can handle complex user interfaces. On the downside, the templating gets more complicated and you would, you would also in like put more things into the page then needed on any specific for any specific user. It's like everything for everybody. It can be a bit tricky to maintain. And if you need additional data, for example, if you, if the editing would show you some kind of list that non logged in, users are not allowed to know of - like a list of options and you don't want to leak that information - you will need to add additional calls to fetch that data and so on. And then suddenly it becomes complicated.

Again, kind of similar, you could use AJAX. So say we have these parts of the page that are the same and cached, and then we just have the AJAX include to fetch something with the session that is depending on the session.

Yep, so that would be executed after the page is displayed. So if it's a little information, that's not the thing the user wants to see first. It's okay. Um, but if it's like the main thing, you have this wobbly experience of the page loads and then information starts plopping in. And you'll potentially end up with lots of requests to the backend and again, you complicate things. So overall it's similar to the previous approach where you, kinda make your application a bit more messy to optimize it for the caching.

Caching Despite Cookies

Now you can force caching even if there is a cookie with Varnish, um, but it's extremely easy to trade problems. Like if you cache something and the, and that something really depended on the user, then the next user would see the same thing even though he's a different person. So you want to really avoid that. The backend would in this case needs to control what should be cached. So if it's individual, it really has to, to set this no-cache header. You could cache things depending on the session by saying it's cacheable, but it varies on the cookies or the Vary header says: if this header is the same in another request, then the cache is okay. But if the header is different than even the same URL is not the same cache. Like it's... the cache is then calculated... the identity of the cache is the URL plus that Vary information.

So that could work. If you have this AJAX call to show my login status, for example, that would be the same. When I do the call with my cookie, it's going to be the same all the time and I will do that call a lot because I do it on every page. So that would be a place where you can use that. If you put Vary on Cookie for everything, you better remove the cache because there will never be any cache hits basically. Because if you... like, every user will have a different cookie because the different session and then everybody will see, like every request will be different and you have no almost no cache hits. And if it's about the same user accessing the same page again, then you better make the browser cache that then your varnish. So Vary cookie is useful in special cases. And then if it's something that doesn't depend on the session at all, you would just say: cache it - it can be cached, even though there was a cookie.

If you want to do that in Symfony, you will have to explicitly tell Symfony: I know what I'm doing, since version 3.4. Before it was, if you had a session and then you set the header "this can be cached", it would transmit that. And then it would be Varnish that said like: "oh, that looks fishy, I don't cache it." But now Symfony itself will, as soon as there *is* a session, it will set the cache headers to make the, the response not be cacheable. And to disable that behavior you have to, to set this kind of magic header to tell Symfony don't override my headers. I actually, if I say cache then I, I wanted to say that. I know what I'm doing.

And then on the Varnish side, so this is the default behavior. Varnish will see is the request even a GET or a HEAD request and if not pass means don't try to cache. And if the request is not a GET or a HEAD request then there is no point in caching. And then by default the next step is it looks at the Authorization header which is used for the, like these pop up basic password thing. Or if there is cookies. And if there is any, then it also decides pass. And then the last line where it says hash is saying in all other cases I want to try do cache look up. Now if you want to cache or have the possibility to cache, even if you have cookies, you would change this method. You will override that in your own vcl. You keep the thing about the GET and the HEAD because you still don't want to cache POST requests or whatever. But then you don't check for a Cookie or Authorization header, you would just say: now try caching.

Edge Side Includes

Now, this thing can come in handy with edge side includes, which is a bit like the AJAX thing we discussed before, but on server side. So, it's um, your response that the backend would deliver, instead of rendering everything in one page, it includes URLs to say like: "Here, please put in - copy paste in - this bit that you get from this other URL." And then Varnish will fetch, like look at that and fetch all separate elements and stick that together and then send it to the client. And the very interesting thing for us here is that each bit, like the main frame and then all embedded parts have separate caches. So each of them can have different cache rules. So, for example, you could say this, this login thing, we cached this, would Vary on Cookie and then the rest of the page doesn't depend on the Cookie. And then Varnish will have two parts like this login thing and the actual page. And the actual page is cached as soon as anybody has looked at it. And your login thing is cached as soon as you logged in. So you increase the chances that you have a cache hit.

Symfony has built in support for the edge side includes. So in, in HTML, like what it will output to you is this. So it has this <esi:include thing and this is a URL that in this case would be relative to the URL of the document. And, so the cool thing about this is we have separate caching for each fragment. It's on the server side, so search engines will see one page and not some JavaScript thing stitched together. And in your application it doesn't add a lot of overhead. The, like one issue with this is every single edge side include has to be resolved before the response can be sent out. And with the normal Varnish server it will resolve them one by one. So if you have something with like a lot of includes that can become really slow. There is Varnish Plus, which is a commercial version of Varnish and there they offer this additional feature that the ESI can be

resolved in parallel. You can configure like do at most whatever, 10 requests in parallel. So it's doable.

In Symfony you will activate in the configuration: you say, I want to do ESI and these, like, sub parts of a page should be exposed at this route that you can specify, for example _fragment. The thing is this will, with a naming convention, expose controller actions, even actions that don't have a route. So you really want to make sure that this route is not accessible from outside. Varnish must be allowed to access it, but the public internet should not have access to that fragments route, otherwise they can try and call controllers with weird stuff and you will lose control of what's going on. And then in Twig you would included like this: say render_esi() and specify the controller, you can specify parameters. These parameters, um, can't be objects because they need to be included, it builds a URL from this which must be some sort of string that then Varnish will send. And it's also not the same PHP process that will handle this. So if you, I don't know, if you have some information in the container or if you did some globals, I don't know, that's not gonna work with ESI because it's a separate request that hits the backend. If you have a multi-node setup, it could hit the different node even. So it's really a separate request. So you have to be careful if your application has too much state in it.

Caching by User Context

Now, even with ESI, we still either cache, globally for everybody or we cache individually with the session cookie for one single user. Now the rest of the talk I want to spend talking on user context. So like, example from traveling where they say... we just take this deciding criteria: if you're UK or have a European Union passport, then go this way and everybody else goes somewhere else. And so the distinction is not on the individual passport but it's just: is it an EU passport or a UK passport. So we try to build groups with somehow the same permissions or the same properties. In Symfony, for example, it could be the groups of a user, because Symfony security has this concept of groups already. This can be implemented quite transparently: the reverse proxy does the job and the application doesn't really need to know about it that much. The Hash itself is individual for every user. And, yea, we can like build our own rules how we built that hash. And then again the hash look up will be a request to the backend, but we can cache that. Because if you have a session, like it makes sense if your session, if your hash stays stable as long as your session is going on.

So how does that look? We have a request that comes with credentials. And that request, Varnish goes to the web application and says I need a hash for this user sending along the credentials. And it gets back a response with, with that hash. And the Vary, like this response depends again on the Cookie or the Authorization header. And then we do the actual content request and add the, this context hash to the request and if the response really only depends on the hash, on the groups of the user and not on the individual user, it would reply with it varies on that hash. And then if like on the second time this happens, we would have a cache hit.

So in a more flow diagram, the browser asks for /welcome and the reverse proxy first gets the hash for the user... would get some hash. And then asks for: I want /welcome with the hash is this and gets a response that says, okay, the page looks like this and it depends on that hash. Now, next time we have a request, if it's the same user, we already have the hash. If it's a page that anybody with the same hash, so anybody with the same set of permissions already asked for, we can have a cache hit and say, okay, so here you get this same page that the other user got. And the thing is you're, the web server when it handles the actual request it, doesn't need to know about this hash thing, it just takes the credentials of the current user and builds the page as it builds it for that single user. So you don't need to rewrite your application to to handle it in any special way. You only, at the end, you need to decide is what I did... did that only depend on the groups of the user or was it individual so that you set the correct Vary information.

In Varnish side, it would look like this. So, when we receive a request, we... so here, I use the curl Varnish module, which allows Varnish to send a web request before forwarding the actual request. So here we create a new request and we add a host header, I call auth - that's just the name of the server that handles authentication and do a request on ourselves with all the original headers. And then if the, if the response is not 200, we return with the response status code, most likely that was the credentials were invalid. And otherwise we copy the, this user context thing from the response that we got onto the request and then continue and send the original request to the backend with that hash on it.

And...the reason we don't go directly to the backend here is that we want to also cache the context lookup and with setting this auth header we can, as a first thing in the receive, we can say: "oh do you ask for auth? Then, if you're... if I am myself, like if I was just doing a request on myself, then we go to the backend and force the cache lookup. And if it was any other client asking for getting a hash, we stop that because it makes no sense. It's likely somebody trying to hack into our system.

And then finally on Symfony side, we could do something like this. We take the roles of the current user and then, build a hash from that. So this code is built on top of the FOSHttpCache HTTP cache library which provides tools for, doing this. And actually like this bit with the user roles that's already provided in the library. So writing your own thing would be if you need something else than the user roles.

Wrapping it all up

Does that make sense so far? Any questions at this point or confusion? Nope. Cool. Then I'm going to wrap up at this point. I

presented a bunch of different approaches or solutions, but often they can, they can be combined. So for example, you could cache things even with cookies and use edge side includes for that or use AJAX to do it on client side or mix user context with the AJAX things or even another thing, you move like a lot more of the application to the frontend and build more of an API on the backend. And if you have small enough endpoints possibly you could increase the cache hits with that as well. And of course the whole user context thing, we're using that in an API as well. So with JSON data it works just as well. When you have different API clients with different permissions, that that works fine too.

And depending on your scenario, you could be writing your own information provider for that context hash. For example, the, the thing from the beginning, the newspaper where it has the temperature where you would say the hash actually is your geographical region. Or maybe you have a localized page for every country. Then you could do a hash based on, on, what is your location from the IP and whatever tools you have available to determine that. Or based on the user agent or on some client capabilities. Or maybe even for A/B testing where you would set some kind of A/B cookie and then make a hash based on that and have caching of your A/B testing.

And as I mentioned before, there is a FOSHttpCacheBundle that I'm maintaining together with others. Which besides having all this stuff for this user context handling, it also has support for cache tagging where you mark caches as: this page contains this or that article maybe and when you want to invalidate you can say invalidate everything that is tagged with this information. And it contains a system where you can, in your Symfony application, you can record, okay, after this POST request I want to invalidate this and that cache. It supports annotations on your controllers for, for the caching headers and it also has a system of request matchers where you configure a on this URL and everything that matches this pattern, I want to set some cache headers. Like all old things under /user are no-cache or all things under /static are always cached, things like that. Because with plain Symfony we have the Response object and you would have to set on the Response object cache headers in every controller and if you have lots of controller actions, this can be useful.

Yep, and with that I'm at the end. I'm open for questions if there is any.

Questions

Yes, please. Do we have a microphone? Oh, that's, that's too big to see.

By the way, we use Varnish and it handles millions of requests and it's awesome for us. Anything for using it locally? Because when we are trying to, to go the homepage or stuff like that, it's really painful to always try to add parameters to invalidate the cache locally and stuff like that.

So for us, when we work locally we usually don't put the cache in front. Like we go through, to the Symfony application directly to, to avoid the caching.

Yeah, but we use the ESI include. And for us we, we need to test it locally. Otherwise in staging or live the behavior will be different.

So if you use ESI with Symfony, then Symfony will see, like if you use the thing that I showed... yep, this one. So if you use this construct Symfony will actually see if the client is capable of ESI and if it's not capable, Symfony itself would resolve the ESI includes. So in that case it could, it should be able to render the whole page for you - without Varnish - so that, that should work. The other thing you could possibly do - are you on Symfony or is it something else? No. No. Okay, then one thing you maybe could do is define a separate Varnish configuration for that case where maybe you, in the receive, before it does the hash, like this return hash thing to say I want a cache look up. You could put: I actually want to pass, I never want to do a cache look up. And then you can always go through Varnish and have all the logic the same, but only this little bit is different. And then it wouldn't cache ever. I think that's actually a good idea because you have a system that is very similar to production.

Our idea is to have the same, the same system as live.

Okay. Can you pass the cube over. It's very light. Throw it.

Hi. I'm working with Varnish for a couple years right now and coming from another framework and just edit the esi:include in this other framework. And I'm, I'm using the esi:remove part for, for dealing with local development because this is what I see when I'm not using varnish. So I edit the real rendering and the remove path depending on, some, server environments, the filling out of the environment in the local development mode, and then adding those esi:remove tags with the real content, but still providing esi:include tags, whether you... this is how I can just switch easily between using varnish with the include path and having a local copy with the real output. So maybe this is some kind of solution you want to use as well. And if you want and have just a couple of minutes I can show you on the evening what I, what I did in this other framework.

Thanks a lot. Any other questions?

Hello? Is it possible to use Varnish with ESI in a JSON request?

Yes. It is.

So, the only thing you have to look at, you need to set some kind of flag because otherwise it's thinking, oh, this is stuff I don't know. So I'm not gonna... Sorry, if you do ESI... the thing is if you would, I don't know, if you have some pdf or some binary format and you do ESI on that, then probably that's not what you want. So Varnish says, oh, this is like binary things I don't know. So you have to set some, uh, some kind of flag in the startup flags to tell it. I want you to do ESI on JSON responses. Otherwise it won't try to do it because it doesn't know if that's what you intended. But it is possible.

Hello. Awesome presentation. Thanks. Thank you. I'm wondering about FOSHttpCacheBundle. You said that there is a possibility to annotate, invalidate all, or cache everything. Is there are also option for invalidate some part of cache. Like in this (??) approach invalidate only comments under certain posts. Like [mighty] post.

There is. It's um, so we leverage the ban feature of Varnish itself. Like Varnish itself has that as well where you say: I want to invalidate everything that matches a regular expression and you, you can match on the path. You can also match on certain headers, like the domain name or random other headers.

That works fine, but if you use it too much, you run into problems because Varnish has to keep like, Varnish might have thousands or hundred thousands of pages in the cache. So when you send that request, it's not gonna go into the cache and look at everything to remove that immediately. But rather it says, oh okay. So I, I take note and whenever a request comes it compares that with this ban or with all the bans that you sent it. And then if one of them matches it said, okay, so this cache entry actually is gone. But that means if you have a lot of bans and a lot of requests, you will slow down your requests and at some point that goes... like we had that for awhile when we accidentally did too many bans, where Varnish would repeatedly be unusably slow suddenly because it has so many things to look at before it can return a response because it has to check like, oh, is it banned? Is it banned with any of these things?

So, it exists and it's, it can be useful, but I would be careful when to use it. But the cache tagging, there is a, a Varnish module where it does some sort of binary tree on the caches, so Varnish actually understands the tags and that can be really efficient. So there, if you can tag things, that would be much more efficient if you need to invalidate often. And if it's like very rarely that you say like, oh, whatever, this whole section of the page needs to be invalidated now, but you do that like once a day, whatever, then that's not a problem. But if you have some processes and it does that like hundreds of times an hour, then it's a problem.

Thanks. Thank you. Any other questions?

Uh, hello. How works Varnish with the https together. Or not working?

Um, so the open source Varnish, you would put something in front of it that handles the https because Varnish itself doesn't... like it doesn't... period. The Varnish Plus - the commercial version - there they, they sell some, that they sell, they added something for the https handling. But the, it can make sense to put some Nginx or some, a small... like there is a couple of small proxies that only are really built for that kind of thing. Like just https termination. But Varnish itself doesn't handle it.

Alright. Yeah, I think we're closing here and I'm still around. So if you have any more questions, come up and ask me or ask me later today or tonight or tomorrow and have a good lunch.

Chapter 7: Changing PHP

Tip

SymfonyCon 2018 Presentation by Pedro Magalhães

PHP releases a new minor version every year. Major versions happen when there are enough changes that justify to do so. Who is making those changes and how does that process work? What is the process to get an RFC to vote and the subsequent merge of the code? How can I make my first contribution? Is there anything I can do even if I don't know C? If you are intrigued by PHP internals, this talk is for you.

Okay. So, Hi everyone. Thanks for joining this talk about PHP.

Hi Pedro!

First, a little bit about myself. I'm Portuguese, I'm Pedro Magalhães, by the way. I'm Portuguese, I'm 33 years old. Uh, I'm a PHP developer at the Dutch company called Emesa during the day and I'm a PHP contributor by night. Some nights, at least.

Um, I authored one rejected RFC, which was to get rid of that small "b", uh, before the strings because in PHP you can always put a "b" before any string. Uh, and this is a relic from the past when this was introduced for forward compatibility with PHP 6 and the binary strings and the unicode strings. And that never happened as we all know, there is no PHP 6, but the "b" is still there 10 years now and it still doesn't do absolutely anything. But I tried to get rid of it, and people didn't want to get rid of it. So, it stays. Uh, and well, also one approved RFC - it's the first one that was approved for 8.0.

And uh, yeah, I also tried to contribute a little bit to other open source projects, like I have my very own badge of Symfony code contributor. And, yeah, I like games, and beer, and technology. Yeah, okay.

So in this talk it's called changing PHP and changing PHP can have two meanings, uh, and I'm going to talk about both of them. So one of them is as an objective, um, changing is something in a state of becoming different. And so we will talk a little bit about how PHP is changing lately. Um, and then as a verb. So I really like this definition to make the form, nature, content, future course, etc of something different from what it is or from what it would be if left alone. So this is the two parts of the talk.

Changing PHP as an objective

So, let's start with the first one - with changing as an objective and, um, let's start with a reality check. So this is where we stand today. Um, 5.6 has 24 more days of security updates and then 5 is done for, forever. 7.0 is already out of security updates. Even though there will still be one less release, uh, that was not planned, but there will still be 7.0.33. Um, but yeah, that's it for 7... for 7.0. Then 7.1 is already security only since five days ago. So, currently the only actively supported version of PHP is 7.2 - is the only one that is getting actually bug fixes. But, the good news is 7.3.0 is released today.

PHP 7.3

It's not announced yet, uh, on the website. Um, but, but it will be later today. I promise, by night you can have 7.3 in production during the social event we can do it. Um, and yeah, it will be, it will be nice. But yeah, it's really, really great. It's a great timing that 7.3 is released today. So because of that, let's talk a little bit about PHP... about 7.3.

Um, and let's talk about some of the things that it will bring. So, one of the first that I want to mention is JSON_THROW_ON_ERROR as it kind of advertises, it will be possible to pass chase and throw an error to json_encode() and json_decode(). And it will start throwing exceptions instead of you having to go out and check json_last_error() if something wrong happened. And, well, yeah, sometimes it did, sometimes it didn't. And a lot of the libraries for handling JSON are pretty much just wrapping this up, just checking the json_last_error() and then throwing it for you. But now, PHP can do it by itself.

Then for the OCD ones among us. Now you can use a trailing comments in a function calls. Um, so your, your diffs are going to be much nicer now because you don't have to add a comment to the previous line and all of that. I, yeah. For some people it may be, it may be useful or nicer. Then, array_key_first() and array_key_last() also as the name advertises, it gives you the first and the last keys of an array. Um, and um, yeah, without changing the internal pointer of the array. So, the array stays in the same place where it was.

And this way you can check it. It has, didn't fortunate decision in my opinion, of returning NULL, in case you pass an empty array, I would prefer to throw something, but mostly it's, um, it's, um, it's a good addition.

And also, flexible heredoc. Uh, so for heredocs and nowdocs, um, you can now use this, uh, this syntax where the difference is that the closing marker. So, in this case, the end no longer has to be followed by a new line, so you can actually now close the function call that you are doing a after the closing marker. And the other thing is that the closing marker, I'm basically specifies, um, the indentation of the rest of the text. So, in this case, because END is not really at the, at the, um, at the first column, um, and it is at the fourth column, it means that C will only have one space before it when it's printed.

So, basically, the closing marker now dictates, um, where the, um, where the intention starts for all the text. Then, uh, SameSite cookies - it already existed in Symfony since 3.2. So, this is not news for a lot of you, um, and since 4.2, there was 3.2 is already from two years ago and 4.2, it also added support for that for session and "remember me" cookies. And the SameSite cookies for those of you who don't know, basically it's a way to prevent a cross site request forgery. And, uh, basically the way that it works is that if you put it too strict, uh, the cookie won't be sent if the request is cross site. So, if you are on a different website and you'll make, um, yeah, you click a link to your website, the cookies want to be sent with that request.

Um, but because that will be really unfriendly in some cases, uh, you also have the Lax which allows the cookie to be sent if this is considered a safe request and it's safe request in this case is mostly a GET request, or any "read" request that is a top level navigation so that when you are on Gmail, for instance, and you click on the Twitter link, you stay logged in because, uh, if even if they use the SameSite, if it is Lax - than your cookies will also be sent.

But, in PHP, this is the current signature of the setcookie() function. And so we wanted to add one more - SameSite. Can I join you guys? And the guy said: No! So we had to come up with the, with the alternative syntax for this because this was, this went to vote and it was rejected. And so, the solution that ended up coming forward was this one of setting the cookie and you get the name, the value, and then an array of options. And the array of options is of course, all the other options, so $expires, $path, $domain, $secure, $httponly and $samesite.

More on 7.3. In this case, so, we have a while($foo), switch($bar), case "baz", and then we have a continue. Who here thinks that the continue will continues the while(). Who thinks it will continue the switch()?

Okay. Well, that was not a lot of confusion but in case it is confusing because it can be confusing to some people. Now it will throw also a warning letting you know of that, that the continuous the switch() is just a break and it asks you if you wanted to, if you meant continue to because then you, as with continue and with break, you can add the number to say the number of loops that you want to break out of or continue from.

Another thing also in reflection, in errors, uh, you would see before you would see integer and boolean. But that's not really what you typed down. So now it's, it matches what you use as well. So, now they are always called int and bool, which is nice.

Then we have a new password hashing algorithm if you build it with the, with the password Argon, with the lib... Argon library. Then you can also use now the PASSWORD_ARGON2ID. Um, in FPM, there are also some changes. The entire logging of FPM was refactored and right now you can set the log limit variable to define what will be the maximum length of a line on the log, um, which prevents some before you could lose some information because of, because of the line that was too big. And right now this solves that problem.

And also it allows you now to not decorate the workers outputs because PHP would also always add some information to the, to the output of the workers and now you can actually specify that you don't want that. Um, besides that, also FPM finally got the getallheaders() supports, which is basically to give you all the requests heathers and well, it didn't work in FPM.

Then, another interesting one is the hrtime(), which is the highest resolution monotonic clock and the monotonic clock, basically, it's a clock that's always goes in the same direction. That is a guarantee so that it's not affected by a daylight saving and stuff like that. It will always, always, always goes forwards. Um, and it starts at the unspecified point in time, in the past. It's not important, it's just that you can measure it twice and we will get the most precise difference between those two times possible. So this is really useful also to measure stuff, of course.

And of course there are a lot of bug fixes and performance improvements and stuff like that. But who cares? I'm, this is the truth. And sometimes it's also the truth for contributors, like, ah, yeah, I want to do something new and shiny, but yeah.

PHP 7.4

So, but this is mostly what I have to say about a 7.3. About 7.4 which is coming one year from now. So holds, you have to sit down a little bit to wait for that. But, uh, there it is already showing a lot of very interesting stuff. The first one that I would like to talk to talk about is about the preload, which is already accepted and it's already merged in the master. Um, and it was developed by Dmitry Stogov and to quote what he said because I think it's a good definition.

So on service startup, before any application code is run, we may load the certain set of PHP files into memory and make their contents permanently available to all subsequent requests that will be served by that server or the functions in classes defined on these files will be available to requests out of the box exactly like internal entities. By example, the string length or, or the exception class.

And what does this mean exactly? So, currently, on 7.3, um, so PHP starts and the first request comes. And I imagine that on your index.php where your, your requests are arriving, you do a new A. These will trigger the autoloader because it doesn't know about it yet. So it will trigger autoloader: Hey, what, what is A? And your autoloader imagined that it is composer's autoloader a will then find that file and include it. The file is compiled into OP codes and those OP codes are stored in the OPcache. And after that it will run the constructor of A on the subsequent requests you do new A again. It will again triggered the autoloader, it will, um, the autoloader will include A.php but this time we don't need to compile it anymore because it's already on OPcache.

So now we load the OP codes from OPcache and we run the constructor. On 7.4 with preloads. So we, when we start PHP, we can pass these, uh, these setting, or we can have this ini setting saying opcache.preload and the file in this case, so, for this example, I'm just preloading A.php directly. Typically, it would preload the file, that would include all the other files that you want to preload.

You can only specify one file and then that file is responsible for preloading the rest of what you want. But in this case, for the sake of simplicity, let's say that we just preload A.php, and at this point A.php is compiled and it's in memory. So for the first request, when you do new A, it runs A, um, A constructor because it already knows what it is. It already, it's already loaded in memory. It doesn't need to hit OPcache at all.

So, yeah, all the requests are now immediately... you don't have these intermediate steps. This, of course, there're the downsides, uh, all the files that are preloaded, if you want to change something in them, uh, you need to restart PHP. So, there is no way to invalidate these files. The only way to change them is by restarting PHP.

Typed Properties

Then, typed properties, finally. Um, so yeah, it, it's happening. I'm sorry if you're having a recruiter speak flashbacks. Um, but yeah. So, it's happening. It's going to be on 7.4. Um, and some, some interesting things, so, object types, uh, cannot have a default value unless they are nullable and indicates that they are nullable then you can use null as a default, but you cannot set a default values for, for thing, for properties that are of a specific class and they will not be null. They will be undefined if you don't define them in your constructor. So we have a new states for, for properties which will be undefined, which you shouldn't really encounter if, if you ever encountered undefined is because someone forgot to set the property on your constructor. It shouldn't happen, but, it, uh, it is there.

And, currently this is still being discussed. Uh, but this one was already voted, was already accepted. It's not merged yet because they are still struggling with some implementation details. And the other one, it's currently on discussion. So this one I cannot tell you that it will happen for sure, but I'm pretty confident it will, which is about variance. So in this case, so we imagined that we have a class A, grandparents and then we have a class B which is the parents. And then we have a class C which is the child. And then we define an interface and that interface receives a B, and it returns a B.

We can now, well with variance, we can extend these interface and we can use a contravariance on the, on the, on the, on the parameter types, which means to go up one or multiple levels, so... and, uh, for the return type you can go down. So that's covariance. And, basically, these does not break the, it does not violate the Liskov substitution principle because currently you could already, even just when you have the type hint and on the child, you could just remove the type hint. But this is now more complete, so it will actually, like if your parents can deal with the B, then your child can deal with something that is greater than a B and then can deal with it as the same way that if your parents returns a B, it's safe for the child to return to the C because the C is also a B. So it never breaks, it never violates the Liskov substitution principle.

PHP 8.0

Okay. So for 8.0, um, there is already a change. So I said that, uh, the RFC, um, I have an RFC that was accepted for 8.0 and it's very simple, but it's basically about this. So if we have an array with the, with this initial structure, and then we add one more element to this array, uh, what will be the index of these new element.

You wish, but it's not. Currently, it's not. So, as the documentation says: If no key is specified the maximum of the existing integer indexes, uh, is taken and the new key will be that maximum value plus one, but at least zero. And I really, really, really disliked this and so I got rid of that part. So, on 8.0, uh, it will be minus 41 while currently it's still zero. And because this may bring some, uh, yeah, BC breaks, um, this was, uh, this was a slate that for, for a new major version.

But for the rest of 8.0, um... OK, I have to switch modes. Well, I want you to contribute to, to PHP and to help shape up 8.0. So now it's really the right time to do that. 7.4 still being developed so you can still introduce new deprecations for 7.4 for things

that you really wanted to change on 8.0. So now it's a good time to do that.

Changing PHP as a verb

And so, let's talk a little bit about how you change PHP, about the verb part of it. Um, but first I'm going to tell you a little bit my first, about my first contribution and tell you a little bit the story about my first contribution. Um, and it was like these. So, one afternoon, in an office somewhere, uh, we were having a security scan, um, and certain requests were hanging on certain conditions. Um, and basically it boiled down to the fact that the text protocol of Memcached cannot really handle new lines in the keys properly.

Um, and yeah, because it would hang or it would even allow injection. So this is actually, this was an, uh, an injection for Memcached was possible to, to abuse these, to get that. And this was easy to fix on our side, right? So just, okay, don't accept new lines in the keys. Uh, but yeah, I started wondering like why, why does PHP let these through? What, what, yeah, why would they do that?

Um, and so, luckily in the PHP world, we are in the world of free software and two of these freedoms specify that you are free to study how the program works and change it so it does your computing as you wish. And I wish Memcached to not be vulnerable to injection and then the freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes.

Well, so let's, let's try to get that for everyone. And so I started looking at the code of the PHP Memcached extension and I was positively surprised to figure out that, well it's not rocket surgery. It's not that hard. And the change in the end, in terms of code, ended up being just these and maybe it's not very well visible, but I think that this is reasonable for anyone. This is not very complicated. It's C, it's a little bit, syntax is a little bit different from a PHP, but in this case, not even that much and this was all that was needed to, to cover that, to cover that hole.

So then, um, well now what do I do with this? Uh, I emailed the security@php.net because this is a security concern and they told me, yeah, just open a pull request. Don't mention that it is about security, just open a pull request and uh, and we will merge it eventually our. Okay, sounds good. So I did, I opened the pull request, and it got merged, and since PHP Memcached 3.0.0, that's the problem no longer occurs. So, that's nice. So it's no longer vulnerable to at least this type of injection.

How you can Contribute

Then about your first contribution. First, I wanted to know, did anyone of you already contribute to PHP anything in any way?

Maybe you did, but. Okay. So, let's, uh, first things first. Subscribing to the mailing list is pretty much essential. So this is where all the discussions about PHP happen. Uh, this is where the RFCs are presented. This is, uh, yeah, where a lot of these discussions happen. This is also a nice place to get help if you're trying to fix a bug and you cannot figure a step out, you can always mail internals and there is these very nice website, externals.io that basically just exposes the internals to the outside. Is a nice way to read the mailing lists. It does the whole treating thing and yeah, it's, it's very nice to, to read there. But you can always subscribe to the mailing list on php.net/mailing-lists.php. So this is, for me, it's a good first step.

Help with documentation

Then, helping with the documentation. Sometimes there is ambiguous language that should be fixed or updated documentation or you can help with translations to your own language. If it's not english and it's missing something and you can visit edit.php.net, and it gives you this fancy interface with the facebook like button where you can actually edit the documentation there and you can submit your patch to review and then anyone with a php.net account, if they find that this makes sense, they can merge it. And then the website is a rebuilt every 24 hours. So it might take a while, but eventually your change will be there.

There is talks of possibly migrating these to GitHub, which would make sense and make things a little bit easier. But on the other hand here you can also do it anonymously. So we'll see. We'll see what will happen with that.

Bugs reports

Then - reporting bugs. So, we do have bugs.php.net. Um, and yeah, as with any bug reporting, you should try to always provide the minimum reproducing code. So, okay, this is all you have to do to have this problem. Any other useful information and of course what you expect versus what you actually got. And in this case it's a, it was a bug reported by Nicolas, caused by me and fixed by me as well. So that was nice.

Then, um, you can also answer bug reports provided that you have an email address or not. Maybe you don't really that and that you have some basic mathematic skills - you can solve that problem you can contribute. Um, and yeah, I mean

sometimes you just know how to reproduce it with less codes.

Sometimes you just, you see that the person that posted the question is not giving you enough information, is not giving anyone enough information so you can tell: Hey, can you, please, be more specific or anyone can, can help with this. So replying to bugs is all, so really, really nice. Um, and then who's doing it here, also doing it on StackOverflow, answering people or on a, on the chat of StackOverflow. Um, those are good. Those are good places to help out as well.

And then about fixing the bugs themselves. So, PHP is on GitHub. It's a, it, it's true. it's a mirror of git.php.net. But it has pull requests, you can just do your change, open a pull request, like with any other project and the change can be merged. So, yeah, it's pretty simple to start off.

Um, to help you with the, with your diving into the PHP source. Um, there are some helpful tools. One of them is LXR, which is a way to look at the code a little bit like an advanced idea because you can click on anything and it will take you to that function. And yeah, you can see that one on the room11.org, lxr.room11.org. Um, Room11 is the PHP room on the StackOverflow chat and a lot of PHP maintainers or contributors are also their daily. Uh, so it's also a nice place if you're trying to dig in and you don't, um, yeah, you need some help. It's a good place also to find that help.

Then, you also have 3v4l.org, uh, which is really, really, really great. It's a, yeah, it's a website where you can run a snippet of code and it will run it on 200 plus PHP and HHVM versions so you can see exactly when a behavior was changed in which versions did that happen. Um, and the shout out to Sjon for, for making this and making it available.

Then, you also have wiki.php.net, which has a lot of useful resources if you want to dig in. Uh, so yeah, so is a recommended reading. It has some things that are a little bit outdated, but yeah, you will always be able to find the interesting documentation there. And also on the, on the, on the repository of PHP source itself. It also has the contributing file and it also has some nice information and from these I would say to use your favorite editor to avoid any religious wars. I did this. Um, so yeah, just use your favorite editor as long, in my opinion, as long as you can debug properly with it, uh, feel free to choose whatever you want to use.

Writing tests

Then, writing tests. So, PHP tests look like this. This is a .phpt file. Um, they have a pretty simple structure. So, basically, here we see it is, it has the test name, the script itself, what it will do and what it expects. It's all good. It will pass if the, what expects is not there, uh, it will fail. You have another, you have a bunch of other sections. You can have EXPECTF instead of expects to define it in a format. You can have INI to change some INI setting specific for this test. Uh, you can have a SKIPIF so to, to run a little snippet. And if, uh, if the output is "skip" - then the test is skipped because the extension is not there or for whatever reason.

And then you can run the entire test suits if you run... make tests on the repository or if you want to only test a specific test or a bunch of tests that are related to each other. You can also use the run tests, um, well make test uses run tests, but then you can use run tests directly to specify that you will only want to run a certain type of or certain tests.

Proposing changes

Then, proposing changes. So this is the RFC process. RFC stands for request for comments. Um, and yeah, your approach should go a little bit like this. So the first thing to do is to check what happened the last time that this was discussed. PHP is a pretty large project. It has been around for a lot of years and most likely what you are thinking about, somebody's already thought about and maybe it was already discussed. So that's always a good starting point is to figure out: Okay, what happened the last time that someone brought this up?

Then, if you're convinced, okay, nobody really, nobody talked about these are the reasons why they rejected it before are not valid anymore. Uh, you should email internals and introducing your proposal and listening to feedback a little bit like this first phase of, uh, of feedback and actively listen to the feedback and try to adjust your proposal to that.

Then, uh, you can write the RFC itself on the wiki and it will produce a fancy document looking like this. Um, and yeah, and here you should really detail everything that has to be said about the proposal, like a feature, uh, what, what if it has any, uh, BC breaks. If it, if it has any future scope, like in the future we could also do this, this and that, the proposal itself. So, yeah. Um, and then, uh, preferably, this is not mandatory, but preferably you would also implement our proposal.

Uh, but this is, as I said, it's not mandatory. It happens sometimes people propose things even if they cannot implement it. Um, and then someone else will pick up the implementation, most of the times. Um, versus the SameSite cookie - a thing that I talked before, it was proposed by someone else, but then they couldn't come up with a, with a implementation and I ended up, I ended up implementing that myself. That with the options, a rare.

Um, then, you should start your discussion period, which should be at least for two weeks. And, uh, well listen again to feedback, change things that have to be changed or fight people until they said that, they say that it's not needed to change

anymore.

Um, and then you can start the voting period. The voting period should also last for at least two weeks. Um, people go to the wiki and they can vote on it. Uh, you can, uh, you don't need to have a php.net account to vote. You just need to have a wiki account to vote and some people that are not directly contributors to PHP versus Fabien has, has the, um, the possibility to vote on the, on the RFCs. So, if it is accepted, the PR gets merged into, into PHP.

Questions

And that's pretty much it. Um, I would appreciate any feedback that you have about the talk in that JoindIn a link and a, yup, I would take questions. Does anyone have any questions?

Uh, they will start at 0 if you don't say anything. They will start at 0 anyway. Uh, this will only change if for the first keys that you use are negative, right?

No, sorry.

Okay. Yeah, that's why we'll wait for a major version. Right? So, instead of like I propose these initially for 7.2 I think yet, and they're like: Oh, of course not. It's not an a minor. Are you crazy? Oh, hey, sorry. Uh, then, well, then I do it in a major. Is that fine? Yeah, on a major is fine, okay. Then I'll do it in a major. So, yeah, so that's basically the reason why we will wait for 8.0 to do this change. Because, yeah, it might break some things, or some people if they write code, like if you write code, if you're using implicit keys to add things to the array and then you use an explicit key to access it - you're probably doing something wrong. Right? I mean if you don't care about the keys, you don't care about the keys. Or either you care or you don't. Make a choice. But, yeah. Anything else? Yup.

I shouldn't say. Come on, I did all the effort to not say, but okay, I use the VS code. Uh, and it works really well for me. I use it on Linux and it integrates really well with the GDB for debugging, so I can just add breakpoints and it breaks and I can see the state of the entire application. For me, it works really, really well. But yeah, I won't say more than that. Anyone else? Yup.

No, not strong. Uh, actually like probably if I will have to write C outside of the PHP context, I would be a little bit lost to be honest. Um, but yeah, it's uh, I don't have a strong background in C. I learned a little bit about it in school, but that was it. And by day I work with PHP, so that's what I'm used to. But uh, yeah, as I said before, it's not rocket surgery. It's not that complicated. It's, yeah, you'll get used to it quickly I would say. Anyone else? Okay. Well, then that's it. Thank you.

Chapter 8: Using Symfony Forms with Rich Domain Models

Tip

SymfonyCon 2018 Presentation by Christopher Hertel & Christian Flothmann

With the popularization of DDD people started shifting from anemic models with only getters and setters to a rich model describing the state changes in specific methods. This way of designing models does not play well with Symfony forms. User provided input is inherently invalid while we want to maintain certain invariants in our domain model. A common approach to overcome these limitations is to create data transfer objects our forms are then bound to. This can lead to lots of mapping & glue code that might be cumbersome to write and maintain.

But couldn't we do better? In this talk we will discuss the different aspects of a rich domain model that makes it hard to use it in conjunction with the Form component. We will then look at the possibilities to hook into the data flow of the form handling and discover how we can modify it to interact seamlessly with our model.

Welcome. Hi. Hello. The stage is huge, it's intimidating, but we're pretty happy that so many of you joined us. And we're talking about Symfony forms. So who have you used Symfony forms before? Oh, that's an easy one. Right? And we're talking about Symfony forms using rich domain models. Who's familiar with the term rich domain model. So quite a few and uh, but we are starting with introducing ourselves.

Hello Christian and Christopher

So let's do this. That is Christian, Christian is working for SensioLabs, in Germany and uh, besides being a software developer at SensioLabs he's working as a core member of Symfony software and Symfony docs.

And this guy next to me is Christopher, he's working at SensioLabs in Germany too. And he's also one of the organizers of Symfony user group in Berlin. So if you ever come to Berlin and you're lucky to, to attend the user group, you may meet him there too.

The Use Case (Starting Code)

And Christopher will now guide you through a simple example application that we created to show you the concepts of our talk. So in order to make it easier to understand everything.

Yeah. Well, so we're doing it hands on with a simple use case and our use case is a simple product CRUD. CRUD is an abbreviation for create, read, update, delete. I think you're familiar with that principal, a principal and a, it looks like that. It's a simple interface. Looks a lot like bootstrap I guess. And there are two forms in it. So, one about the category and one about the product. And the domain is a product with a name, a category, and a price. Pretty simple. On the category itself has a name and an optional parent category pretty simple as well. And the price object has an amount tax rate and a currency.

So, uh, that's the workflow of the forms and we're talking just about the product form with a create and edit. So you're quite familiar with that. I guess it's a very simple action that handles, uh, both steps the create and edit. So let's see, at the very basic solution, that's pretty much a, like the best practice document in the documentation. So we're having a product controller with a form action and that's just creating the form type, handling the request, binding request to the form type and then the very typical, if it's submitted and it's valid structure with a redirect at the end or the rendering of the form optional with data and errors or plain.

The form type itself, is very straightforward as well. So we're having a product type with a name category, price amount, price tax and price currency. So we're not having a price object on this level, but three price attributes and the product type. And we're binding the product product class to the type.

Next up, the Product class. So the model, which is the focus of our talk today, right? The model. And we're having a simple, a simple product entity with a private properties with assertion annotations on it and simple getters and setters, right? So pretty straightforward.

The Data Flow

So, um, what's happening now when we are getting our forms. So, um, this is the form as you will see it in your browser and

you will enter data and so on, and your browser will submit an HTTP request. You can see on the headers and then you have the body of the POST request with all the data coming from the form. And inside your your front controller, the incoming request will be transformed into a request object which is part of the http foundation component, which is an object oriented layout on top of the super global. It's basically what PHP offers you to handle HTTP requests and this is what it looks like if we dumped the structure. So you basically have an object around a structured array. And then inside your controller on this side, you have the request object on top, which is then passed down into the handle request method, which means that the request is handled by the form component and we can later check if form was submitted or not. Yes, right here. And here the form is taking into account that we are binding our form to the product class.

This was the final object, all the data that was coming into our controller was bound by the form and is now part of the form object.

Vendor code, Model code, Glue Code

So let's take a look at three columns of our three types of code that are involved to solve our problem. First, there was third party code that we just want to use that we don't want to maintain that we don't want to write ourselves, which means the front controller, which is just bootstrap code from a application. We have the HttpFoundation component as the object oriented layout on top of HTTP. We have the form component of course. And we have the validator component which we have seen in the product model than we have added assertions to our model to, to ensure that domain roots are taken into account. Then we have our domain layout, which means the core part of our business. Everything that's individual for our application, which is basically the product class the category class and the price class or the price price object.

And then we need to write some glue code to somehow wrap all these things together. And this is the controller, which means we are handling the request that we are dealing with a request and returning a response. We need to create a form type which describes how we map our request to our model and if we take a look at this from a graphical point of view, we have the data flowing in from, from the left and going through all the layouts in our model, our model is bound to the form which is then handling the request and putting data back into the model. So usually we want to focus on these yellow blocks, which is the core part of our business. And we usually want to reduce the green part. So the part that is just there for coupling our domain to the framework to the least minimum.

Okay. So that's the basic setup. Probably nothing new for you.

Anemic Domain Models vs Rich Domain Models

And the next step is that we look at our model, um, and we need to differentiate about anemic domain model and rich domain model. What we saw, the product type and the product is a dynamic domain model which is focused on data. So there's a structured way of having data in an object. It's very easy to implement and to maintain even you can generate it using helpers, like MakerBundle or a or the PhpStorm has some utilities as well. And it contains little or no logic at all, so maybe no constructor, no validation logic and setters, and it's not guaranteed that the data that is in their object is valid or is consistent. So that might be a problem.

Rich domain model combines data and logic and it is valid by design. So we have mandatory constructor arguments, for example. So we need to give a category name and category can't exist without a name. It's quite easy to test. Same is true for the anemic domain model, but some might argue it's not really sensible to test the anemic domain model. And we can now create state transitions by name and test them and um, do assertions on transitions.

And that's the last point, defined state transitions and we can, um, the concept is to be more explicit, explicit on the names of state transitions and not just have set name, but uh, maybe rename process.

Important small disclaimer, we don't want to lecture you about using rich domain models is the best way to do it. It's sensible to choose between them. Right? And that's, that's up to you. And uh, this guy, Martin Fowler wrote in 2003 that anemic domain models are anti-pattern and then stack overflow was a bit curious if it's an anti-pattern or not. And then there was discussions how they differ and how to avoid anemic domain models, how to find the right balance between using both maybe. Um, someone came up with this idea. It's not an anti-pattern anymore. Someone says it's SOLID design. Some said it's not SOLID, rich domain is SOLID and stack overflow was very confused. So that's the basic bottom line at stack overflow article about it and how not to agree about software design. And we just want to point out their differences and reasons to use one of them.

Anemic models are good for prototyping, very, very easy to use, easily generated and rich models are more about clean code and object oriented, um, testability and yeah, we can design our domain in it including using name transitions and process. Um, but it's your choice, right? But the next point is we want to show you how to use rich domain models with Symfony.

Rich Models in Symfony Forms

So if we take a look at our former model, we didn't have any defined state methods since we had just setter and getter methods so you would have to call them however you like and then you would have to run validation after. That's not how we want put that. So the first step is to, to, um, require every attribute to be passed when we are constructing the product. So we have to put the name, the category and the price. For category and price we can be sure that both are already valid because these routes apply there too. But for the name we need to, to perform some, some validation. We need to, to make sure that that the constraints that we defined are still applying here. And you can also see that we have removed the validation assertions, so this is something that is now happening inside our, our model. The same, happens here when we are renaming our product and you can also see now that we don't have a set named method anymore, but we have a rename method now so this is more like, like describing what we are doing with our model.

And here's our validateName method, which was just basically checking the length of our name. The price is a bit special because the price itself is not identified by an id or something like that. Um, price, just, well having a value. And this is what makes it so special because one price object can now be shared between different products. But if we are doing that now we want to compare them. So the value must not change. And this means that our price object has to be immutable. We will see that later, what that means for our forms after that. So we're doing all the validation again. And the same applies for the category. So if we are now using this model without making any other changes, we would run into many different exceptions and well that's not really funny. Um, so we need to do something about that to make our forms usable with our new model.

Solution 1: Data Transfer Objects

So the first solution that we came up with is using data transfer objects or DTOs. That's quite easy and looks like this. That's the DTO having public properties and the validation assertions on it. And then we are having some kind of mapping methods where we map the rich domain model to an DTO and back again to an entity. So right here we can check if the DTO, if there are changes and we can explicitly call those rename, moveToCategory or even the Costs method and apply those changes only when we need them. So that's pretty easy to do. And the code looks a bit like the model before, right? Um, the product type is actually the same besides the small change down there, changing the data class. The controller is a bit different because we have to map the DTO from entity and to get the DTO from the form and map to the entity again before persisting it. So quite simple steps, but we now can use rich domain models. Right?

Okay. From the user's point of view, that's not so much that is just changing the interface still looks the same as before the data that is sent is still the same, it's still transformed into the request object, as before. And, the interesting changes are happening here now because as Christopher already explained, the entity is first transformed into the DTO because we need to, to unbind the form to this DTO. We're handling the request here now and here we see the binding to the DTO again, we have the reside here this is the DTO and we need to um, transform this back into the entity. So we are catching the data from form and parking it here. Um, the entity so we can now persist this entity again.

This three column model that we have already seen before. And what changes here now is the right column, because we now have additional glue code. We have written DTOs and this is more code than before, um, which you can see here too. So before, we only had our model here and now we also have a green rectangle on the left and on the right and well basically we want to reduce this part. Now we have have added some, so um, maybe that's not the best solution we can come up with. But the solution enables us to use rich domain models by decoupling the model from the form type and it's very easy to implement and maintain as well. So the DTO was quite simple. Um, but it's additional glue code. Right? And I don't know if you saw that, but we have redundant validation rules, right? We have validation in our rich domain model and we need to have validation in our DTO. So this one has an impact on the maintainability. Next solution we came up with is harder.

Solution 2: Data Mapper

Okay. Who of you has implemented a data mapper for forms before? Okay, one, two, three. Okay. 10 or 20 or so. So I'm not so many and yes, I think it's really the hardest solution, but we want to explain it nonetheless to show you. Um, and I think it's, if you've never used it before, it might be quite fast, but it's not the point to explain data mappers, we just want to show that it's possible to use data mappers as a solution to decouple and use rich domain models. Yes. So our controller now looks again the same as it was in our first solution when we were using the anemic domain, which basically means that we are now again binding the product entity to the form.

And the form is the part where things changed now. First we introduced a new price object which is just extracting the price related properties from the product and put it into a new model. And now we have a data mapper, so a custom data mapper. The form component ships with a default one, but we cannot use that. Um, so we add a new one here and we just say, okay, The data mapper that we want to use is implemented inside, this product type. You could also move it into another class that doesn't really matter here and a data mapper has two methods, methods. The first one is this one is called mapDataToForms and it's populating the form with the initial initial data. So we, basically here take the subtypes and set the data based on the model that was passed here. So the data variable is our product entity.

And after the form has been submitted, we must remap the data from the form into our entity and this was the part where things are a little bit more complicated now because we don't have any setter methods anymore. Um, so we are first getting the data from the form, comparing them to the data inside our entity, and then we call our state transitions, state transitioning methods. So for example, rename here or here, we are moving our product into a different category or the price changes and we are calling this method in this case.

When we are creating a new product, we need to make sure that the form is as populate a new product entity is populated with the submitted data and we are doing this here with the empty data option. Passing it a closure where we get the form and we can extract all the data from the form and pass it to the constructor of our product entity. We need to make sure here that we catch all the exceptions that could occur so that the user does not see an internal server error, but we can present them some kind of a nicer error message.

The data flow, it's still the same, or basically the same. Here's a little bit, a little difference that you can see because we now have the price as an additional nested element, but that's just a small detail here. Yeah, we moved the price values to a lower type, right, so we have a price type right now. Submitted data is still the same. The user does not change what they want to enter there and you can see here that the controller now is the same as in the first example. The request was handled again and here is the difference, we are now not not, the FormType still is bound to the product entity, but the important part now is that we have created our own data method that is used here. And the resulting Product, that's the important part is still maintained. We are basically changing what's happening when our form is handled, but we're not changing our model.

Here our three column model that we are familiar with now. Changes a bit. Now, we don't have DTOs, but we have data mappers and the validator component is gone too because in the process, step one, we introduce in the second step when we introduced our rich domain model, we are implementing our domain assertions by by doing the validation inside our methods and throwing exceptions and now that we don't have DTOs anymore, we also do not need the validator component anymore.

But the glue code part still makes a big part of our application. We have now data mappers that we need to maintain, that we need to change where we need to think about how can we test them, which is yeah, massively different story because testing data mappers really is not fun and um, but you probably want to do that because this is an important part of your application.

So the bottom line with data mappers enabled, enables us to use rich domain models and that requires advanced knowledge about the form component and the extension points of the form component. And this is additional glue code as well. Like it's not a with redundant validation rules we got rid of that but it's quite hard to test and it's quite hard to understand if you're getting in a new project and there's data mappers all over the place.

sensiolabs-de/rich-model-forms-bundle

So, DTO's, redundant data mappers are hard.

Maybe we can, we can ask questions. So who have you after just having heard this, um, would like to use rich domain models with Symfony forms? Oh, okay. That's not so many. Would you prefer. So I think it was five or four people. Would you use DTOs? Or would you use data mappers? Oh, okay. Half and half I think. Okay. But uh, yeah, I mean it's clear that this is maybe not the best solution yet.

We have another one, RichModelFormsBundle. Yeah, we could, we could work on the name. Okay. But it's already out there. So it's RichModelFormsBundle. So, um, you can easily install it. It's still experimental. So we're working on the idea and uh, today is once again a kick off for a discussion, maybe. The installation is quite easy with a tool called Composer.

Bundle Features

Um, and now we're talking about the features. So yeah, basically the idea behind this bundle is, and that's why we have shown you the third solution that the, the data mapper approach maybe could be generalized to something where you do not need to write the code yourself, but you would just configure this generic data mapper so that it does what you want based on your use cases. And um, yeah, now it's your turn. Thanks. Again.

First feature different write and read property path, there is an option already property paths, whether you can configure how the property accessor, um, talks to your model, but the new thing is that we can use, read and write property path, um, different. Because the problem with the initial, um, within default data mapper is that the getters and setters must really be named getName and setName.

In this case, we'll look at this example and um, yeah, we didn't want to have a setName method because in our domain it's called rename. So, um, we thought, okay, why don't we split the property path in, read and write. And yeah, that's the name, read_property_path and write_property_path. So in our case we are saying, okay, when you're reading the value, you just call getName and please call rename when we want to change the value. You could also pass a closure there if you need to

to maybe call different methods based on the value of a checkbox, for example, because you maybe have, enable and disable and not something like set enabled with true and false.

A quick side note, the bundle implements a couple of form options, additional form options that you can use on your form type. So, exception handling. So our domain model throws exceptions on invalid input or on maybe in constructor, like this was the price object, right? Yes. And what you can now use is the expected_exception option. We are not really satisfied with this name because it reminds us more what we're used to when we are, um, using php unit, but in fact we don't want the exception to be thrown. We just want the bundle to intercept here. So yeah, if you have ideas for a better name, we would really welcome any suggestion here. Um, but I think the idea should, should be pretty clear so you can, can define which exceptions can occur here and which are valid to be thrown because they might be thrown by domain validations here and we want to show something to the user, some kind of of nice error message to show them what they did wrong.

Yeah, I guess the, the error handling of the domain is quite important part. We need to tackle because the exception message is not very good for the interface, so we need to, uh, to decouple that. And maybe there are some other validation concepts out there, not using exceptions, but using a message bags and something.

One thing we didn't mention yet is that all type errors that may occur because you passed the wrong data type to the method will automatically caught being caught too. So if you, um, have a model, you will often have realized that you will have to allow null values being there, and this will be intercepted as well.

Mandatory Constructor Arguments

Next one, mandatory constructor arguments. The form component, uh, creates objects, right? And, um, there's an empty data option and you can hand over already created objects. But in this case where we have an immutable value object or a value object, it's quite hard. This also applies for product entity, and yes, we could work around that by writing this closure ourselves, but that's um, 10 lines of code ourselves for every form type. So, um, let's define a new option which is called factory and factory can be the name of a class, which means, um, all the, um, values of this type would be passed to the constructor of this class. You could also, um, pass a closure on any PHP callable, um, so maybe if setting method on another class or so, and this would then return a new object. But only for cases where the form is populated without any initial data object. So when you maybe are creating a new product or so.

And the names of the properties, uh, the names of the fields are mapped to the names of the arguments so you can easily change them, um, to, to have a different order, something.

Immutable Value Objects

Next one, immutable value objects. So once again, the same code, um, but we need to create a new object on change because we can't um change the basic, uh, the base data object, right?

And um, okay, we already have the opportunity to define a factory. So let's just add another option and let's define it immutable because that's what we are dealing with here. Immutable objects and if we set this option to true every time you submit new data a new object will be created and the initial data object that was bound to the form is still the same as it was before we were handling the form.

So about the implementation, controller still looks the same as in the first solution. The form type is a bit different now because we have added all these options that we have shown you before, but that's the part where we now need to make adjustments. From the point of view of the model we are now still using the rich domain model, which is the main difference to the first solution where we were using the anemic domain model. In the data flow there's not so many differences here, here the request is coming in and behind the scenes the important part is now happening as part of a RichModelFormsBundle. And the code, um, we had our custom data mapper implementation before this is now gone because we have the new bundle. And the important part here is the switch or the movement from the right column, which is what we have to maintain to the left column, where there is code that hopefully just works as it is supposed to do and does not need to be maintained by us. So, um, we are here and we can now focus again on our core domain and we don't have to deal with stuff that is not part of our business. And well, we are not making money with it probably.

It generalizes the previous approach of the data mapper, so behind the scenes, um, the bundle implements a couple of data mappers and creepy stuff. Um, but we have additional data options form options and um, they are, the target is using rich domain models so we can improve, uh, the goal is to improve that and it's released, but we're still working on it. I think you tagged 0.1 yesterday or today. Yes.

Some say there are some installations on production already, but I wouldn't do that maybe. It's not feature complete. Rich domain models can be very different because they belong to your domain. And we just, we just targeted those cases we came up with. So maybe you have other targets and use cases and you could easily share them with us on GitHub. That's, that's your part to do maybe, um, yeah, that's a lie. But, uh, I don't know the project, maybe that's not that big. So it's your turn

to help us give us input about, um, maybe the namings of the options and maybe your use cases. Just maybe just try it in your application yet, not in production maybe, but, but on your test server, or on your local development machine. Try it out and give us feedback, what works, what maybe does not work and be nice and where you need to write your own glue code once again. So we want to get rid of the glue code, right?

So, DTOs to sum it up, um, it's easy to implement, duplicated validation, it's additional glue code. Data mappers, hard to implement, hard to test, and additional glue code as well. New Bundle, no glue code on your side, it's experimental and maybe we can bring some of those ideas to the core in the future.

Okay, thanks.

That's it. Thank you.

Questions

Three minutes for questions. Please don't ask hard stuff.

Oh sorry. Does this work? Hello? Can you go to a slide where you actually put the data array the exception in and stuff?

Sorry, I don't hear anything.

Can you go to the slide where you put your extra property paths and everything in there to the form type?

Um, maybe you can ask the question without the mic and we'll just repeat it. I think it's better to understand. Maybe we can. No, we cannot.

This one?

Yeah.

So the question is if you can use...You mean for this or for that? Oh yeah, that's how you use the form component. That's okay.

Can you repeat the question please?

That's, that's part of the interface that we cannot change here because that's coming from the, from the core. So the question was, do we need to pass an array here or can we pass an object? So yeah, that's not, that's not possible.

Yeah. But then we would have to work around much more stuff. Yeah, that's probably not what you want to do.

Alright. Christian and I will be around conference social as well, so see you there. Thank you.

Chapter 9: One year diversity initiative

Tip

SymfonyCon 2018 Presentation by Lukas Kahwe Smith

Last year in Cluj the diversity initiative was announced by Fabien. This talk aims to give an overview how those efforts were organized, what was done as part of the initiative and what progress was made. Both increasing how welcoming we are to people from all backgrounds as well as community reach out to help grow the Symfony community. Finally we will look at what could be the next steps this initiative.

Alright, welcome everybody. Um, I guess it's not such a packed room, so I assume I'm kind of preaching to the choir, which is good because, you know, we're talking mostly about things that happened over the course of the year. And actually I, I'm, I'm, I'm kinda hoping that, actually as a follow-up to this talk on Saturday, we will have maybe a short meeting of everybody that is very interested in this topic to see how, what we can do maybe for the next year. Um, so without further ado, let's start briefly.

History & Why of the Diversity Initiative

My name is Lukas Smith. I work at a web agency in Switzerland. My twitter handle is @lsmith. And um, yeah, so let me actually, the talk is one year of the diversity initiative, but let me start a little bit before that. So kind of, um, in 2017, I think sometime in spring there was a SymfonyCon or Symfony Live in San Francisco and there was an all male, white male lineup speaking. Symfony got called out on Twitter about that.

And uh, and of course there were plenty of very valid excuses why that is the case mostly because there were only applications, you know, respondents that were white males that submitted. And you know, Symfony live in the US. It's much, much smaller than they are in Europe. So, you know, it's, a much, much smaller event. So yeah, it's tricky. But it kinda - I don't know - somehow that that kind of resonated with me in my head. And mostly because, um, I, I kinda, um. So my partner, she is an engineer. She's not in IT, but since I've been living together with her and just getting all these daily things that she experienced in her work in a factory, it kinda made me realize that, that is not enough to just try to be a good guy and to be passive about these things. Like I would have to become active to actually work towards change because the situation was even much worse than I always thought, like in these, before I heard these stories.

There was another thing also that kind of triggered this, and kind of in the same timeframe was we had, um, a female apprentice and she wrote a blog post on our blog about how hard it is for her to go to school and have to explain every day that even though she is a girl, she is studying computer science, right? And just like the amount of emotional, you know, weight that was putting on her shoulders and in fact, in the end she quit, right. Because it just, you know, she doesn't want to explain every day anymore why she's there. Right. So, so both of these things combined with, you know, this, you know, when, when Symfony got called out on Twitter, you know, really, um, I think put it on a map for me that I need to become active.

And, uh, and then I also started talking with other people in the core team and we all agreed, yes, we should do something. We should become more active. We may need to put this topic on the agenda. So briefly, just again, this, this talk isn't really about introducing these concepts, but just briefly so that what we're talking about here, we're talking about diversity on one end, so, you know, the full range of human differences and making sure that, um, uh, or, or, you know, that's one thing, but actually the more important to term is probably inclusion. Um, and, uh, so that is really... so once you have these differences that you actually involve them, that you empower them, right? That's, that's actually kind of like the step that you want to get at. So that's, that's kinda what we decided was the scope of, of this initiative.

Getting to Work

So I started reading a lot in the summer. I actually ended up writing a huge blog post. Um, and at some point I decided I'm not really an expert on this topic at all. Um, so I needed to get some professional advice on this thing. So, uh, my wife and I both decided that we think this is important. So we ended up hiring Sage Sharp to review the blog post. Um, and, uh, uh, and as you know, during the review I realized that, you know, it, it goes even much deeper than what I thought, right? So that, you know, it requires even more education than I thought. So in the end, the blog post was never published. But it helped me quite a lot to, to understand many of the topics there. Um, and so we finally then decided in the core team that we should create an official diversity initiative. It was announced at Cluj last year at SymfonyCon.

The First Year of the Diversity Initiative

So, um, so basically now I'm going to talk about everything that happened since. And so the first thing actually happened at SymfonyCon and, and that was really also a very, a revelatory experience, right? So I start to think more about this topic and I see these badges and icons starting to pop up. So we had on our website, we have the silhouette here. Um, and uh, and so this was the default silhouette if you didn't upload an avatar. And, so again, like this was, this was actually the trigger that, that triggered me, that I thought, "wow, what were you doing" - more this thing was it. Like we had this, this badge, the first badge you get when you join the Symfony community was the, "Hey Dude" badge. And during the award ceremony we had these badges, like, "thx Dude" and things like that.

So these are things that, you know, they've been in the community for years and I thought I was, you know, a good guy, you know, as I explained earlier and I didn't pick up on it all right. I didn't, I didn't pick up on it. And only when I started to actually educate myself, actually invest some, you know, brain cycles on the topic, I started to notice these things that are so obvious, right. Um, and, um, so, so yeah, that's one of the first things that we fixed. So Javier, um, you know, updated those badges and implemented a new solution for the avatars. You know, so, so that's, that's, you know, a quite interesting experience, right? We always say, yeah, but, you know, is it really so bad I have noticed anything, right. And it's, you know, it's kind of a privilege to be in that situation that you can say I didn't notice anything because probably if you really look, you should notice some things. And of course if you're a part of some underrepresentated group, you do notice them and they do every day kind of tell you maybe you are not supposed to be here. Right? And that's not what we want, that's not what inclusion is about.

First Steps: Organization and Transparency

So some of the first steps we did was we created a Symfony Slack channel. Actually, maybe we already did that a few days before the SymfonyCon. So we have a diversity channel on Slack. I created the GitHub organization underneath the Symfony, or no, I created a repository underneath the Symfony organization - diversity - where we started to track with tickets that the ideas we had. And very quickly we had about 30 tickets. Um, I would say about 10 of them have been completed. Plenty of them are still open and we open new ones all the time. But, um, with that we sort of had like the, the initial infrastructure, to get set up.

Um, and uh, um, the other thing that I did was I am, so at Liip, my employer, we have an education budget, so I decided to actually use my education budget to Sage Sharp that I would have monthly one hour sessions. Um, and the idea there was that I could double check some of the things that are going on that I could ask for advice. Um, so Sage is just sending me links about things to consider and things like that, but also it was kind of like a monthly, uh, you know, challenging me, uh, what we have done, right? So kind of making sure that I don't forget about this topic. And that has been a huge help over the course of the year. Um, I think we've reduced it now two 30 minute sessions. Um, but yeah, it's a huge help and I think that's actually a very important. Again, I'm not an expert on this thing at all. I'm, I'm, I'm a programmer and there are a lot of things that you can do that are very well intentioned but that are actually harmful.

So for example, the first thing that I thought, okay, I need to educate myself so I'm just going to walk up on everyone that looks like they could be potentially be part of some underrepresented group and try to get advice from them. Right? And that seems quite well-intentioned, you know, you're trying to get an education from the source, right? Um, but it is actually a very bad idea because if they are, for example, at a conference, they're here to learn about Symfony. They're not here to educate me on things that I can educate myself on probably quite well because there are a lot of things on the Internet about all these things, right?

So while it may be well intentioned, it actually is not a good idea. Now, if somebody from an underrepresented group comes up to you and explains things to you, then listen, do listen, don't dismiss it because it didn't happen to you yet. But don't expect, you know, people from underrepresented groups to educate yourself. So these are all these things that, um, uh, yeah, um, you know, this, um, these exchanges with Sage Sharp have been quite helpful. Um, and that's something that also, again, on Saturday when we discussed maybe this is something that we need to bring to a different format where also maybe some people from the community can benefit from this kind of exchanges.

Now, of course there was negative feedback when the initiative was launched. Actually not a whole lot, but there were concerns and I think... so, you know, the biggest concern was around like who's going to write the code once everybody has been kicked out. And... is suddenly a pull request going to get merged just because it's from somebody that is from an underrepresented group. Like, are we going to reduce the quality? And, uh, I think these concerns are, again, these are things that, you know, I know at least to me, I don't, I don't share those concerns. But it also is not very useful to just push them away and ignore them. That being said, at least I haven't, like, like these were concerns beforehand. I've actually not heard a single person actually point to a specific case where any of that happened. So I hope that maybe over the course of a year we've managed to sort of reduce those concerns by just the mere fact that none of these problems seem to have actually happened that people were concerned about. Um, but it's definitely something that keeps popping up here and there. It came up again also when we then finally implemented the code of conduct, which I'll talk about in a second.

The Social Dimension

But one thing I think that, that is very important that, um, oh, I forgot to click here... is that it is very naive to think that you can have an open source project and it's just about code. Like the second you put a bunch of human beings working on some things, you immediately have a social dimension and ignoring that just doesn't work. And I can very much say that. So I was the release manager for PHP 5.3. And I don't code C at all. So I was the release manager for PHP, I don't code C, it was just a social thing, like trying to get multiple developers to agree to things, um, to decide to finish that these were all very social things. They don't revolve around code and there were a lot of people that didn't like each other that I needed to get to collaborate.

Slack

Um, so there's always a social dimension to open source. Um, now, um, yeah, so one thing that, that also, we did very early on that we started to have discussions on Slack about how we communicate with each other. Um, for example, there was a lot of cursing going on - not really necessarily at people, but it seemed kind of like it wasn't really helping the communication. And so we had a fair amount of discussions around that and I think, you know, this was before we had a code of conduct there were, I think for awhile we had a bot that that reminded people not to curse. Of course people, you know, it's very easy to hack around that, you know, you just replace some characters and suddenly you can still curse. But I think overall I think these discussions, while some, some really flared up and became quite heated, I think none of them really derailed into something really bad. And I think, overall, the experience has improved. I don't really have any KPI's I can show you, but I get the sense that it has improved.

Symfony.com Accessibility

Then the other things we did was to improve symfony.com itself. And, for example, we have, we built this tool for doing automated checks on websites to check for accessibility issues. So I set that up just because I already had a setup at Liip for that. And so quite a lot of improvements were done there. And that's, I think also a very good example. So if we improve the contrast on the website or the font size and things like that or you know, ensure that that colors are not such a key part, it might help make the site readable for some people, but it also makes the site more readable for everyone else. Right. Um, so it's a win for everyone. The win might be bigger for some people, but there is no drawback. Right. So that's a lot also what the diversity initiative did. Like we did a lot of things.

thankyou & victory

Actually I've forgotten on the previous slide, we also introduced the #thankyou and the #victory channel. And it's just also again to just create a more positive atmosphere. And I just love that channel. Like it's, it's like if I need a few good moments, like I go to the Symfony Slack and look at that and scroll up or something like that. And it's, and it's kinda like, it creates this positive relationship to the project that I think is helpful to everyone, right. And hopefully it will make it more welcoming for, for new people to come in, potentially from underrepresented groups, but, you know, just everyone, right? And it makes it easier for them to celebrate their first real victories and get recognized for that.

Docs & Pull Request Language

And other things we did on the Symfony website was, um, we created a little guide, um, to help, um, how to do feedback during a code review. Like how to phrase things in a way that, you know, the other side isn't put down. Because you're just trying to find a better technology solution, right? You don't need to make it about the person. So it's just, you know, some advice on how to do that. So Sebastian did and it was really awesome. Uh, David, he went over, wrote a little script and went over the website just to remove some of the belittling language. Like, Oh, "this is very easy just to x", right? And you know, if you're a beginner and it's not easy for you, you already feel small and you, you, you want to leave, right? So doing these kinds of things, and again, this is not that it helps the specific group, it kind of should help everyone, but it might make us more welcoming, in general.

Code of Conduct and the CARE Team

Then the other thing, and that was probably the thing that took the most effort, we created a code of conduct and we also created the CARE team. And I mean in terms of code of conduct. So Egidijus actually just before SymfonyCon last year, you already created the pull request to add a code of conduct to Symfony. And we had lots of discussions afterwards and not necessarily so much about the code of conduct itself, but more about how to actually, like, if there *is* a problem, if somebody noticed something, how do we do the reporting and how do we make sure that we actually follow-up on the reports in a way that uh, you know, really addresses those concerns.

And so we, we, we studied a lot. Because there are a lot of really good code of conduct that you can just drop in, but there are not that many reporting process. So we spent quite a lot of time researching that, we had several meetings virtually, with,

Michelle and Egidijus and Tobias and Nicolas and several other people discussing this back and forth. We also benefited again, quite a lot from the expertise from Sage Sharp and we finally put, you know, put it all together, I think sometime in late spring. And I, and I, what I really also wanted to emphasize then is that we need to make sure that the people in this CARE team that are receiving these reports are really well prepared. So we ended up doing a donation drive to get extra professional training from Sage Sharp so that these people are ready to, to, um, to deal with these reports. And that was actually already the first little test. Like, is somebody in the community going to sponsor this? In the end it was actually very quickly that we got three companies sponsoring, so we had enough money to pay for the training. And yeah, and I think that's, that's something that's very important that we don't just blunder ourselves forward, that we really, we look for professional expertise where it is available.

And yeah, I think that's also one important thing. Like, a lot of the concerns that were raised about code of conduct was that - and they're always about the worst-case scenario, like somebody will get kicked out of the community. And, I don't know the details of, of the reports that have happened this year. I do know that there have been some. Nobody has been expelled, um, as far as I know. And I think many of the things that that may happen, are miscommunications, cultural differences and things like that, that you might not be aware of. And I think that's very important when you're reading the code of conduct and you're looking at things that, that are stated there that are unwanted... or there is a lot of grayness, in there. Right? And this is not a legal document. We don't have a police, we don't have lawyers. We don't have judges or anything like that. That is beyond the scope of what the code of conduct can really cover. What we're trying to ensure is that there is a process to deal with specific situations and these situations can just be, just be aware that this person, based on the code of conduct has the right to say, I don't want this to happen to me. And then the expectation is that this person that did that listens to that and adjusts their behavior, moves on and everything is fine. Right? But *if* that person keeps that behavioral pattern, then yes, maybe at some point there might be some escalation in the way that it is being dealt with. But in many situations it's just like, listen, learn, move on.

That's it. Right. That's all what the code of conduct is doing. It's just setting up a process for these kinds of feedbacks to happen. Now of course, if something really, really bad happens, then we also have a process in place. But in many cases also if something really bad is happening, then maybe you know, it's not something that the CARE team should actually be responsible for. Finally addressing, they should maybe involve the police or someone else. But also, they should have - and this is also something that is maybe non-obvious - is that "how" to actually, so the person that reported the issue - that, that very severe issue - that they are asked: "do you want the police to be involved" and things like that that you have maybe also some alternative ways to get them help. Right? And these are the things that require some expertise, that require some preparation to be ready when it happens. Because when happens, we need to be able to act very quickly and decisively.

Conference Diversity

Um, so another thing that happened actually in the summer, I was getting a little bit depressed. I thought that things weren't really moving anymore. And so the momentum was gone. Like everybody was maybe patting themselves on the shoulder that we have a code of conduct. We have a CARE team and everybody was starting to lean back a little bit I felt. And then sometime in late summer, Joni from the Dutch PHP conference contacted me and said, she's working on, she wants to build a guide for herself, but also for other conference organizers on how to improve diversity and inclusion. And, and she heard that I might be doing something in that direction as well. And I said, yeah, we have a ticket where lots of people have dumped links about useful information, but we never got around to actually create some, an actual usable guide out of it.

And she took those links and probably a bunch of links she had on her own. And like a week or two she came back with this amazing document, where she really summarized all, like tons of key issues with links to additional details and things like that. And, so this is put up, I made the short link up at the top because the full link is rather long - it's underneath there. So it's https://github.com/DutchPHPConference/conference-diversity-and-inclusion/blob/master/Guideline.md. And so it's full of tips. It's probably not a complete guide yet. We'll, we'll, continue to expand it. One thing that we also want to do and the guide is to highlight a little bit more, which of the different topics require money? And potentially a little bit of scale. How much money? And also which of the things are, require time or, or very little time. So that people that go over this can very quickly identify things that they can probably very quickly implement. And, but they also can see if they, if they can, can somehow manage to do additional things that, you know, these are additional things that they can work on. But also we look at things, how to potentially get money to be able to finance such things like potentially adding a little optional thing people can put on the ticket price, uh, when they buy a conference ticket, to create a budget to enable you to do certain things, which, you know, um, you know, the companies that are paying the tickets, it probably doesn't really matter that much if the ticket is 20 Euros more or less. And so it could be a very easy thing. So lots of ideas like that.

Symfony Events & Scholarship

So at this event here, some of the things have been implemented, not all of them, um, obviously, um, so in some of the things maybe we need to improve a little bit. So we have the scholarship program. Um, so that's cool. We now, previously it was always that it was actually required that the name that you used when you registered for the, for the conference needs to match your photo id and that is then also printed on, on, on these badges - I don't know, I put my badge somewhere. Um, and

so as a first step to, to make it easier for people that are, you know, that have a different name that they prefer to use, we now make at least possible that we use the Symfony Connect name rather than, um, the legal, or photo id name. And you know, these things maybe, maybe you shouldn't, shouldn't even require that, maybe it should be a write-in field or something like that. So these might require some logistics like check-in, and as you notice, check-in is already quite slow. So we need to think about how to make those processes maybe more efficient. But you know, that's the first step.

Pronoun Stickers

And we also have pronoun stickers at the goody desk. Again, this is the first step. I intentionally, I actually wanted to blog about this before the conference to explain a little bit what the idea behind it is. But essentially, while, you know, the, the standard in society for a long time has been, you will allocate a gender at birth - you're either a "he" or "she" and end of story. Many people don't, not fall or, feel that they fall in this very binary approach. Um, and so they might prefer other pronouns like they or even some others. So, uh, it's, it's a good way to not assume the pronouns when you're talking to someone from the first time. Right. And those pronouns stickers would make it easier rather than having the first piece of your conversation being: "What pronoun do you want to use?", you can look at the pronoun stickers and you know, and you can directly jump to the actual interesting content that you want to talk about. So right now we have this pronoun stickers with he, she, they, again, maybe it will be nicer that we would make it write-in. Um, but I do want to encourage everyone, even if you think, well, I'm, I associated with he or she and it's very obvious and it's very simple and has never been a problem. Still use those stickers, normalize the use of pronouns stickers. That makes it easier for the people that do not fall in the classical assignments of the past. So we did that.

We also introduced the quiet room for people to use if they just want, if they sometimes need some space of quietness. But also if somebody needs it for maybe for prayer or something like that. So we introduced the quiet room as well. Um, and you know, just the fact that the topic of diversity is mentioned during the open session I think is already a step as well.

Going to New Places

Um, then, um, another topic a is we. So many other things that I talked about so far, they're very much about making the community more welcoming, right? And safer. But of course, part of what we wanted to achieve is also to grow the community. And again, you know, growing the community, everyone is welcome, right? But there's of course some, some places where we are, you know, obviously underrepresented. Um, and there are many dimensions to this. One of the most obvious and most talked about his gender. And there we are clearly not at the point where we can say that we have actual representation of how many women they are on the planet versus how many men.

But there are plenty of other dimensions that are very important as well. So for example, right now we are very strong in Europe. There's some presence in, in the North America. But, you know, in other continents we're very underrepresented. Um, and so it's very, uh, you know, it's great that now, for example, Fabien is going to Vietnam and to India for some meetups there and trying to jumpstart those local communities. There's discussions about conferences potentially in South America, potentially in Africa. Um, and these are of course very, very, very important things to grow the community because, so for example, if you're in Africa, then, you know, coming to Europe is actually quite challenging. You know, you need a visa and so on. So in fact, actually one of the three scholarships we gave away was from someone from Nigeria. And since we, we took too long to put out the scholarship program in the end when we finally gave the confirmation to that person, they couldn't get a visa in time to come here.

So even though it would have been paid for, it wasn't possible for them to attend. So we instead, we just made the promise that for next year, um, they'll, they'll get that scholarship so that they can apply for the visa in time. So that's, that's great. Um, so I mentioned the scholarship program already. We also, or API Platform, sponsored Rails Girls Summer of Code. Um, they also offered to mentor. Uh, in the end, unfortunately they weren't picked as a mentor. But, what I got from Kevin was actually by the mere fact that we were on the website, there were some contributions coming in just from that. So some students that saw API Platform mentioned on the website then started contributing to API Platform. And that, and that's obviously great. And the other thing I mentioned is the event organization guide, which I hope will be something that more and more Symfony events, PHP events, etc, will start using, um, and, you know, just as inspiration for things to do.

Now. in many ways, we're kind of following the lead of Django. Django, actually, they, they have quite a significant effort around DjangoGirls, which is similar to Ruby Girls, um, with the intention of bringing in new people and giving them a little bit of a crash course on the framework. Um, and that's also something that we discussed over the course of the year. I'm still hoping that we can maybe get that set up. Um, and, but they started in 2014 with that. And, you know, just, you know, uh, from, from what I've seen from the Django Foundation team, it's not like they are now radically diverse. Like it's, it's, it just doesn't happen within even three years of a very, or four years of a very concerted effort that suddenly everything is, you know, diverse and inclusion is done and all these things.

So I'm in many ways I think, I mean, I don't really have any numbers right now and maybe that's also something that we should start collecting, but I don't think that we now have done any, any really significant changes to any KPI that you could

think of, um, in terms of actual numbers of representation. What I *can* say, however, is there have been several people that have personally approached me and said they now feel safe to be in this community, where previously they were unsure. And in many ways I think we've already achieved a lot if we retain, sort of, the people that are here and that they don't get pushed out. So like, the story of the apprentice that I mentioned that she got pushed out, right? If we can get to that point that we're at least don't push these people out, we've already achieved a lot. And if we can then maybe create some word of mouth that this is actually a great place to be if you want to be in IT and you're part of an underrepresented group, this is a great place to be then yes, maybe at some point we can grow the numbers.

The other thing that I think is very important, I think every community that does pay attention to this topic makes it easier for the next one. Because there's a lot of experience that is, you know, written down like we can, like one of the ideas of course with putting up, maybe having a Symfony training program is, we can go to what the DjangoGirls people have done. We can go what Ruby Girls people have done and we can learn from them. Right? So we are now in the PHP community, I think to some degree leaders on this topic and um, that's something that we're proud of. But also we should also help carry that to the other communities. And I think once, you know, all open source communities start to make this a priority, it will become much easier for everyone to get, or to ensure that people don't leave these communities, that they stay around and that maybe new people come in as well.

What's Next?

So what's next? Again, I think that's something we should discuss on Saturday. So at the Hackathon, I would say let's start meeting at 10 in the hackathon space. Um, and uh, yeah, it would be really happy to see all of you and maybe some more there. And, some things that I currently I'm looking into, I've been talking to Jill Binder, she's been very active in the WordCamp community, so Wordpress and doing speaker, speaker mentoring. So she, so she basically has a training program to help people become speakers, which she's trying to, you know, move out of just the Wordpress space and to professionalize a little bit. So that's one topic I'm looking into.

And to me, like obviously, I'm in many ways like, okay, so if we, you know, we get, we get great diversity on speaker line ups, then we will not get called out anymore for non-diverse speaker lineups. That's not the goal. Right. But, you know, but I think having more speakers and having more diverse speakers I think is a very good way of showing people that it is possible to get recognized in the community regardless of what your background is or what you identify as. So I think it is a great way to very quickly communicate that this is a safe place, a welcoming place, if we have a more diverse speaker lineup.

Another topic I, I'm hoping that we can maybe address on Saturday is that maybe we need to formalize a little bit more the, the diversity initiatives. Like right now I basically have this title given by Fabien, but maybe it would be good that we have somebody that says "I'm going to focus on speaker mentoring." Another person: "I'm going to focus on improving events", in terms of diversity inclusion and things like that.

Also, right now we're missing a process for how we deal with the donations. Like we don't have a decision process how we actually use the money that we get donated. So we, we urgently need that in order that we can publish it and be transparent about what we're doing there and also prevent conflicts around that topic.

Special Thanks

So here I, I wrote, I started writing down special thanks. Um, and I, I already missed a bunch. So, for example, Timo, he's been helping me quite a lot in making my, the words that I write actually readable. And Hamza has been helping a lot and bringing a lot of perspective into things. And, and yeah, a bunch of others.

So again, the good news is we have over a hundred people in diversity channel, so it is something that people pay attention to. I'm sure of 100 people, there may be quite a number of people there that are in there because they're still critical of what is happening and they want to pay attention. I very welcome that and I want to get their feedback as early as possible because we don't want to steamroll people. We want to take everyone in along the ride, so to say. Um, yeah, so many people have been, been super instrumental in, in, in different topics that have been pushing forward. And I'm very grateful to them.

OpenCollective

So last thing was already mentioned in the, in a bunch of places. If you can, it will be great if you can consider donating. Maybe also your employer can donate. This was also very much a feel-good moment. So we started the donation drive I think in early November and by the end of November we already had over three and a half thousand dollars collected, which I think given the amount of money that people make with Symfony is probably, you know, very, very small. But, but, you know, last year when I had the first discussion with Fabian about doing such donation drives, we were both concerned: so how would it look if we do this and nobody donates? And so, so in that way, I think that that's already quite cool. But yeah, if we want to be professional about this, then we will need some financial support here and there. So it's great if people can continue donating. And yeah, so there have been plenty of donations from companies. So SymfonyCasts, Liip donated,

Symfony, Blackfire, JoliCode, SensioLabs, and so far they all donated, which is awesome. But I think what's even more awesome, we had a lot of people, individuals donating as well, including one person even donating $50 a month, which is great. Like I understand not everybody can afford it, but if you can, it's great, if you can consider donating.

And again, as I mentioned before, let's meet at a 10 hour at the hackathon space on Saturday and see how we can move this forward. If somebody has to leave early, we can also try to see the Wifi here is pretty good, so we could also have a video conference for people that want to join remotely. Um, yeah. And with that, thank you for listening.

Chapter 10: How Static PHP Analyzer Changed the way I Look at Code

Tip

SymfonyCon 2018 Presentation by Nicole Cordes

Let me introduce you to the world of static PHP code analyzers. I'd like to show you which tools exists, how to use them and how they help you to improve your code quality.

Yeah, thanks for joining my session, or my talk today. As you can see, it's about static php code analyzers and what I learned by using them. My name is Nicole Cordes. I'm a freelancer right now. I started freelancing for PHP applications and backend stuff this year. I used to work in a Berlin agency and next year I will worked for Tideways. Maybe some of you know, it's like profiling php.

Well when I started to use static php code analyzers, I noticed finally after a few weeks, few months I changed my mind on how I code. And I just want to talk about that and to maybe to give you some reasons why you, to start using code analyzers. And before I want to talk about how I changed my, uh, my, my, um, my expectations of my code, I want to introduce you to php static code analyzers and the tools I'm using to analyze my php code.

Tools

And I chose four of the available tools maybe you know, some of them, maybe you didn't hear about it and maybe you even feel familiar with one of those tools on your own. So let's start.

PHPStan

When I, the first time I heard about static php code analyzer was when PHPStan was released on its GitHub page and this was my personal getting in touch with PHP code analyzers. As you can see PHPStan is available on GitHub. You can just install it as a global composer package and finally run it locally on your machine and start analyzing projects. I prepared some screenshots to show you how it will look like. In my point of view, it gives you a lot of information when you run it, but it's strange. The type-safety checks, like does one function return the same type, the other function expect and can deal with.

So as I told you, I have prepared multiple php code analyzers, and each one in my point of view has its own strength and it's own way of showing and dealing with the php code. So when you install it, PHPStan, and run it locally on your project you just run it on the cli and get some output grouped by class. And as you can see, you get some information about ??? names, you get some information about the return type statements. You will get a lot of information, I'm sure in your project. And um, I chose some of them in my later slides and will not stick too much to the output, but I just want to show you how the tools will look like when you're using them in your own project.

PHPMD

The second one after dealing with PHPStan I found there are different projects and services dealing with PHP code analyzers. The second one I want to introduce is the PHPMD the PHP Mess Detector. Maybe some of you already heard that because this is, I think very long-living project already. Maybe even so long-living that it's already was forgotten that it's available. But it still exists and still maintained. And what I like most on the PHP Mess Detector, it can handle your code structure and measurements, I think in a more better way than PHPStan can do it, or at least output the information about those measurements and code analyzations. It's available as composer package and PHAR file as well. And if you execute it you get some more or less written text, as well grouped by file. It's not that nice way like PHPStan outputs, but you can find all necessary and available information in this text and you can even change the kind of output. PHP Mess Detector can output an HTML page where you can navigate through open issues and files as well. So depending on what you need, you can switch the output.

Code Climate

The third one I want to introduce is some kind of service. It's called Code Climate and it's subtitle it has itself is: "Automated code review and quality analytics". As you can see, this is published as a code analyzation as a service, and it has a nice

website. You can connect with your GitHub account and you can just use all your open source projects for free and couple it to your GitHub account and GitHub projects. So this is quite nice to even use it in the existing projects because you just hook in with your GitHub, give some credentials to the Code Climate application, and you can start right away analyzing available projects with just one or two clicks.

After you add your existing project, you will get some really nice information. You can see some nice summary. They use the A to F level to just give you an overview how maintainable your code currently looks like and what they found out. They even can analyze the duplicate code. So you will get some notification if you have multiple files using the same code and you can just refactor your code and exclude duplicate code into own classes and object. So this is, I think this is a more nicely way to get those information then on the cli. So I'm more or less using the services that I'm introducing to you right now.

And just to give you an overview, how you get, how the results are more structured, presented. You can even see the codebase where the information is available. So you can not just see some lines of php files and some information about your code structure, but you can even see the code and maybe compare the information or the remark and the code itself. So it get, you get a better overview what you have to do to resolve the remarks. And you are able to even group depending on some criteria so you can just say, okay, I want to care about naming things and want to ignore all the complexity stuff like that. So I think it's worth to have a look at this.

Sonar

And to be honest, my preferred one is the SonarCloud. Maybe you even heard of SonarQube before. SonarCloud is the SonarQube as a service, it's free for open source as well and it's even, now it's not that easy to run it because you need some preparation in your repository, but you connect with your GitHub account as well. And finally you're ready to start. You have to add some configuration file and some execution in your project. Currently on my project I run some building process or testing process on TravisCI. I think TravisCI or GitlabCI is known by everybody in this room. And one step in the CI configuration is to send... to generate the information and to send it to SonarCloud.

And SonarCloud itself offers like Code Climate, a nice overview in some website style of course, it's a web application. And you have the information about bugs so that trying really to analyze php on multiple levels of php code you have bugs, you have even vulnerabilities. So they have kind of security checks as well. You have those code smells. It's the things I want to talk about today. And um, you have some section even for PHPUnit code coverage. So you can even include your php code coverage reports and the SonarQube and SonarCloud information and see everything for one project on just one page.

Just to give you some other expressions, we can then open the issues so you get more or less a list of issues found for the project. And um, well you have those, you have those grouping and check marking options and sorting filters as well as on the other page. And when you get one step deeper, uh, you even see more, more lines of code of your project for each remarks. So it's, really underlying the problems and you can get some more information about the remarks. They have a huge database where they all describe what, what remarks they are offering to your code and this is really a nice way to be informed and, uh, get some information and impressions how static code is analyzed.

Measurements: Naming

And because I just used some, some newly introductory words, I want to introduce you to the world of static code analyzation. First of all, we have the topic of measurements. Measurements means it's about numbers more or less. So we have a remark, so information about our variable names. We can, we can execute checks: are the names too short? Are the names too long? Do they stick to lower camel-case names? Other services analyze method names. So they do they have a proper length? To methods names which are too short, should maybe be extended to be more readable. There are some tools even can check the uses of the constructor method instead of the class name and they can even go a step further and check the return type of your method and offer some suggestion to rename the method from getFoo() to isFoo() or hasBar() if your method called getFoo() returns a boolean. So there are a couple of information you get about just naming things.

Measurements: Lengths

And as I said, it's all about numbers. Those tools are, also measure your method length. So how many lines of code does your method have? How many lines of code does your class have? How many parameters does your method have? And you can, you can configure your personal values or you can configure your personal likings and say, okay, I want to stick to parameters list of three parameters because if I need more parameters, maybe the method that's, that's too much and I need to refactor my code and group maybe parameters into other objects and stuff like that. It even has some measurements about field count and public field count of public methods. So like saying your class has a lot of public method, you can just configure, okay on my classes should have five public methods, otherwise they doing stuff too complex and I need to rethink or restructure my objects and class information.

Code: Structure

Then they are analyzers concerning your code, your php code. So one of the analyzation is: do you have a lot of commented code? Because I think all of us use some kind of versioning system. We can just simply remove code, there's no need to comment code out. And, personally I would say there's no need to have commented out code for, maybe "I need it later". So just remove that. You can restore code according to your local history in your versioning system.

They analyze how many return statements does a method have? Because when you have a lot of return statements throughout your, um, throughout your method, maybe it's not that practical and it gets too complicated to read your method and to understand where does which return come from. And so my recommendation would be return early but only once. So more or less all my methods have an early return. But afterwards I just have one final return and storing the value in, some variables. Or even I don't need some variables if the return can be passed from conditions. So I can just, I don't need to use extra variables to store the return value and then have the last statement return value. But I can just say: okay, return if empty or if this and that is ????. And if you return early, I consider or I would recommend to prevent superfluous else parts. I think we all have written code where we have, if some condition is fulfilled, return true and then I'm, I used to, I used to stick to else, return false. But if you have an early return, you don't need the else part, you can just say, okay, if I'm not inside this condition that returns true, I just want to return false. There's no need for the else part.

And um, when you're dealing with return type statements, and this is, maybe you, you remember I said the PHPStan, is a really good tool to deal with return type statement and dealing with: does the return type match the expectation. When you have too many return type possibilities, those tools cannot deal with the proper, with the proper type-hinting and say, okay, I'm not sure what this value here is and maybe it's not valid and they give you a wrong, false positives. So stick, or try at least stick to one return type. And my personal recommendation would be try even to prevent nullable return types. So really use empty arrays use empty objects, stuff like that just to have one return type in one method.

Code: Structure II

As I said already, they, the tools are useful to give you numbers of how many methods you are working. If you have the recommendation or if you have the information that's that one of your class uses too many methods, you maybe should think about multiple objects or you maybe should think about refactor the object and split task into two separate objects just to reduce the numbers of methods in one class. One other recommendation from the analyzers is: throw dedicated exceptions that can be caught outside the code itself. Don't throw, don't throw RuntimeExceptions, but throw ConfigurationIsNotSetException or ValueIsNotExpectedException that can be directly caught by some other code around those methods.

And for multiple reasons the recommendation of those static php code analyzers is avoid static calls. Just try to use dependency object that have own method you can call and own return statements that you can deal with. This makes working with your code a lot easier when you have to deal with testing. So static calls in the manner of you want to test your code, you don't want to have that. You don't want to have static calls in your code.

Complexity

And my favorite topics of all this php code analyzers is a whole topic about complexity of your code. So complexity of your code, it's about how hard is your code readable or how hard is it to understand your code. If you have a lot of structure controls inside one method - so like if else, foreach, switch and you have really huge kind of levels inside a method - your method is not understandable anymore and you have too many branches and too many conditions. So this is really, really good information from the static tool analyzers. I got, in my first weeks using them, to learn that they completely changed the way I finally looked at code because, as I said, each control is considered as one logical point: they have a number that increased per control structure. And even if you have, for example, an if statement and using some else or or inside one if statement, even those logical operators increase the count of complexity. And even ternary operators and null coalesce operators account as own increasing point for the, for the complexity.

So if you have a really long method with huge conditions and use logic controls, it's hard to understand them. And I'm sure nobody then nobody else and you will understand the code properly. And so the recommendation from my point of view is merge nested if statements like you don't need to open if condition one, then if condition two then if condition three, but you can just group them into one if statement. So you can merge the if conditions. And use helper functions to resolve those conditions in the first time. So it's more readable to have three proper name method for your conditions. So if: is condition 1 met or is condition 2 met or is condition 3 met, it's much more readable than having those three conditions on one line and not having a clue: okay, what is this test about? What is this condition about? So we factor conditions in own helper method and named them properly. This really helps you a lot to understand what's going on in your own code.

My Conclusions

So after I introduced you on some magic php code conclusions, I want to give you my conclusion, how my code changed via using the PHP code analyzation and what I try to do now for my code and what I maybe want you to ask or you offer to test on

your own. Like I already said, I pay more attention in naming things. I know the, one of the main issues in development is naming things. And when not common with naming things and we have still used problems with naming things. So it forced me to pay more attention in how do I name my variables. And even more attention in how do I name my methods to make it more readable and to make it more speakable so that even other developers who use my code - or you have to work with my code - understand what this method is about, without even having to look into the method, but just by reading the name.

As I said I already refactor conditions and own methods just to proper combine them into one if statement and have all the single conditions and own readable methods and even my methods just return this simply line of code. But because I'm able to read the method name, I know what this check is about and I know why I'm checking this and this explicit condition. So, there's this one thing I try to do each time and as I said, I really try to prevent multiple return types because I think and, and those, those, those two paths go hand-in-hand because I can combine multiple conditions, I can just have one early return and then go on with my proper code of application controller logics and have just one, one line with some conditions that can catch up the preconditions of this method, if the preconditions are not met I can simply return. And afterwards I can just go on and I'm, I can be sure all my preconditions are met because I named the preconditions and already checked it. And well, as I already told you, I really try to use and force myself to only use one return type per method and to even to prevent return nullable return types because this makes dealing with, with the return values a lot of easier even in the, um, in, in the parent code above, uh, outside my method. So I, I don't have to deal with multiple values anymore. I can just say, okay, if this exists or if this property has some kind of value, I can go on. Otherwise I don't care, I don't have to care about that.

So maybe what you think the conclusion is: I really start thinking about code before I writing it and I think this is the most important point in my point of view. I'm not just writing code. I think about how to deal with the code. I think about how, how I can better maintain my code even in the next few days or few weeks or few month. And I think it's even easier for others to deal with the code I'm writing. And the benefits of thinking about how to write code is I think my classes and concepts for applications got a lot of better because I just care about naming thing. I just care about concepts because I want to group them in the proper task and proper the proper groups of methods. And as I already told you, I think my code is more readable because I have more speaking code. I don't need to get into method just to understand what they're doing. And my code gets a lot of testable. So fighting tests, unit tests, integration tests gets a lot easier because I have all those methods available and can really test everything on its own and I can decouple my objects from, from the testing point of view and can offer some mocking objects there because I use dependency injection and stuff like that. So even testing gets a lot easier with the proper code.

Warning

After I give you so many information about why you should change the way you were working with code and why you should start using php static analyzer, I just want to give you some words of - I call it warnings. So when, when I started working with a php code analyzers, my initial goal was to resolve everything. So every remark has some valid point of view and I need to change my code to just resolve every remarks from each and every tool. And, I, I really would ask you, or what I noticed after I did months with static code analyzers is: stop over-engineering. You don't need to resolve any problem or any remark given by this tools. I noticed not all remarks are really valid because you may be depending on some kind of frameworks and this is your code is exactly that way the framework expected to be then you don't have any chance to resolve those problems coming from the code analyzers. Or depending on your time and money in the project, that's maybe not that possible to work on the project anymore or to have the time to make a clean up after you've written the code.

But I really would like to ask you if you see, if you see the output of the php static code analyzer and you don't have the time or you don't have the money to clean up your code, just use those information for your new code. So, I can totally understand that there are projects and, where you don't have maybe the effort, maybe the time, maybe the money, I don't know. There are multiple reasons why you don't work on existing code anymore. But knowing those issues and not learning from them, I think it's the most worst case you can have So if you get the information that some older code has problems or some older code should be refactored, use those information for your new code to prevent those, uh, those, uh, uh, errors, those problems you had before.

And, well, as I said, even sometimes in my code analyzations of my open source projects, I noticed there are really wrong false positive information. So, the tool has given me some kind of information, it's about complexity or it's about naming things. And I think, okay, no this is not that valid. For example, one of the remarks from the SonarQube code analyzer is to store every string value that you use multiple times within one class in constants, to refactor the code and use constants instead of the string values. And this is something I cannot agree with. So this is not a valuable recommendation. And, I, I'm more or less excluded this checks from my project because I think this is some information I don't want to have each time because this is something I won't change in my code.

And as I said, all the knowledge you gained from this PHP code analyzation, even if you change existing code or not, the knowledge, use it for upcoming code to write really clean code. Think about it. Think about how others may see this code and how other would understand, or if others would understand your code. And when you're working on open source projects or have public available projects on your own, just think about contribution, make it more easier to contribute, make your code

more understandable and start really using php code analyzers.

So I hope you get some impression or you get some useful information about code analyzing. Thanks for joining. And maybe one or two have some questions or I even prepared some output. So if you want to get into some output information, we can have a look in the real life example, if you're interested.

Chapter 11: Building Global Web Apps with Multi-region Hosting

Tip

SymfonyCon 2018 Presentation by Jordi Boggiano

This session will explore various setups and case studies from my attempts at hosting sites used by global audiences.

There are many ways this can be achieved with different levels of success, budgets and global-ness.

The talk will touch on Terraform, AWS, global DNS resolution and CDNs amongst other things.

Hello. So I hope you're having a good time so far, we're getting through the first day slowly. So, I'm Jordi Boggiano and I want to talk today about hosting applications across multiple regions or meaning geographically, across the whole planet ideally.

Hi Jordi!

Um, so just a quick word about myself. I've been doing internet things for quite a while now. I've also been leading Composer and Packagist development, a bunch of other open source projects. And for, let's say like work... kind of getting money, you know, because one has to at some point, I work part time at teamup.com and part time for private Packagist, which is kind of helping to pay the composer development as well.

Why Host Across Multiple Regions?

So the first question is why would you want to host something across multiple regions? Because it, you know, it is definitely going to cause you some pain, like it's not the easy way. I mean, obviously I think the main reason for me is that you have used those across multiple places and latency tends to be like painful. So, there was this video that came by the other day, I don't know if you've seen it but it's like the effects of one and a half second of latency in the real world, you know. I mean it kinda shows it's a funny, like a silly example, but it's true that like a little bit of latency can really mess with people and it just makes for a really terrible experience.

Other point is like if you have hosting across multiple regions, one single region can go down and in theory, you know, you should have a resilient system in it. You should be able to stay up even in case of kinda critical host failures, which you know, these days it's like cloud infrastructure, it's like all magical and nothing ever breaks in theory. But you know, sometimes things go bad. Like it has happened that AWS has a complete region down even including like, you know, they have this concept of availability zones which in theory should be completely separated infrastructures. So in one single region you can host like different availability zones. And then, they guarantee you somehow that, you know, if one zone goes down, the others should stay up. This, in the past hasn't always been true. So it's not quite enough if you really want to be safe.

The third point is really, like most of us, I think are using CDN's for at least delivering web assets. This is a fairly common practice and it's very easy. Usually there is lots of tools and websites that help with that. But why don't, why don't we do it for the rest, like if we saw that it is valuable to do it for that. Why not the rest? So, why shouldn't you do it? I mean, I don't know: is anyone here like hosting things like on a global scale or like more than one region? It's kind of hard to see but I see a few hands. But like maybe three to five percent I guess. So this is not so common. So why don't you do it? I don't know. I mean, I came up with a few reasons why I didn't do it until recently.

Reasons Why You Typically Avoid Multi-Region Hosting

Maybe they don't match exactly yours, but, it's just a kind of a lead of where to go and where we could improve things. I think the first reason: we use the database. Like having the database in multiple regions is a major pain. Like this is really, like that's usually already a killer by itself. Like, if you don't have like a multi-master setup, then it gets really complicated to have synchronization across regions and if the master region goes down, then what happens to the replicas? It's tricky, what can you do there? I think like using one of those, you know, cloud databases from the get-go is probably a good idea. I think the issue is like usually people start with MySQL on their computer, and then everything is fine, but then five years down the line when you actually need the global scale, it's like it's too late and it can't rewrite the entire application. So this is a bit of an issue. Um, but like all the providers have some solutions and then you have like MongoDB for example is, you know, it's available no matter what platform you're using. Ah, there's another talk at the moment in the other track if you are interested.

Um, anyway, you know, there are some solutions, I don't have a ton of experience with these so I don't wanna dive in too much, but you know there are things that can help you, but for that, usually you have to write the application really with this mindset from the very beginning.

Um, yeah. What I found is that really like the cloud providers, they sell this magical cloud thing but it's usually, actually doesn't help you that much. Like for at least for this problematic, I found that there are some limitations which are really annoying. Um, like one is Redis. It's just a simple example, but you can replicate Redis for aws within one region, can have as many replicas as you want, no problem. We cannot go beyond the regions. If you want to replicate in another region, there is no way. Um, so, sure you can host your own Redis and, you know, you can do things, but they just don't solve the problem for you.

Um, another issue we had, which thankfully fixed about a year ago was that so this concept of VPC. I don't know how familiar you are with us, but just to explain it really quick, it's like a, it's kind of like your private network for, for all your infrastructure within aws within one region. And, so ideally you want to keep things within the private network and only have one entry point for like the web, you know, like the http port on one entry and that's it. Like the rest shouldn't be reachable from the outside because that's just more secure that way. Um, if you have things in multiple regions, usually they need to talk to each other somehow. Like if you can't connect two VPC's to each other, that means you need to open everything up on the internet and that's like, ehm, like I don't really feel like trusting MySql or Postgres authentication to the Internet. Like, it's just bad things have happened elsewhere in the past. I'd rather not take the chance. So this thankfully has been fixed now so it's fine, but it's actually like guided some of my decisions in the past, which, so that's why I'm mentioning it.

And then finally I think awareness is also an issue in that most of the developers they work with like fast internet connections, they're usually close to their servers and so they just don't feel this pain of latency. Like it's usually, we're not the ones experiencing the problem, it's more like users in, on some random satellite connection or some far away country from your hosting location. Ah, so I think the demand internally is not there usually.

Case Studies

Um, yep, So that's that for the intro. Now I want to look at a couple of case studies. And kind of really the idea is just to share, like, a few approaches I took. I'm not saying this is like the ultimate solution and not trying to sell you the silver bullets. It's just ideas that might help if you are attempting this yourself because what I found is that there's not a lot of information on how to do this stuff. Like I've looked at this for years and it's just, I haven't found a lot of info. Usually those that do it, like mega corporations with insane budgets and they can afford to have, you know, hundreds of Dev ops engineers doing this stuff. Like for most small companies, that's just not an option. So anyway, let's dive in.

Case Study: Packagist.org

So first of all, I wanna, I wanna look at packagist.org, which I guess is something you're all familiar with, so that kind of makes it a bit more interesting, maybe. Um so there we have like just, just to look at what the goals are with what we're trying to achieve, first of all, really high reliability. Because if this goes down, the repository, like with all the metadata composer just fails and things go bad very quickly. People tend to use Twitter very quickly. So, it's like, it has to be up. That's, that's just a reality. It also has to be simple, um, for different reasons, but I just like to keep things simple if I can, because it's, you know, we have a very small team, it's mostly me doing this. Um, so like, you know, this, again, like there's not a hundred Dev ops people that can do like crazy infrastructure. It has to be a simple solution that works well and that just also doesn't break because I can't be like on the watch 24/7, so. It has to be global because we have really users throughout the globe and ideally low cost because this being open source, well there's just not a big budget.

So what did we end up with? Um, I kind of have a self-built CDN. So we have like these primary servers that kind of generate files and all the metadata and then we have a set of mirrors that are spread throughout the world. And those just like synchronize from the primary. So the benefits from doing it ourselves versus using some of the existing providers, uh, is mainly invalidation: because we need really fast responses because when update, like the, they push something, you know, they want to run composer update like 10 seconds later and if it doesn't work, like if it doesn't find the new tag, they just pushed or something, then they come and complain this needs to happen fast. Um, these days, I think there are a few cdn providers which are actually, um, which offers like invalidation to the extent that we need. So I'm looking at maybe switching to, to one of them, but we'll see.

Um, then to kind of route people between the servers, like the replicas we have Route53, which is the DNS solution of AWS, um, which basically does like a latency-based routing. So it's like if you are close to this region you get routed to one of these servers in that region. Then we have health checks on all the servers. So if one of them goes down, or is not responsive anymore from AWS, it just gets killed off, and people don't get routed there anymore. We have like, you know, a few minutes of, DNS TTL. So it kind of, re-routes people fairly quickly.

And like, again, like in terms of simplicity, it's really easy to set up a new one. Like a few weeks ago we had this issue where, um, some of the mirrors were just unreachable for some people. But it was like, the problem is it wasn't a global failure. The

servers themselves were fine, it was a routing issue on the Internet somehow. So the health checks didn't pick that up and it was just, some people were affected. I don't know, probably some of you in this room had some problems. Um, but like, I mean people were saying, you know, I got, I got, I tried at home and it was fine and in the office it's broken, like it was really strange. So, at some point I just couldn't do anything really about the Internet routing sadly, so I just a completely swapped these, these instances and like created new ones in some other, some other regions and like it takes me like 20 minutes or so. It's quickly up to speed. So, um, so that's good.

Uh, so the setup kind of looks like this where we have like multiple replica regions. And you see like for the... so the metadata is being pulled from the, from the primary to the replicas, but the website is not. So the website we only have it in one region still. That's just for simplicity, because yeah, I mean it's a bit more latency for the website users that are far away, but that's just a cost I was okay with. I mean that's a tradeoff you have to make. So when users go to the website, this is just a proxy from the replicas to the main one.

So, what are the problems? Well, as I just mentioned it, this is only the repository, not the website. So it's not a complete solution for sure, but it solves the, let's say the high availability needs we have, that are mostly at the repository level. Um, so that kind of solves the critical part. Another thing that was kinda weird with this is like I had reports of files sometimes like people will get a 404 when running composer. And it took me quite awhile to figure out what was going on, but it's, the problem was the, like as we route people like in a kind of round-robin fashion, like, whenever you resolve the DNS you just go to this server or that server within one region, there was a race condition actually, where one server could be up to speed with the latest metadata but the other one not yet in one, one single region. And so one request would go and hit one server and, like, it would get the filename to get next, and will try and get it and will hit the second server that wasn't up to speed and then you get a 404. and then, you know, if you retry it's fixed, like within seconds, usually it was fixing itself. So it was kinda hard to debug why, because every time someone reported it, I'm like: "I don't know, the file is there, I don't see the problem". Um, so there's a simple proxy hack there where it's just if the file is not there locally, it proxies it to the main region again instead of returning a 404 and that kind of solves it. Um, just, you know, little things that you don't necessarily think of when you, when you get started.

Case Study: teamup.com

So second use case, um, second case study is team up. So that's a, it's a calendar application. So it's also used like pretty well, pretty much all around the world.

Um, so what are the goals here again? Global audience: we kind of need to be everywhere. Um, low latency just because it's not a good experience whenever you're like clicking something if you need to wait half a second, it's just not nice. The high reliability, I say full data access, obviously it's good if you can like, you know, work fully with the application. But the critical really critical part here is accessing the data because we've had times where we were down in the past and people send us emails like completely freaking out because their entire business was down. Like they just, somehow they use the calendar, this one source of information for everything they have to do. Which is good, I mean, it's quite fascinating to see all the use-cases there are out there, but it's just, that means this is extremely critical infrastructure for a lot of small businesses. And like yeah, we just felt really bad about like any downtime because we're like: "oh my God, this is just someone out there is like sitting in the office, like completely lost". Um, I mean it's the same when, you know, when Github is down or something and don't laugh too much. Like we have the same issues with some tools, right? Like industries, like different industries have different bottlenecks. But, and again, one of the goals there was low maintenance because we're a very small team, like only like four or five devs and so it's, yeah, there's just not a lot of manpower to keep this going, so it has to be somewhat self-sustainable and stable.

So what did we end up with? Um, so what we used wasTerraform. I don't know if you're familiar with it, it's kind of like puppet or Ansible, but for like setting up the, the infrastructure. So it's kind of a high level: just you configure all the servers, all the VPCs, all the routing, all the, lots of things. Um, and that allows you to automate things and that means it's pretty good because if you need to add a new region, you can just like copy a few lines and say ok, this, this one is now, like just load this all this config, but for this region and serve that region and you're done. You just run it and creates all the servers and everything.

Again, here we had to make some compromises with what is run where. So we have the primary region that has everything like websites, databases, background workers and all that. And then the replicas they have website, a database copy but no workers.

Um, other kind of trade-off is we're storing the sessions in Redis. And as I mentioned, you can't easily replicate across regions. We thought, well, I mean actually people usually don't go from one region to the next, like, you know, unless you are in a rocket or something, you don't transition from one region to the next that quickly, that losing a session would be a problem. So we just decided to have like local session buckets in every region and that's it. Like there's no, there's no concept of global session. So the reads, like if you were just looking at the data, this is handled locally in every single region. And then, when you write something, um, we're talking to the primary database in the primary region, uh, through this VPC

peering because we built this early this year. So this was available thankfully.

Um, so it looks something like this, with a primary region, some replica region. As you see they're really the same apart from the workers: user comes, does a GET request is handled locally, no problem. Um, if the user does a POST doing some changes, we have the database writes going across. So we started with the region, the primary being in the US west coast and a replica in Europe.

So I don't know if you can spot the problem there, but the result was something like this. It was just not super fast. The reads were going fine of course, because they were handled locally, but like the writes were just horribly slow. Uh, so what happened? Like I felt really stupid when I realized this. I don't know why I didn't think of that before, but obviously every Redis call or SQL query that you do has to go from Europe to the US west coast, which means, yes, somewhere around 100 milliseconds of latency. So, you know, typical pages maybe running like 5, 10 queries. Well you multiply that by 100 milliseconds and you quickly end up witha response time that's actually way worse than hitting the US server directly.

So, yeah, it wasn't a very proud moment. But I thought okay, like I can see there's some issues I can fix here. I was like, one of the problems is already like establishing the connection. So a single, just doing a single query was pretty bad because establishing the MySql connection, including TLS, means usually like two round trips. So already just opening the connection, you're like 200 milliseconds, then you send the query and then it's just, it was really bad. So I thought okay, this proxy SQL I can use this to build connection pools, then we will reuse the connections, so we don't have to have these round trips every single time. So that helped, sure. I mean, it shaved off like these two round trips. Um, but yeah, it was still really unworkable. Like some, some of the pages were doing way too many requests and it was just, so way too many SQL queries and so it was just not really working.

So we changed the approach and I'm like, I'm a very stubborn person, so I didn't want to give up. So what we ended up with, I don't know if anyone else is doing this, maybe it's a crazy solution. Like I haven't had a lot of feedback on this, but feel free to let me know later. Uh, so we proxied the write. So if we see the POST is coming in or like a DELETE request, we say okay, this is gonna do some modifications on the primary database. So we don't want to handle this locally on the replicas, we just proxy the entire request.

The problem is you can't really do this like at the Nginx level easily because you don't have the sessions in the main region. So if you just proxy the request in Nginx that's the easy way to do it, but then you're missing the session data and so yeah, you just find yourself logged out when you're trying to do some modifications that doesn't really work. Uh, so I implemented this in php instead, and so in the application when we see requests coming in and it's a POST or something that's going to modify the content, we say, okay, we'll just take the local session, pack it up in a header forward everything including that header. We also for the client IP, obviously with the usual, like "forwarded" headers and so on.

To make sure that this is, you know, not possible to abuse because the problem is, we're now like, unserializing session data from headers, which, you know, it's not the best idea in terms of security or taking user content and like just dumping it into the session and serializing it, like you want to be really careful when you do things like that. Um, so definitely we do use HTTPS over the proxy link. Uh, it's also going through the VPC peering for additional security, and on top of that, we also sign the requests just to make sure that nobody can inject a request that would kind of deserialize session data.

So with all of this, not too bad. Then if like if this, somehow the proxying fails, then we will go back to actually dealing with the request locally as we would otherwise, and we do the slow SQL request over the ocean and it takes a while then to run, but at least it completes successfully. So, now it looks something like this. Um, so if you do a POST, you come through and then we send the same exact POST, but with some additional headers for the session, the client IP and the signature.

I hope this makes sense. So results kinda like, you know, faster turtle for sure, but still a turtle. So what happened here was like when I tried this, I was like trying to disable the replica and just hit the US servers directly, and I got and average of something like 120 milliseconds, like roundtrip time to, for any kind of request response. It Was around this ballpark. So it's not terrible because like, US is fairly a good link. Um, but then I noticed when I used the replica on the reads, I got about 20 milliseconds. That was super fast, hitting the local server, great. But when I did a POST, well I had this 20 milliseconds to reach the replica and then the replica internally would, for some reason add 200 milliseconds. And I was like, what the hell? I don't understand how it's much slower to execute from the replica within the AWS network and everything, I would think this runs faster than me hitting the US.

Ao yeah, it's just turned out to be, actually the proxy every time had to open a connection. Again you have this round trip time of like opening the SSL connection. So what can we do here? Well, we can again do some connection pooling. So, uh, so this time I added, a local Nginx proxy because we anyway have Nginx running on the local machine. So we just, instead of proxying to the US directly, we proxy to the local proxy, proxy in a way. Ok, it's getting complicated. But, so that way will you, I mean, this adds really nothing in terms of complexities like these ten lines of config in Nginx and like this never fails, it's quite reliable. You just have to make sure that you do a few things like this... oh, that's interesting. This laser pointer doesn't work at all on the screen. Anyway, I shall use the mouse.

Um, so you want to make sure here that you set the HTTP version to 1.1 to make sure that you have keep alive enabled, or you could use two workers, but I don't think Nginx supports it for proxying. Um, and the other one you really need, is this connection, overriding the connection header just in case the client has a connection close in the request just to make sure it's gone. Um, and then you set this keepalive on top.

Um, this kinda, it went well, but at first like I had very mixed results. Like I was trying it and it would sometimes be fast so I had like sometimes the request like a response within like 80 milliseconds. And then sometimes it was still doing this 200 millisecond overhead and I was like, I just don't get it, like it was really random. And then eventually I figured out that this keepalive 8, like the way Nginx works, it actually just allocates these eight buckets and then you have these worker processes in Nginx that says, you know, typically you set this to like the amount of CPU cores you have or double that or something. So, on a big server you maybe have like 16 or 32 of these Nginx processes and each of them actually has eight connections. And so depending on which one you hit, you hit one that has a connection open or not. And so like sometimes it was fast, sometimes it wasn't. And so like, it's just the kind of things that make you lose a lot of time for really small details. But I'm glad I understood where it was coming from because it was kind of not making me feel good to have this somewhat random result.

Um, okay. So what are the other problems that we're having with the solution now? Uh, because with this, now it's consistently faster to hit like the European server, this works really well. Um, the issue is while you're going across the ocean and more, so there is always something that's going to fail. Like there's always one request, that's gonna fail. Like if you do enough requests, some of them are going to fail sometimes. So that's why I kept this fallback step, so that in the worse case we can kind of handle it slower but at least it's handled.

Other issue I had, and this one I didn't get to the bottom of, is the load balancer on AWS sometimes times out these requests. And I just, I don't get it, like looking at all the logs, it seems to be hitting the servers, gets a response like within 20 milliseconds or so - it's not a timeout problem. But for some reason it just gets stuck at some point. I can never get back. So eventually we just had to decide to abandon the load balancer for the proxy request and we just hit the EC2 machines directly, which is not ideal, but that's just. like it, it actually works better in the end than with it.

Other challenges, as we are sending the session via the headers, session size is then limited because the header size has a limit. Um, so this may or may not be an issue for you, like we actually really hardly use the session, so it's really just for marking the user as logged in or not. So there's very little data in it. So this doesn't really hurt us, but, ya know, if you're storing like tons of stuff in the session, it definitely might be preventing this whole thing from flying.

So in the end we got to this point where it's like super fast. Um, so what are the downsides though? Just to quickly recap. In case the primary is down, we're still like in read only state in all the replicas. Like that's just something you can't really fix unless you have multi-master setup. I don't think we're getting there, like with our current team size. Like, AWS announced I think last year that they will at some point release a multi master Aurora, which is like this kind of AWS implemented the version of MySQL and Postgres and whatnot. I don't think this is out there yet, but I'm not sure. I haven't checked in a while. I also don't understand how they can possibly guarantee this will work. I can't just conceptually, it doesn't make any sense to me, but they have very smart people so maybe they figured out a way. It would be really amazing if we could just like switch from using MySQL or Postgres to using like a multi-master MySQL or Postgres, like by just pressing a button, that would be great. But we'll see.

Um, the workers, I mentioned they're only in the primary region, it just makes sense for latency reasons: those are writing lots of stuff. We just don't want to have them all over the place. It's a danger, yes, like if the whole region is down, that means the workers are down, but it's just something we have to live with. Other downside, definitely higher complexity than, you know, the other solution. But it also does a lot more. So I think it pays off for us.

So the end result of this kind of infrastructure: that's the history of all uptime, like from February 14, 2014 to now. We had lots of really bad months where it was like down to 99 percent uptime time. Like this is really bad. I don't remember the numbers but this is like really, really bad. And now in the last 9 months or so since we migrated, we have something that's more up to like 9, like four 9's almost. There was a glitch there in July, not sure what anymore, but otherwise it's really been super stable and we're really happy with this.

Summary

Ok, so just to sum up quickly, I think really this you have to take on a case by case approach. The first point is to look at the audience location obviously. I mean if you're doing some website that's only for like German users, you know, sure you want to probably host it in Germany or somewhere nearby, and you don't need to have like some, some region in Sydney because yeah, it just doesn't make any sense. So that's a per-project thing. I don't know, depends on what you're working on.

All the other issue is really: what are the requirements in terms of latency, like what's okay latency for you. Like, if you know, if you think taking something and you get a response like 300-400 milliseconds later is fine, then that's your number, right? Like you gotta look at what you want to achieve. Like we wanted to really try and bring this to kind of snappy, like instant feels, so

you kind of, it's like, it's like a desktop app kind of. Um, but yeah, that's, that's again something you need to evaluate for yourself.

Then I think one of the big factor in like deciding how complex you go is the team size. Because I think anything is possible but like some solutions require really big teams and that's not always available. Then finally like the tech stack again, like go for a cloud database or wait for Amazon to solve physics and just come up with this magical database, that will do everything great. And that's that. Thank you very much.

Questions

I think we have like one and a half minutes for questions unless I got the time wrong, so I'm not sure if there are any questions. Yes. I don't know if we do microphones here or not. Okay: just shout, I'll repeat it.

> So why are we only running the workers on the primary region?

Uh, because otherwise you would have this latency, like let's say, as they run in the background anyway, like having them near the user has no benefits. Ok, I can't hear you anymore sorry. Let's discuss this later. Anyway, yeah. So that's, that. Enjoy the rest of the conference. Thank you very much. If you have questions, please come by.

Chapter 12: Building Apps for Immutable Servers

Tip

SymfonyCon 2018 Presentation by Daniel Gomes

Immutable Servers are not new and they are not that simple to achieve. However, the advantages of having Immutable Servers versus Snowflake Servers are worth it.

In this talk, I will explain the differences between Snowflake and Immutable Servers and also what considerations you should have while developing your App. I will also share which tools and strategies you can use to build Immutable Servers.

Welcome! Good afternoon! I'm here to talk about building apps for immutable servers. But that's not slightly true. While I was building the talk, I slightly changed the title so it's more scaling apps with immutable servers. And as just a disclaimer, there will be no code. We are not going to build any app. So, you should expect to know more about servers, and how you can use immutable servers for your app and how you should use them, and if you should use them.

Hey Daniel!

So, a bit about me. My name is Daniel Gomes. I am Portuguese. I'm from Lisbon. I'm a software engineer at Teamleader. I am also the organizer of @phplx, the user group, the PHP user group in Lisbon. You can find me on twitter and sometimes I blog.

So, before I start I just want to know, how many Portuguese people I have here. Can you just raise your hand? A few? Probably 10. That's nice.

Topics to Cover

So let's get started. Some topics that I'm going to cover. We are going to cover different types of scaling. We are going to cover configuration drift and management, different types of servers, and also a build process for your web or application server.

The Beginning of a New Project

So, at the beginning, usually when everyone at the office starts to work in a new product or a new application, you start some wire framing, some designs and into the site. And then you start doing the code, there. and then you need to decide our first infrastructure.

But usually, everybody already should do something like this, did do something like this. Is that you put everything in one server because you are just starting, you want to just have something really quickly to the customer to get some feedback. Right? So you end up with something like this. How many have you ever ended up with the application, the database, cron jobs, whatever in the same machine that your application is running. Can you raise your hand?

Yeah, 80% of the room. So, I have been there as well. So, everybody does this and this is okay. No problem. Then you launch your app, everything went well. Then, users start using your application, you start having some rough problems and your server is unavailable. This is a good thing because if your server is unavailable because of the users - it's, it's nice, it's working. You are getting results, right?

Scaling from a Single Server

So, the CPU, memory, whatever it is - is not handling the usage, the load, and then it's the time that you say: well. we need to scale, right? It's easy, right? Scale is easy. Is that easy? Scaling? Do you think? Do you think that you can scale this thing?

You are correct in one point, but in another one - you are not correct. Yes and no. You can vertical scale it, right? That's the first step. So, you add more memory if you need, you add more CPU, more this or you're just fine tuning your php.ini, your PHP-FPM, Nginx, whatever you have. These are the options that you have with the vertical scaling. Otherwise, then you would have more hardware to add more CPU or memory.

Horizontal Scaling

So, and for horizontal scaling. So, if you want to add... so, the goal with the horizontal scaling is, basically, going from one machine to more than one machine. In this example, is 4 instances of your server, but it can be 2 instances, 3, 10, 20. It's depending on your, on your use case of course. So, let's say that with the current server that we had, we create a cluster and we have this kind of architecture. So the question is: Will this work? Of course, not. Yeah? Nope! Why?

Because you have persistence and state inside our server. So, if I do a change in this machine, in these instances, what will happen to the other ones, they will not be in sync, right? So, what is the point of this? Does not make any sense. So, what you need - you need new architecture, this is not the only way to go. This is an example, of course.

So, you put the load balancer on front of your application tier or web point tier you have a cluster of databases, a cluster for AMQP, a cluster for any kind of infrastructure... the persistence that you need. Okay? You can also have your assets in a CDN, whatever. So, and you end up with something like this, right? That's okay. This should work.

Bugs on Production

But yeah, you have it, everything is working fine, it kind of handled the load, but then you have a bug in production. This happens. Who did add a bug in production? Everyone, I think everyone, the ones are that don't raise their hands are a bit shy I would say. So, and yeah, sometimes you are in a rush or it's in the middle of the night and you just need, I just want to fix this thing and I went to go sleep and you do SSH... Oh, I figured out what it is, I change it. It's okay. But in a cluster, this is not a good idea. Right?

So, the problem was solved and you go to sleep. You think. But was it really solved? If you have horizontal scaling. If you only change one instance, do you think it's solved? No. I see some... some heads saying no. So yeah, you are right, it's not solved. This is what happens. So you changed one instance on the top right. That's the fixed one. But the other ones - you didn't change them.

Configuration Drift / Snowflake Server

So, you have an inconsistent state in your... in your instance, in your cluster? This is not good. And, at this point, you enter into the configuration drift. And configuration drift is basically manual adopt changes to your servers that are not recorded. If you don't record them - you cannot reapply them to the other instances. So you have difference, different states between your servers in your cluster. So, this is configuration drift, and this is your cluster after a few months if you continue to do that. If you change the instance with SSH without recording and reapplying to the other instances, you end up with something like this. And what happens is that your servers become a snowflake server.

I don't know how many of you know what is a snowflake server? Some hands.. so, yeah. So probably, most of you, or a huge part, a big part of you will have snowflake servers today that are long running servers. There are no consistency between them in your cluster. They hard to reproduce. So, you cannot just build a new, a new server from the image that you have because if you don't record the changes, you cannot reapply them to a new server that you launch. So, you cannot reproduce. And this is the most important for me - lack of confidence. Do trust your, your server, your cluster, if you have different states in your instances? No, right?

Configuration Management

So, how can we solve this? So, you can use configuration management with automation tools. And the configuration management is just the process of systematically, systematically handling changes to a system in a way that it maintains integrity over time.

Let's see an example. So, let's say that I want to spin up a new server or I have a running server and I went to do some changes to my, to my server. What you need to do is you first change your configuration files and then you run the configuration management to apply that changes into all your servers. So, that way you end up with all your servers in the desire state.

Let's see how this works. So I have my 4 instances. I have my... I want to apply the state D. I run my, my configuration management and after it runs, hopefully if nothing goes wrong, everything will be on state D. Yeah, because even if you run configuration management, something can go wrong if you don't have internet and stuff like that.

So what tools can we use to... for configuration management? There are a lot, there are... These are some that I know. You have Chef, Puppet, SaltStack, Ansible. How many of you have already worked with some of those tools? So, how many of you work more with Ansible or Puppet? Ansible? Yeah, I think, yeah, most of the people I think works with Ansible. I prefer Ansible too. It's more easy, for me.

So, let's do just a quick recap. So you have seen different types of scalings, vertical and horizontal. You have seen what configuration drift is. How you turn your servers, all your services can become a snowflake server and the problems that they

have. And how you can use configuration management to solve those problems. Right? So we are reaching half of the presentation. So this will be more topics so, I think it's the perfect time for you, if you have any questions so far to make it now because probably at the end you will not remember. Any questions so far? Nope. So, let's continue.

Phoenix servers

So, let me present you Phoenix servers. Anyone of you know, Phoenix servers? Only one I think. Oh, nice! So, Phoenix Server! Quoting Martin Fowler: a server should be like a phoenix, like regularly rising from the ashes.

Yeah, that's true. So, if you do this, you will avoid configuration drifts. You have disposal, disposable servers and servers that can be built from scratch, right? Because you can apply everything with the configuration management. So, you're sure always the state when the server is built. So you can just terminate an instance, and create a new one, and you are done. That's nice.

So, let's see an example. So I have my 4 instances. I terminate an instance. Instance terminated. A new one is spinning up. I apply state C, run the configuration management, and voila. It's done. Everything's working fine. Cool.

So, spinning up new servers is not a problem anymore. You are not afraid of doing that, that everything will not be in the same state because you have the tools in place to help you.

But, is idempotence guaranteed here? Do you think? Nope. Yeah, you're right. Do you know why? What if the package repository are down when you are doing this? And imagine that you are having a major outage or you have a problem in your servers and you really need to deploy a new version of your server. And, S3 was down for 48 hours, something like that, and it was a mess. Nobody could download anything because other repositories were there in S3 a couple of few years ago, I remember that, was really crazy: what the... is only S3? And I cannot download anything. I'm not even talking about Composer, installing Apache or Nginx. It can happen.

So, let's say that you have this process that you spin up a new server, you run the configuration management, the configuration management will install packages, create folders, create users, upload the app, and then when it's finished, I have my server ready and in the desired state. So let's say that at certain point the internet or the repositories are unavailable. This means, that I can create the folder, I can create the users, but I cannot install the package. So I will not have my server in my desired state. So, yeah, configuration management is nice, but if I don't have internet that does not serve me. Right? If I need to deploy right now without the internet, it will not work. And that's not good for our business and for you, if you are running critical stuff.

Hello Immutable Servers

So how can we fix this? Anyone has an idea? What? That would work, but if that instance has problems, you're not fixing anything, you are just ensuring that we have a new, but yeah, that would be, could be a solution in it. But yeah, you can have immutable servers - that will help you. Let's see why.

So, according to Kief Morris:

> An immutable server is a server, that once deployed, is never modified, merely replaced with a new updated instance.

This is really powerful. So what does this mean? This means that when I do a build, I have the final image with everything baked in. This means that my application is there, all packages are installed, everything is configured. It's just spinning up a new, a new server and application will start running. You don't need to do anything. You don't need, well, you need the internet because if you are using aws - you need, yeah, but that's... than you cannot do anything right. If you don't have the internet in your local machine and if you need that to spin up new servers - that immutable servers won't help you because that's not the problem of the immutable servers.

You cannot perform any change after the image is built. So, if you want to, if you did a mistake and you want to fix it, you have to build a new image. Okay? And you need to include all scripts, everything you needed for the application to start on boot because you cannot SSH into the machine, you cannot do anything like that. Okay? That's the point and the goal of immutable servers. So, this is really easy to scale out, to deploy and rollback, right? Because let's say that you deploy something, you have a bug, and if you don't save the image of your last version, you have to build everything again to roll back. That is not a good thing. But usually people's save the last image so that will not be an issue, but still.

So, therefore you have much more confidence in what you have in your servers, in your infrastructure. You rely on them. You can... you have reliability on your infrastructure and what's the most important - trust. Your business will trust what you have. You are going to trust, everyone will trust in what you have. Right?

And I think this is also very important for me. You can sleep very quietly and you know that it's not because your server is

having issues that you are not, you are going to wake up. If you have auto-scaling then it will also help you. But let's talk about auto scaling after.

12 Factor App

And also, I don't know if you know the 12 factor app, but the 12 factor app is a methodology for software as a service that basically says these 12 points. I'm not listing all the points here, but it says that you should have this possibility, your configuration should be in the environment for the application. You should have dev and production parity, and the backing services should be attached to your application server. Backing services is your database, AMQP, whatever, anything that is not your application running, not your application code running. So, any service that it uses to run, etc.

And also, this one I discovered while I was doing some research for it. There is the Reproducible-Builds.org that is, they define that a reproducible build is something that you can verify: the path from source code to binary. So, this means that if you, if you give the source code, the build, the environment, and the build instructions - every time that you do that build, you will have the same outcome always. Exactly the same. It's like the checksum. Also, Symfony is part of this organization and there is a blog post on the resources.

Building Immutable Servers

So, how do we do, how do we build it? You need continuous Integration and continuous deployment pipelines. Continuous deployment is not necessary but it would be nice because it makes sense: if you do a change, it can be very quickly to go to production if you're trusting what you are doing, depends on your process, of course. And one very important distinction that you need to do is: a build is different from running your server. If you build - you create an image that can then be launched, can be used to create a new server. Okay? A build just creates an image, period. It's just that.

So let's say that you have this simple build process. This is just an example. So, you spin up a new server, you run your configuration management, you configure the application environment, you have the server in your desired state. Then, you bake in everything that you need for your app and you have your application final image.

Okay? That's a simple one. It's more or less easier, but it can have some issues. Let's say that you want to apply some security updates. It can be more tricky. I usually prefer a multistep build process. This is my preference. Okay? So, basically, I like to have a base image where I have my OS hardening and all common tools that I might use because you might have a server for PHP, you might have server for Go, you can have different servers, but you want your base server to have all the tools that you need for your infrastructure. So, we don't have to configure different things for different types of servers. Okay?

Then I have my application base image where I install all the software needed for the application to run, I create the folders, the users, permissions, everything that you need before you just upload the application. And you put everything that it needs to. Okay? It's just the basic stuff to run: Nginx, Apache, that sort of things. Okay?

And then, the last build would be creating the application image, the final image that you use to deploy to create new servers. So, you upload your app, boots, you, you make any script to boot the application on start up - composer install, to clear cache, all that stuff. That way you know that when you deploy the application, the server is ready to receive traffic. Okay?

So system upgrades, security updates. Usually, I like to do them on the base image, but this means that if I do them in the base image - I will have to rebuild everything. It's a chain, right? If I do a change in the first - I have to build again everything. It's a bit overload, it can be costly in terms of time, but yeah, it's, I think it's a more structured process. It has advantage and disadvantage, but yeah.

So, any application security by application is not your application, not a bug you have in your security... issue you have in your application, it can be a security in Nginx, so a security problem in Nginx or PHP-FPM, or whatever, that should be done in the base image or any configuration should also be done there. So this is basically, this is a multistep process, I like this process, it has a bit overhead. It's not for everyone, but yeah, it's an example.

Building for a Symfony App

So building tips for a Symfony app, you should run the composer install, cache clear, clear any cache that you might have in your application and you build your assets or push them to a CDN. So, there are more but these are the ones that I remembered.

Build Tools

So what tools can you use to help us building our servers? We have some, this is just a subset of them because otherwise it will be a slide full of logos. But I will just mentioned some. So we can use Vault, I don't know, if how many of you know Vault? So, probably one third of the audience. Vault is, basically to store your secrets and that way you don't need to have ENV vars

in your, in your application and all that stuff. Vault will store your secrets for you. And when, in your build process you go to Vault to fetch those secrets, depending on your environment.

Packer, how many of you know Packer? A bit less, probably 10 percent. So, Packer is a really nice tool from HashiCorp. I really love it. With Packer you can build your server for different platforms at the same time in parallel. It's amazing. So you can create your production container. Let's say you want to use Docker, you can create your Docker production container. If you use Vagrant for development, you can also create your Vagrant image, you can deploy, you can create stuff for Amazon, for Google in the same, in same configuration file. It's really powerful. You have Docker, of course, Azure, AWS, Google Platform, Chef, Puppet. So, there are a lot of tools to help us to create these immutable servers.

So, what are the next steps? If you can have immutable servers, you are safe, you can trust in our infrastructure, but then you can start doing more crazy things. So, we can start thinking about high availability, auto scaling. How many of you already do auto scaling here? Hmm, a few. Probably, five percent of the, of the audience. Nice. So, if you have immutable servers - it's really easy for you to just say: Okay, if my CPU is at 80 percent, launch me a new instance. It's easy. It's everything baked in. It's just a matter of a minute, probably, depending on where you have your infrastructure. And now it's built. It should be really easy to auto scale and also to scale down your cluster. Or, if I'm not having that much traffic - kill me some instances and you save money. Also you can auto scale your web tier, application tier. It will always depend on your architecture that you have in your infrastructure. Okay?

How many of you have heard about Chaos Monkey from Netflix? Okay. 2% more or less? Three, five people? Yeah. So Chaos Monkey was a tool that Netflix had the need to create for them to test their own infrastructure. So, basically, Chaos Monkey goes into your cluster and says: hmm, I will kill this instance now. And it shuts down the instance. To stress test their application, how it reacts to a failure of an instance suddenly. That's a nice tool. How many of you would have the courage to just go now to your laptop and say: I will terminate this instance now. And it will run? I saw to it. Yeah, so that's nice. That's a good thing, but many of you will not have that confidence to do things like that, right?

But then Netflix said: Well, one instance in one cluster is cool, is fine. But what if I shut down my entire region? That's Chaos Kong. So they basically say: Okay, you are good in one region if one instance or two failed - you're okay. Let's try shutdown the entire region. Now the fun starts. So the guys are crazy. Of course, many of us will not need anything like this, but yeah, when you have immutable servers and servers that you can trust, you can start thinking about these kinds of things. Of course you are not going to need them always or in every single job that you have or probably in your life. It all depends who you are, because there is not many Netflix out there I would say.

Final Notes

So, some final notes. The goal of this talk is not for you to come out of here and tomorrow, start doing some immutable servers because as you can see it's a lot of work. It's some overhead, and you might not need them. So, my goal with this talk is more for you to think what you have in your application and start thinking, if I want to go to immutable servers, what I should do now to prepare for that. And that's a good exercise that you should do because you always say: Oh, we don't need to scale. But if you don't need this now, let's not do it. But when you know that you are going to need to scale, right? And if tomorrow something happens and your application starts receiving a lot of traffic, you will not have the time to change from vertical to horizontal. Trust me, it's not something like this and it's done. It's not. So, you should prepare really prepare your application, your infrastructure to start thinking about immutable servers to be able to achieve that one day. This does not mean that you are going to need them, so only use them if you really need them. Okay? That's my advice for you because they are really tricky. They are really useful, really nice, but they can be really tricky to get them. Okay?

Questions

I have some resources here. I will probably update this slide. I will then share the slides and update some resources, but there are some resources that you can find out more information about this. Uh, thank you. Uh, I work at Teamleader, we are hiring, so if you like what we are doing and you want to find out, talk with me. Any questions? Questions, anyone? There is one, or I never throw a thing like this - it's my first time. You? Let me see if I'm. Ah, sorry!

Yeah, a state in Symfony. Um, session. For instance, you should not store your sessions in your application server. Otherwise, if the user, the next request goes to another server, the server will not know if it's authenticated or not. Right? If you use the sessions for storing the state of the user for the application for instance.

(Question inaudible)

It depends on your use case. If the authentication user or anything that you have in your session is critical for you, then it's a problem. There is no rule for all the use cases - it depends on your application, what is critical for you or not. So yeah, for, for some applications, having state in, uh, in different states on the session in different places might not be a problem, but for others not. If I'm dealing like invoicing or Ecommerce that you want to check out stuff - that's an issue. Right? Can you throw it?

Oh, there is another one. Oh, that will be art. Can you pass? I will throw for you. Well, I'm not good at this.

Hi. I have one question about logging. For the immutable servers, we have to keep the log, of the logs of the log files of the applications.

No, you should have event streams, um, log streams. So you should log, you should not store anything, logs, anything in your machine.

So before the instance go down, like when it is de-scaling the system, so it should send the log to the streams, ports, or it's like it's a cron job type of thing like that will just upload the log files in a timely fashion manner.

Yeah, yeah, true. When I say terminate is not like brute force terminate an instance. When you terminate the instance, you should have graceful, your application should graceful terminate everything. So, and also in a PHP-FPM and Nginx will not allow you, you can, you can configure it, to only terminate after it's not receiving more traffic and stuff like that. Okay? So yeah, you should be taking in consideration also um, finishing, finishing up the requests. If it's a cron job - you should send a signal for your application to terminate the, the workers and stuff like that. Yeah, there is some preparation for having these kinds of things. That's why it's not that easy.

Thank you.

Anymore questions? One more here. Go ahead. Sorry!

Thank you. Are you... in my company we are using something similar and I was wondering about the overhead. So you have a certain build in production of that image and then of course you push some changes to the master replication and then you get another image, but during the night when you sleep quietly, you'll probably have to deploy the previous image. Right? The one that is in sync with the others. You don't start the full deployment of the new version of the server that you build. This get can get a lot of...

You should create probably a new cluster and when everything is okay - you switch.

Yeah. So there's a lot of overhead involved here. We have a problems this way, we always have to expand infrastructure.

Yeah, but that, but that's the beauty, but you can create a new cluster with all the new instance and then just say: Okay, from now on I want the traffic to go to this site and if everything is not okay, let's move the traffic back. Or just say: Okay, let's try the new version, let's put just 10 percent of our traffic to the new version and if it's okay, let's move everything. So, it gives you a lot of flexibility, but yeah, also overhead,

Yeah, there's like a one... if you don't need the immutable server and maybe you have to consider this when you start moving to this kind of deployment.

That's why I'm saying that you should start preparing for them, but you should not go for them just because you want. You shouldn't go for them when you really need, but you should prepare for them yesterday because you never know.

Any more questions? One more. I don't know. How many time do I have? I still have time.

Thank you. Um, so this works for your application, of course, the immutable servers. But what do you do with databases? Do you have a replication cluster?

The databases? It is a different thing. So, you can have a master/slave, you can have clusters, you can have them in multi-regions, then some sort of sync - that's up to you. But they need to be separated from your application.

Yeah, of course. So that is something to consider separately from... alright, thank you.

Anymore? I think no. So, thank you very much for listening to me.

Chapter 13: Bulletproof MongoDB

Tip

SymfonyCon 2018 Presentation by Jeremy Mikola

An all-too-common approach for database error handling is to log the exception, return a 500 response, and move on to the next request. MongoDB and its PHP driver have an array of features that can greatly improve an application's resiliency in the face of unexpected errors. This talk will examine how the driver monitors connections to a cluster and look at how we can tune its behavior to meet an application's unique needs. We'll also demonstrate how PHP applications can take advantage of newer features such as retryable writes and multi-document transactions to guarantee ACID data integrity without having to fall back to PDO and an SQL database.

Hello, my name is Jeremy Mikola. We have 40 minutes and 58 slides, so I will try to be brief and there probably will not be time for questions. I will be outside and catch me before tomorrow, etc.

Hey Jeremy

Um, welcome. A little bit about myself. I'll make this very brief. I've been in the Symfony community for a while. I missed Cluj last year, but I've been to most of the other conferences. Later join me on the main stage for Jeopardy. I work for MongoDB, it's been about six and a half years now, which is longer than I expected it to be. It's still taking good care of me and they keep giving me more clothes, so I'm happy there. We might have a WurstCon tomorrow, but these days, I work less with Symfony. I work on the MongoDB driver and Doctrine. Andreas is here and there's some other folks from Doctrine here I think this weekend. So that's mainly what I work on.

Topics to Cover

Um, I did not get fired for this. This is MongoDB snapchat for databases. Uh, this was a few years ago. I'm still employed, happily. And so just an overview of what I want to talk about today. The subject was a little strange with bulletproof MongoDB. But I want, kind of a best practices talk and talk about some of the newer features that take us away from, whereas a few years ago this was maybe more relevant that the criticisms to MongoDB, but there's more new features that give it more maturity and stability. So I want to cover.

So I'm going to start by just a show of hands. How many people are familiar with using MongoDB in production or playing around with it first. Okay, great. That's about half. So we'll talk a little bit about deployment models. I think for development environments, a lot of us use just a single mongod process, we don't use clusters or anything. But talk a little bit about that because of the better concepts definitely work with clusters. Talk a little bit about how the driver works specifically the php driver, which makes a lot of, have to do special behaviors because it's, we don't have threads, we don't have a lot of the other features that things like Java and Python have. And then talk about designing applications and the three subjects there are some concepts that you can apply to your application, how, what are the best practices for error handling and how do we take advantage of transactions, which is a newer feature within the last year of having actual ACID transactions that you'd find in a relational database.

The Presentation Abruptly Ends

That's the talk. Please leave feedback on joind.in. Is this going to happen regularly or is there a presentation mode?

Yeah, that's what happens if he doesn't go through the slides quickly. That's a good password. Yep. There we go. We're good? Yeah. Well, I think we're good. Yeah, disable the screensaver. So if you want to know what to do (inaudible). Sorry about that.

And Starts Again: Setups

Thanks Andreas. So the basic is a standalone, this is just running one mongod. This is a good development environment, my production websites, I'll just use this because I have everything on one linode. This is: everything stands alone. You don't want to do this in production.

The two main things that you would want to deploy your database is one with replication, this should at least have three MongoD processes on different machines and your client and driver is talking to the, um, in a cluster, you have one that is the

one that you can write to, and usually you also read from, and then the secondaries are constantly replicating the same data. And they're also communicating with each other to keep aware if one of them disappears. So if the primary server was to crash or fail over, you take it down, someone unplugs the network cable, you still have two out of three and they can take over. And one of them will become the new primary. And your application goes from talking to this primary to talking to the next one.

And so those are, when we talk about failures and recovering from failures, this is a very common thing where a machine goes down or you're doing maintenance or things like that. Uh, and then more advanced, which I expect maybe less people are using is with a shard cluster. And this is where you have your app servers and you would have another cluster of replica sets, another three that store the configuration for where data is stored here. And then similar to sharding in a SQL database, you'll have your data split across, sorry, you'll have your data split across multiple shards.

And typically in this case, you might have a very large collection say... where is it... your user accounts maybe a smaller collection, but your big collection may be all the photos they're posting all the tweets that they're posting. So you would split that across multiple shards. And this would be horizontal scaling, it was discussed in some other talks earlier today. But that's, you're spreading out instead of making a bigger Amazon instance or scaling vertically. And because you can have hundreds or thousands of shards, we have another process that kind of acts like a router so that the driver does not have to, the php driver does not have to open up, in this case, this is a comically large php app where you have fpm workers and they all put in sockets. So you want to avoid the case where we have thousands of connections being opened by every php process. So there we would go through a routing process.

Driver Internals

So those are the two basic deployment models: sharded cluster and then a replica set. And they leverage each other. So sharding, uses replication internally. And then if you're a smaller application, you don't need to scale out, you would just use replication. But the driver different than, you say that the PDO MySQL driver, it will typically talk to one server at a time, whereas most of the MongoDB drivers are responsible for doing, talking to the entire cluster of database servers. And this is also different from, if a driver, if anyone uses Cassandra from the last presentation, I remember going to Cassandra, there the driver talks to one node and that talks to everyone else. So in most other databases, the drivers talk to a single database node at a time.

But taking just a dive through what the PHP driver does when you use it. So we start with a connection string, which is similar to if use Doctrine with a relational database there's a DSN. This is very similar. We have specifications that tell us how to parse this. So it kind of looks like an http url. We have host identifiers, so maybe this case it's one server, but if it's a cluster, it could be comma with multiple servers that you might talk to. And there is additional and DNS if anyone's familiar with SRV records. And this is something that was added recently, instead of having to maintain constantly changing connection strings, if you add more nodes to your cluster, we can use DNS to keep track of that for us, which makes maintenance easier for the very large customers. So this is something that our cloud provider uses. We have a cloud service in MongoDB and it uses SRV records to give you shorter connection strings.

And now once we parse a connection string, we know that's maybe one or two servers that we have to talk to. We need to discover everyone. So we're going to start by doing a handshake. We know a few servers that we can talk to, we reach out to them, we send them a command, and that is basically an exchange of knowing what protocols does the server speak because there's many different versions, and what protocols does the driver speak. And this carries over with, if we're doing authentication, if we're doing compression, a zlib or snappy compression for the protocol messages, we just want to negotiate what features are available. And I'm rushing through a lot of this but the slides are very verbose, but everything references, if you'd like to look this up later, I'll share the slides.

So once the driver talks to the server, it know, well we support this compression. We support this authentication mechanism. The next step is to now we would go through authentication and we'd say I want to start discovering who else is in the cluster. So if it was a replica set and maybe we just connected to that one primary, we do want to reach out and know who the secondary servers are. With a shard cluster it might just be talking to multiple router processes. And so this we call server discovery and monitoring, we abbreviate it as SDAM. It's one of the larger specifications. It's one of the more complicated ones that all the drivers have to do. But this defines how we have a bunch of sockets talking to servers and how we monitor them to keep track of new servers showing up old servers, going away, getting network errors. And before we even talk to the servers, we know from the connection string, maybe we can infer that there's a certain, we think we're talking to a shard cluster, we think we're talking to a replica set, but once we actually start getting responses then we know for sure what kind of servers that we're talking to.

And so this is where things start deviating. And we have a multithreaded drivers, think of like a Java app where they have large app servers, with request handlers and threads, and there's one instance of the driver running on the app server. and then the other example where we had all the boxes on top. In a php app, with typically fpm, you have hundreds, thousands of workers across a large app, a fleet of app servers. And so collectively a single threaded app like php is going to open a lot

more connections on MongoDB side. As well as, just in the individual app server, if you have each php worker, we can't share things between PHP workers because every, in a php application, everything shares nothing with the other. It's kind of a design point of PHP is that we share nothing with the other php app servers. So it definitely complicates things in how we have to deal with talking to a distributed database cluster.

So we need different approaches to monitoring. And in a Java driver, Node JS, if we have a multithreaded asynchronous or we could have a background thread that's in a connection pool, which is more common. And then if you look at php, a lot of database drivers in php, there's no reason to use connection pools because we can't... most sane people are not using threads in php. And so typically a php database driver, we persist the socket. So the PHP request comes in, we'll use the socket, there's only one thing running at a time and when we're done with that php request, we leave the socket open for the next request that gets served. So for an FPM worker, that makes sense. The first worker will open the connection and then every other php request that got served can use the same socket.

And that saves us the time of setting up SSL and going through the whole handshake. We can kind of inherit some of the state that we're connected to. So in a single-threaded, separate sockets would be redundant. We don't do connection pooling, it doesn't really make sense for php's design. And then there's different improvements that we can make for monitoring. So there's a concept of when, just in php itself, when you do, when you work with streams or in other database drivers, you have a socket timeout in php, which is the default socket timeout INI setting. For some reason php defaults that to 60 seconds, even though the PHP execution time is 30 seconds. So it's quite possible that you would timeout talking to doing a fopen or doing a http request in guzzle or something and your php scripts would die giving up. So those defaults are a bit odd. But in the Mongo driver, we have a socket timeout and that's us talking to the database, and then a connect timeout is for us establishing the connection. So logically we'd like this to be a bit smaller than socket timeout. Socket timeout is going to be you running queries and doing other things and those are going to take longer. Connection timeout we want that to be quick: try and connect to the server, if it's not there, we'd like to know as soon as possible so we can return an error.

And there's other improvements that we can make here to kind of do this for a single threaded driver. We don't have a background thread, we have to kind of do everything on demand monitoring. So what we can do is use an event loop internally to just kind of try to monitor all the servers at once, and waste as little bit of your time so that your php script can continue executing.

And so with monitoring implemented, I won't dwell too much on that, the next part is for the drivers to select servers. So the driver connects to a bunch of servers and now we have to, when we do an operation, we need to talk to one. So if we're doing a write, we need to know maybe three or four servers we're connected to, which one is the primary. If we're doing a read, we might be comfortable sending the read to any one of the servers. We can read from a secondary if we want to. So this is a separate spec, just for knowing how to select a server. And for it's very simple for writes, obviously there's only one we can talk to. But again, for a single-threaded driver, we don't have a background thread. So the server selection is kind of integrated with the monitoring. They have to happen in the same logic.

And so how this fits in with PHP, if this is a basic script, we construct a client, we get a collection which is basically a table, drop and insert and do a find. In the very old driver, and even the new one, people expected that the first communication with the server would happen when we construct a client. We're actually, in the PHP driver, we're deferring that until we actually need to talk to a server, so you have no IO happening when we construct a client, it's the first time we know that we have to talk to a server is when we're going to start monitoring, try to get a server for the operation. So the first operation here is going to be trying to run a drop command, which is a DDL, a database definition language command to drop the collection. That needs a writeable server so that at that point we say we need a primary, I'm going to go discover what the topology is, get a primary, monitoring starts. So the first exception you'd have if there was no network connection, it would be from the drop command.

And if you're curious, this is all the output. So it's just inserting and dumping out the data. And so this is internally what happens, we construct, we have a higher level composer package that wraps our php extension. And then server selection, if we look inside what the drop command does, you see the first thing it does is, I need a primary server and then I'll run the operation.

Driver Configuration

And so that's basically trusting the php driver to do monitoring and knowing that it's a bit less efficient than if we had background threads, it would certainly be better, but we have to basically do it on demand when we need to do certain, when we need to access a server.

So, as far as configuring how all this works, there's different use cases, some people's applications want to fail quickly, other people's applications you want the most, you want to avoid errors at all costs and you are comfortable waiting for that. So there's a lot of different options and flags available and these are things that you would pass when you construct your client.

So the first one that I mentioned earlier is your connect timeout, and this is when we do initial socket timeout. I'm sorry, the

initial timeout in milliseconds for opening the connection. And you'll find this, I think every other stream or database driver has something similar to this. And there, our default, which is, probably you would want to lower this considerably, is 10 seconds, which is a safe legacy value. Generally for this value, I think you'd want to customize it to a little bit above your latency to your server. So if your servers are within one second ping time, and usually maybe there are like 50 or a hundred milliseconds. So I might use 250 milliseconds for this. And maybe double it, but certainly keep it below 10 seconds because what you want to optimize for is if I start the php driver and I try and do something and there's no server at all, I don't want to wait 10 seconds, I'd like to fail sooner.

And the second thing is socketTimeoutMS. And this defaults to a comically high 300 seconds. And this is more because of, we use the C driver internally. That's the default. This is the MongoDB driver's equivalent to - in php - the default_socket_timeout. Since our driver no longer uses, originally we did use php streams internally, and this did apply, and now we do the socket IO ourselves. So now we have our own value for this. So it is separate from php's default socket timeout. But the two things you want to consider are what is your socket timeout for other things that you're doing in php. And then secondly, what is your script's max execution time. So on a command line at zero, it will run forever, which is helpful with a large, when you're installing things on composer, it tends to take a while. And for a web environment, I believe max execution time is usually 30 seconds, which is still a long time to service the request, but you generally don't, you want this to not exceed your max_execution_time. So I would also recommend probably lowering this to 30 seconds.

And as far as monitoring, and this is the stuff that's happening in the background, but you have some control over this. We can control how frequently you want monitoring to happen. So a single-threaded driver, we do this less frequently, so once a minute, and we save state between requests. So if you're serving a quick php request and you just constructed the driver, you're not going to hit 60 seconds. This is more for controlling if your PHP workers serves hundreds of requests, maybe 40 or 50 requests down the line, at some point we're going to realize that it's been 60 seconds and we just want to go talk to the servers and see if we have a good sense, if they're the same topology that we knew about a minute ago.

And this helps insulate us form errors, it detects changes if you were to, uh, one of the benefits of MongoDB is that you can scale up and down while your application is running. So if you were to add more members to your replica set or the shard cluster, at the very least, the driver would find out in 60 seconds, 60 seconds intervals.

And the second is a socket check. And this is a special thing for single-threaded drivers as well. In PHP's case we might have a worker that serves a php request and then doesn't get used again for another 30 or maybe five minutes. Who knows, because you have a pool of workers. So in this case, before we use a socket, if it's been longer than five seconds, we'll just see if it's still valid and that's, that way we can recover internally before you get an exception in your application. These can both be configured if you need to, but generally these are good as they are.

And the most relevant things to your application is going to be server selection. So when the driver tries to select a server and we see that happens every time you do an operation, if you're doing a write, we're trying to select the primary, the two main things that you want to focus on here are the server selection timeout, and the try once behavior. And so by default, a threaded driver will wait up to 30 seconds when it tries to do an operation because it has a background thread for monitoring. If your Java application is trying to talk to the database, it will spend up to 30 seconds trying to find a server to talk to, and that's not as bad as it sounds because they have their monitoring constantly in the background so they very quickly pick up changes and they have connection pools so they constantly have available sockets.

And again, php has neither, we have one connection to a server and we use that one socket to do both monitoring and for your application needs. So therefore, we have a behavior that allows us to fail fast. So if we try and talk to a server and it doesn't exist, we immediately throw an exception. And the reason we do this is because, consider the case where we didn't, and we allowed PHP to block up the 30 seconds looking for a server. Your php workers would start piling up, opening more connections and trying to talk to servers that they can't reach, maybe there was a fail-over and so there is no primary, it might take a five or 10 seconds for a new election to happen for the database cluster to heal itself. And meanwhile, you're constantly opening new php workers. So your app servers are getting backed up and you're not serving requests any faster either. And so the most consistent behavior, even though it's maybe not the best thing for MongoDB's case because it can heal itself, the cluster can heal itself, but because of, to cater to php's unique needs, it's better to just throw an exception immediately as the default behavior. And so that's the behavior that we've had historically. And so we kept it with the newer driver that we rewrote about three years ago.

Okay, I'm gonna talk a little bit about later when it turns into deciding how to retry from errors, we can change that behavior and how to go about changing that without. It's not as simple as just deciding to, say try once false and then I'll start waiting 30 seconds. We definitely want to do that intelligently.

Application Concepts

Okay. Just do a time check. Twenty minutes. Halfway point. Okay. So some application concepts, before we get into error handling and transactions. These are things that... concepts that the drivers provide and I don't think they have analogies in a relational databases, but possibly other, um, other non-relational databases.

Write Concern

The first is a write concern, and in this case, ideally a production app is talking to a replication cluster, so your data is being written to the primary and then replicated to the secondary nodes. And again, to give us insurance so that if this guy dies, one of these folks can step up and be the new primary. And in some cases they might sync directly from the primary and in this, yeah, in this case, they're both reading from the primary, but they're not all getting the data at the same time because they basically tail the activity on the primary. When you use a write concern and you issue a write to the database, by default, we'll send it to the primary and if it succeeds, the driver gets back control, you go on and do your business. And you're not really waiting for it to replicate. But ideally, consider the case where you wrote to the primary and the data you inserted didn't get a chance to replicate. And then the primary died. From your application's perspective you think everything's succeeded. Uh, when in reality it's possible that that data permanently got destroyed with the primary. And when one of these secondaries steps up, the data is basically not there anymore because it was never replicated.

So the way to control this in an application, and this is always at the expense of, if you want more durability in a distributed system, you're going to wait longer for it. And so you're gonna decide to use this. Some things are more important than other things. If someone is doing a financial transaction, you would like to make sure that the data is persisted and can survive any kind of disaster scenario. And so the write concern you would want to use is just saying I'm concerned with how this write gets acknowledged in the system - would be a majority write concern. And this is a, you can either use a number, but majority is a nice string that if you grow your replica set to seven nodes or 10 nodes, it's always going to be the majority number of them, which, and as long as the majority have acknowledged the write, it can survive any disaster scenario. So if we used majority here, the driver would not get a response until at least one secondary, at least two nodes have acknowledged the write. So this is generally what you want to use. And again, it is a tradeoff: if you did this for every write in your application, you're probably wasting time. But there's probably some things in your application that aren't as important to wait for. If you're doing a log messages or small tweets, it's not as important as someone updating their password or storing a billing address or something like that.

Read Concern

On the flip side, we have a read concern, and this is when we're querying the data: do I just want to go to the primary and read whatever's available? Or do I want to make sure that I also read data that is acknowledged by the majority of the system? And so as one applies the writes, the read concern is obviously going to make your query slower because maybe you're waiting for the data to be replicated or there's, you're going to purposely read a bit older data that is safe to read because it exists throughout the cluster instead of the immediate write that was just written only to the primary. So this is a lot to take in, and it's something to research on your own and find what works best for your use case. But I will call out the linearizable. This was something that was introduced about two years ago and if anyone is familiar with the Jepsen tests, which is the call me maybe distributed database tests, a guy, Kyle Kingsbury runs a kind of a test framework for basically experimenting with databases and finding out all the ways, how to break them basically. And so he had tested an older version of MongoDB years ago. He's tested many databases over the years and predictably the older version of MongoDB did not get a glowing review. And he found all sorts of ways to break it and like write to the database and shut down some nodes and realize that your data disappeared. And so this was something that around the time of Mongo 3.4, which is about two years ago, they put a lot of effort into addressing and making sure that we can pass that test suite.

And now that, his test framework is now part of our CI system. So versions since 3.4 have been able to satisfy this. This is important maybe to people on hacker news. As for everyone's, for your individual app, does every individual app need these kinds of guarantees? Of linearizability basically being able to, everything you write and being able to read it back immediately, and having those very strong guarantees? Not for every use case. But the functionality is there if you need it, and it goes without saying that the more guarantees you get, these are going to be slower than just reading, local is basically just reading whatever the primary has for you. But know that those exist and when you need those extra guarantees, there is functionality available to obtain them.

And so because these things tend to take longer, I advise not to use socket timeouts. We don't want to leave... using socket timeouts as a way to abandon operations on a remote database is probably a bad idea, whether it's MongoDB or SQL. If you start a long query and the driver just goes away and ignores it. You're letting something continue to run on the database and now it's basically kind of a Zombie process until the database server realizes that no one's ever going to get the result for that. So we really don't want to use socket timeouts to limit our operations. MongoDB provides... all the operations that talk to a database, you can specify a time. And this basically corresponds to CPU time that the server will kill the operation if it goes too far.

And now this is, socket timeouts are exact. If you tell your drivers, say I want to give up after 10 seconds, it's exactly 10 seconds for the, on the client side, whereas maxTimeMS is a bit softer. It's saying that I'm allowing this to run for 10 seconds of processing time on the remote side. So this may end around 11 seconds or 12 seconds. So you don't want to set this exactly to whatever your socket timeout is. But if you have long running operations, this will ensure that you at least also don't leave them, even if the client does give up, you don't leave them running on the remote side. And so this you can apply to

reads and writes, but for write concerns I mentioned when you're waiting for replication, in that case, just to take us back to this, this "Apply" step is the operation succeeding. So this would be your maxTimeMS. But then waiting for the replication to happen is a whole separate thing. That's the operation succeeded, but you're still, just going to wait before it returns to the driver. So that's known as a wTimeout. And that's another thing you can say: if you're using the majority write concern, you might want to... you know, that your writes are going to be quick, but you also don't want to wait forever for replication to happen. And so in this case, you will be able to distinguish: did my write actually succeed? So I inserted successfully, there was no duplicate key error. But at the same token, we timed out waiting for replication. Does that mean that my write didn't succeed at all? No, it succeeded, but you're not guaranteed at that moment that it had replicated. It's probably still going to replicate if nothing, no servers died. So this is, we might, you might call this a soft error instead of a harder error of the data actually not entering. You have a message?

Causal Consistency

A causal consistency, a causal consistency is basically being able to read your own writes if you write something, there's like a linear progression of data. So if you write to the database, being able to read your stuff back of the previous operation. So there's an ordering component to that. So whenever an operation logically depends on the thing before, there's a causal relationship for it. And so the different guarantees there, one is being able to read your own writes, which is, and when you have a standalone system where it's just one database server, you implicitly have this. But when we add distributed nodes and there's replication and things like happening, it gets a lot more complicated to make these guarantees and that's when things get slower. So in a replica set, the idea of causal consistency is not only reading own writes, but they're monotonic. So that means if they get applied in the order that they are executed by the driver, writes follow reads. So if you were to read some data and then write, do write after that, that write is being applied to the same data on the database side that you had previously read in the driver. So just, logically, this is how we expect things to work. So it doesn't take, it's not hard to grasp what this is, but it's something I think of, we take this for granted basically when we're working with standalone databases where there's only one node.

And so this, there's majority read and write concerns. This basically can be satisfied by using those. And durability is the idea that when I write something it will survive: it's durable and that it will survive if we pull the plug on the primary, it's been replicated. So it's durable: can survive an error. And in your application, we can make use of this. The driver provides a session object. We can pass this around to operations and this provides this functionality if you need it. There's plenty of examples for that.

Logical Sessions

So I mentioned sessions. This is a relatively new feature within the last year or so. Sessions are a way... so previously operations in MongoDB were tied to the connection. And so once we lose the connection, the server has no, doesn't remember us at all, that's our, the queries that were running or any writes that we did, they were tied to our connection. Assuming they didn't, if they made it, if they were applied, that's different. But if any state that we had about who was connecting to us was only based on the connection. So sessions allow us - much like php sessions - give us some state beyond just connecting to the database. And so sessions we can be, we can create them explicitly and we can pass them around to operations. Internally, the driver also, anything we talk to the server now, if it's a new enough MongoDB 3.6 or later, we're going to send a session with it and make sure every operation that we execute is tied to some session. That gives us an extra way to, uh, as a system administrator, to monitor operations that are long running. We can, instead of just having to, instead of getting orphan things, we can, the server keeps track of sessions and they can be cleaned up, clean up like Zombie queries and things like that.

Abort, Retry, Fail

And then this plays into our ability to retry things. And so, another quick time check 10 minutes remaining. So this was something that I pulled out of an old version of Doctrine MongoDB. Please don't do this. This is just the retry loop. It takes the closure, how many times you want to retry it and will continually try to call the closure until it doesn't throw an exception. So I assure you Doctrine MongoDB has never, the ODM has never used this for writing. It's used this for opening connections and doing queries. But that's still bad because if you're doing queries, we really don't know if the query is still running on the server.

So this doesn't exist anymore. I think it was deleted. But what's the problem with retrying anything? We get an exception, why don't we just retry it? And this is something you can apply to any database. So we have to be aware that read and write operations, they can change the state of the system. So read can leave the query... I mentioned if you have a socket timeout of like five seconds you start query, that's going to take five minutes to run, right? The driver gives up, it goes on and does other things in php land, but the server, it says: "oh, I'm gonna work on this hard for you for next five minutes because you really need this response". And then by the time it tries to give it to you, it realizes, oh, that bastard went away.

And that's just a query case. In a write case it's kind of more dangerous: instead of just wasting resources on the server, a

write operation, if it's not idempotent, which means we can't safely run it multiple times, we have the risk of, think of a write operation that does an increment. It maybe, it's an innocuous, it's not really a problem, but if you were to just retry the increment operation, you don't know if you're accidentally over counting or under counting. So at best we're wasting time and resources, at worst we're making the data inaccurate. So we want to really think about how we approach this.

So the different kind of errors, first you want to ask before we decide to retry: what are the kinds of errors that we're trying to address? So I think there's three types of errors and one is a transient error. I think of that as: we dropped the network connection or the primary stepped down because it was under maintenance and a secondary stepped up. This is a kind of a self-healing scenario, it will resolve itself if we try it again, probably. Then there's a persistent outage, which is like the whole data center shut down, but you really can't do anything about that. But we really can't, we don't know the difference between these until we at least try maybe once or twice. And if we retry and there's no changes, we can assume that what we thought was a transient error is probably a persistent outage. And lastly, there's a command error. This is: we did something, the database came back to us with a result and said, you're an idiot, you did this wrong. So in that case, if you're trying to insert something and it says your command is malformed, you don't want to retry the command, you're probably gonna get the same response. In fact, I think the definition of insanity would be retrying in that case and expecting a different result.

So those are your three types of errors and what we really want to optimize for - we really can't do anything about this - this is either change the code or actually this is an error for the user. Persistent outage, we probably, maybe someone will get a page and they'll fix it, but our application probably can't do anything about that either. But we can optimize for this: this is really the case, the transient errors, what we want to handle.

So retryable errors is either gonna be a network error or maybe a response from the server that indicates that it's a transient error. So, if the primary was coming down for maintenance, it might be totally offline - we got a network error - or it might be alive for a bit and realizing that, Hey, I'm shutting down, here's a response and I'm telling you that I'm no longer the primary. And so those are two cases. This is usually caused by, there's maintenance or there's a failover happening and we managed to talk to the server before they, before they totally disappeared. But this is probably the most common one: it's just going to be the network error. This is more of a timing coincidence.

And so from a retrying reads, I'd say we want to avoid leaving long-running things running on the server. So queries that return a single doc if you're just querying by id or a single response, you can retry those, right? Those aren't very expensive queries. If you know, it's a short run query, and this is your personal use case and maybe you're just returning one batch of documents or you know, it's gonna just run for maybe a second or two. Those may be safe to retry. And as of MongoDB 4.2, which will be coming out later in the year, the driver is going to try to automatically retry operations, any kind of read operation, even if it's a long running query. And this is because these server will have now functionalities that, you know, the connection was abandoned so I can abort the operation. So this kind of addresses the, we're going to leave some long running operation accidentally running - the queries are going to churn for five minutes while we gave up. So that'll avoid that. What we still can't do is if you're iterating a large batch on an existing query, what we do getMore() is when you iterate on your results. We can't retry that because those are forward-only iteration, so that doesn't work. But the initial find command or the aggregate command, we can retry that.

And now writes are a bit more complicated because not all writes are idempotent. But given a few things, so sessions are cluster-wide, so they, I mentioned that before, uh, they keep track of our state long after our connection dies. Every operation is associated with a session and we can also give it an id, which I'm just inventing now, just trust me, this is, we're just every session we'll just have, the driver will increment some id value just so we can uniquely identify everything. And then we also already have monitoring and server selection I talked about. So we can rely on monitoring. If the primary disappears, we can go back to monitoring and say I need a new primary. So we can rely on safely retrying things, so if you're doing updating a single document, inserting a single document, or doing a batch of those operations, they're all, each one is uniquely identified and we can continually send them to the server and trust it to do the right thing, which means since they're all uniquely identified, if the server gets the write and realizes: "Hey, I never applied this", it's going to do it and return the result. And if it already applied it, it knows that it did so and it's going to return the result that we didn't get the first time, maybe because there was a network error.

So this basically gives us a safe way to do stuff and we like to call this at most once a. So any of the write that we retry, we want them to happen at most once and because at most once that might be zero times if there is a persistent failure and we just can't talk to the server.

And so this really wants, again in php, the try once, doesn't work with try once because we want to actively monitor the server for a loop. So because we want to talk to the server, the default behavior would just give up if we can't find a primary because php wants to fail fast, it's not going to wait a few seconds for an election to happen and for a new primary to step up. So to really make good use of retryable writes, we want to enable, we want to disable the try once behavior in your write. And there's a link here, of a gist where I demonstrate how to do this. And I also tuned the timeout value down from 30 seconds a bit lower. And this is basically our user Atlas cloud service. I keep spamming update statements to it and I initiate a failover and take the primary down and if you could watch the logs of the script, it doesn't generate any errors. When the primary

comes down it certainly waits about eight or nine seconds for the election to happen, but if you want your app totally insulated from errors that, in that case we can avoid exceptions entirely, which is really nice. In Doctrine, in ODM's case, most of all the updates it does, it doesn't do updateMany, so this is a great use case for Doctrine to take advantage of retryable writes.

Okay. 60% of the time... it works every time. Generally it's going to improve things, but again, it's, we can't resolve the other... it's about transient errors, it's not about the other important things that we can't change.

Transactions

So lastly, I'm a little bit over time, I'll try and do this in three minutes. Transactions, this is our actually ACID compliant transactions. And so getting to this point, goes back a number of years and adding features slowly to the database and ultimately sessions, and then 4.0 is the most recent release with transactions on replica sets. And then in the summer it will be transactions on shard clusters. I cannot emphasize the complexity involved with transactions on shard clusters. Replica sets is a bit easier, but I don't, this is above my pay grade. So transactions at a glance, when we do a transaction, you're probably doing a transaction because of writes. So the entire transaction has to go to the same node and because it's a transaction and you're probably doing writes, that means the primary, it doesn't really make sense to do a transaction with just reads. Your read and write concern instead of per-operation, you do that for the entire transaction. So everything has to have the same level of guarantees around it. And that's important to the concept of a transaction because it's either all going to work or it's, none of it's going to work.

And while you can do many operations there are some things you can't do, you can't create collections or drop tables. But all your basic crud operations, your aggregation frameworks and stuff, you can certainly, those are all supported intentionally because that's the common use case.

So in php it looks pretty straightforward and I'm not using custom write concerns or read concerns here to keep it simple. But from our client, we can make a session object and that's our gateway to start and commit things. And you want to make sure you pass your session to all the operations. If I forgot the pass this here, then this insertOne() is not associated with the transaction it's actually just going to run immediately. So that's kind of , there's not really, we don't have like SQL grammar here. So the API relies on you passing around the session to every operation. And similar, if you're doing causal consistency, you want to associate your operations with sessions. So for transactions you're definitely gonna use explicit sessions and pass that option around.

And now the important thing is transactions can be retried. And so, the driver is able, if a commit or an abort command fails because of a socket error, we can retry that, and we'll do that for you once, just to try and insulate you. And if there's a quick error that we can recover from, great. Uh, if you want, you can retry committing more times if you want. If you want to do that in a loop until it succeeds or retry any number of times. We don't bother retrying other operations that's left to you to decide if you want to do that. And the other important thing is transactions and retryable writes are two completely separate things. So retryable writes was implemented first and that's meant to be used - you can throw that into any application and kind of, it just works for a lot of your write activity. Transactions are something you're opting into and the transactions were really the main idea that the main goal that we had to get to, and retryable writes was something quickly we were able to do along the way. Uh, so keep in mind that, they are separate concepts that are mutually exclusive even though they kind of use the same internal machinery.

But the important thing is if the entire transaction fails or if an operation inside the transaction fails, you can restart. You can restart the transaction from the beginning. So if we were doing a transaction and say this second insert failed, I can catch that exception and then try and restart the whole thing. And so the important thing you want to do here is probably have your stuff in a closure or a function call that you can call and wrap with a start and commit. So you can catch an exception and you can retry quite easily, or do a while loop if you really like the procedural style.

So knowing when to retry. Our exceptions in the driver, there's two different labels. Why not? Gruber seems to agree with me. So transaction errors, we need a way to highlight errors from the driver and know with more than just the exception message or code. So having a label, um, there's two labels out of the gate and one of them, this is, if an exception is thrown during a commit, the exception, having this label tells you that you can retry the commit.

And this is a label that you might get on any runtime exception in your code. So those insert ones might throw an exception with this label and that means things have failed and if you want to retry it, start from the top again and start from, start transaction and try and go through the whole process again. There's examples of how to do this. It's very hairy. So something that we're trying to do in the next few months is have a convenience methods so that you just give us a closure and we'll run it and do all the try catching and retrying for you. But if you want to wrap your head around the annoying try-catch nonsense that you have to do, for handling this, you can take a look at the documentation currently has that and we're going to adapt it to a better API.

Alright. So these are some resources. I will share the slides. This was an older presentation I did, just on if you want to know

about retryable writes. There's a lot of good diagrams and stuff in here. This is a lot of relevant documentation you'll want to read up on the subjects we talked about. And this is my coworker's presentation from many years ago before we had retryable writes. So if you're using MongoDB 2.4 for some reason and you want to retry write operations, he has an approach for what you can retry and how to do so responsibly. Thank you. Please leave me some feedback later. I'll see you all tonight for jeopardy and I'll stand outside for questions after.

Thank you for your laptop.

Chapter 14: Leverage the power of Symfony components within ApiPlatform

Tip

SymfonyCon 2018 Presentation by Antoine Bluchet

ApiPlatform is great for building API-first software. Even better, it's build on top of Symfony! Did you hear about Symfony's Workflow component? Or the brand new Messenger component? They offer powerful tools!

For example, a pizza order can take multiple statuses, each one in transition with another (order, pay, wait, eat). This is a workflow.

While waiting for the product, it needs to be cooked! Every time the order is paid, we have to send an instruction to the kitchen. Today, why don't we automatically push that message?

Through a real-life use case, I will demonstrate how well these two fit on top of our Symfony-based API!

Hello! Welcome to my talk about leveraging the power of Symfony within API Platform.

Hey soyuka (Antoine)!

So first thing first I'm going to introduce myself. So I'm Antoine Bluchet also known as soyuka online. And I'm core contributor to API Platform, but also PM2, which is a Node.js Process management software. And I'm developing or coding in many languages, php, Node.js, also a bit of Rust. And I play Rocket League.

What is API Platform?

Um, so which one you guys don't... doesn't know about API Platform here in this room? Doesn't know. Okay. Yeah, that's good, a few hands. So "API Platform is the most advanced API platform in any framework or language". So this is a quote of Fabien Potencier from last year at the SymfonyCon.

So what is actually API Platform? It's basically a set of tools and these tools are given to you, and give you the ability to build an API in really a few minutes.

So for example, yeah, we have create, remove, update, delete resource management. Uh, we have embedded filtering, sorting, we have the validation that comes from the symfony/validator component. We have pagination, authentication, for example, with the JSON Web Tokens or OAuth and things like this. In these tools are also available the security. So when you think about an API, you don't really think about security and you don't have to care about it because we did all the work. We prepared this for you. What's important is that we are using this JSON Linked Data format by default, but you can use any format that you want CSV, XML, everything that the symfony/serializer component gives you. API Platform has Hypermedia capabilities. I'm going back to this in a minute. And we also have GraphQL support, cache abilities with Varnish for example.

Hypermedia capabilities means that we can generate things for you. For example, this Open API, it used to be called SwaggerUI. Um, we have also this, uh, admin panel you can put on API Platform, you mount your API, then you just run a few commands and you have an admin built out of React Admin. We have code generators, for example, for React, also React Native since a few weeks. Uh, Angular interfaces for typescript. And it's production ready. Uh, the, there are lots of companies, I'm sure you're aware of because you're using it, that are using API Platform in production. And it does really work well.

For this to work, we have a Docker, a docker container that's all ready for you. You just have to clone the repository launch it and you're ready for production.

So API Platform is also open source. So this is really, really great. Today, so I've combined, we have two repositories, we have the core one and API platform with, which is a bit front page. I combined them both and we have about 2,900 commits, 140 contributors. So thank you very much for this. Uh, about 3000 stars and we just reached a million downloads a few weeks ago. So I was checking like two days ago on the Packagist website and yeah, we just reached a million downloads on the core repository. So this is a really great milestone.

How API Platform Benefits from the Symfony Ecosystem

And now about this talk, it's the thing is with API Platform is that it benefits from the Symfony ecosystem. So what does this mean? Actually? It means that because it's built in a Symfony way, you can use every component you want within API platform and really they combine with each other really, really in a simple manner.

So basically now for those who didn't know about API platform before, I hope that's what you're thinking right now and yeah, I just advise you to check API Platform and just try to use it. If you don't like it, it's okay. But really I'm sure you will.

Building our Pizza Ordering App

So now let's order pizza because we all like it. To order pizza usually you take your phone, I don't know, you give a call, whatever, you order. Then the pizza gets prepared and then it gets delivered. So this is what I call a workflow. And then about this workflow is when the user just sent out the command to call, to order for a pizza, we need to notify the kitchen and say: "hey, prepare my pizza please". The kitchen prepares the pizza and then it sends a message back. okay, my pizza is ready, just now we need to deliver it. Now there is a second message that goes through the delivery truck, for example, and says, okay, my pizza needs to get to my client.

So, um, to do this, we're going to base our project on symfony/skeleton. So this is what I did here. So I don't know if you heard about Symfony4, and I insist on the 4 because we have now Flex recipes. What are Flex recipes? These are kind of like Composer plugins that the third line here, the composer require api is in fact, um, launching a Flex recipe that will install API Platform, but also the doctrine-bundle, CORS support - Cross Origin Resource Management - and everything you need to basically have an API started. So these three steps, and I'm done, I've created an API. All I need to do is add some stuff. So really you could have started the API Platform with our skeleton. So check it out on Github it's the api-platform/api-platform project. But I prefer to use the symfony/skeleton just to show you guys how things go together.

So now that we have API, I'm going to start by installing the Workflow component. So I don't know if you guys assisted to this, the talk tomorrow, this morning about Workflow. So yeah, basically this is a quote from the docs. And... so, yea: just require it on top of our current API. So really nothing really new here. Um, next thing we're going to use the Messenger component. He also, we had a beautiful talk this morning by Sam about the Messenger component. And I actually stole a slide from him, just like this afternoon. And yeah, we are going to use really basic functionalities because all we need to do is send a message to the kitchen and to the delivery truck. So we have a message, we have the bus and the bus will manage the, the to send the message and then through our transport, handlers will get this message. I will get back to this.

Yea, easy: composer require symfony/messenger. I remind you, I'm in a symfony/skeleton project. I installed api, workflow and messenger. So now, um, if I want to order pizza, I thought about it and I come, came out with this graph here. So the first step on the top left is just create an order. To do so we're just doing a POST on our API. Then the second step, when there is this small message Emoji, it's actually a message that we are sending from our API to the kitchen in the bottom. When the pizza is done, the kitchen will answer: "okay, API, I'm done, I'm ready". The pizza is ready: PATCH /orders/prepare. The fourth step is, then again, I'm sending a message to the delivery truck. And when the order has been delivered, the delivery truck will answer.

So let's do this. Um, just to remind, yea, this is the stack. So Symfony, API Platform. I'm using RabbitMQ for the transport of my queues. Um, and SQLite because I'm a lazy guy, I didn't want to set up a PostgreSQL database.

Creating the Order Entity & ApiResource

So to do the first step, it's the easiest one. All we have to do is create an entity. So in src/Entity/Order.php. basically most of you guys I hope know about the annotation there. So we have basic Doctrine annotations and the most important one on the top of the screen here is the @ApiResource. In my Order, it's really simple: for now I just have a status. It was, yeah, you would have many more fields like what pizza we want, and what the are products, etc.

And yeah, it's done. I have an API: I have my four routes. So API Platform automatically creates these when I create my entity. I have GET on the collection, then I can create a new one, gets an item that exists, update it or delete it.

Workflow and Messenger

So far so good. Now, the workflow. So to do the workflow, I hope you've seen the conference this morning, it was really interesting. I want, I will not get into details, but basically all you have to do is, to set up this configuration. What is really great is when we did the composer require symfony/workflow, we have a recipe behind it and it preconfigured everything: this file existed. I just had to go in there and add my things. So I've added a few places and a few transitions. Uh, the cancel one isn't really important, but yeah, just to show you guys how easy this was.

And yeah, that's nice with the workflow as, I don't remember the speaker name from this morning, but he showed this

command as well. And he said something really interesting is that: when you do this, you can actually see what you previous configuration gives. And I did this many, many times and I'm, I actually was okay: I will rename things because transitions were, had bad names, etc. And this is the result of the previous command. So we can see we have a first state is the order being ordered, then the preparation, order get... is prepare. We deliver it and it's been delivered.

Now let's talk about the messages. So to do this: same thing, all we have to do is just prepare this configuration file. On top of it you can see that I used, yeah, I put out the environment variable that I'm using below and I'm specifying an amqp queue in RabbitMQ. We, the small thing I added there is the serializer configuration path, where I said: okay, I'm in an API, I actually want you to serialize things with JSON-LD. The rest is just like in the documentation. And to picture it, well I have two transports: one is my kitchen. So you really need to picture this transport as different physically places. The kitchen is really a kitchen and the delivery truck is a delivery truck and I chose to route my two messages - the preparation message and the deliver message - to these two transports.

Now that we have this, we can create our command but we still need to send this message to the kitchen and say: Hey, I have an order now, please prepare my pizza". To do this, I've chosen to add a Doctrine listener. So I added the annotation there: @EntityListeners OrderCreateListener. So every time an order gets created with a POST, on POST on my API, I'm just able to do things. And here, I'm just dispatching the message: PrepareOrderMessage. So I injected the MessageBusInterface and on postPersist I am saying dispatch my PrepareOrderMessage.

So, this PrepareOrderMessage is just a wrapper around my Order. So, um, uh, this morning, Samuel told us about the Messenger component and new things that's evolved since the 4.1 version of Symfony. And so they might be, it may be a bit simplified here. And next step: we have this Order handler. So this is actually my kitchen. The kitchen will get messages in this __invoke() function, it will prepare the pizza, and when it's done, it will just send an order via the API route with a PATCH.

Let's do this actually. We don't have this PATCH /orders/prepare route. We need to create it. So with API platform it's really easy to add a new routes. So if you remember, at first I had four one's: one to get the collection, items, to delete and to update. So I'm specifying itemOperations with my previous routes and I'm giving a new one called status where I say: okay, this has a PATCH method. And in the path I have my order, the id of this order, and the transition I want to apply. And the last one, the most Important one, the controller class.

And this is how my controller is. Um, it's a Symfony controller. I hope you all know how it looks like. I've injected my workflow Registry and whenever I get a PATCH on this new route, I'm saying: okay, get me the workflow please. And just apply the transition I got in my arguments. Also I added some noise there with the HttpException.

So we can now patch our orders. The kitchen can say: okay, I have done, just do the prepare transition because you're done.

Now we are going to do the exact same thing with the delivery. And, sorry, what, what's it... what is really good here is that we used a workflow to picture our process. So what comes with the workflow is actually a lot of events that again, we use behind it. And this is exactly what I'm going to do now. I'm going to say, okay, whenever the pizza has been prepared, please send the message for the delivery truck and say: okay, the pizza is done. Now you have to deliver. So this is a listener, or can't be more simpler. And at the bottom you can see that the subscribe events I say: workflow.order.transition.prepare call onPrepare. And the onPrepare will just get back to order from the event and we will use this bus to dispatch this new message: please deliver my command.

So then again, exact same thing. This is my delivery truck, this is where I'm going to deliver my pizza. Drive a bit, get to the client and say, okay, here's your pizza. And when it's done, I can get back to my API and say, okay, I'm done. I'm done delivering.

Live Demo

This was step five. So now I'm going to show you guys how this all works. If I can find my screen. Perfect. Nice. So, as I said before, I actually have an AMQP, a RabbitMQ service. So here you can see on the right, bottom right screen, I have a RabbitMQ service that's actually running right now. And on the top left of the screen I'm going to, uh, I want you to picture a kitchen. So the top left of the screen is the kitchen and I'm going to say: okay, consume message. And yes, please consume the message of my kitchen transport here. So previously I routed the message, PrepareOrderMessage, and I said: okay, this message has to go to the kitchen. So this is my kitchen. And on the right side of the screen I'm going to start my delivery truck. Now, I'm just going to create an order. So if you remind, I had only one field, it was status order. I'm hope you can see alright. I can maybe increase the size a bit.

Yeah. So I'm using this tool httpie: it's just curl: http POST on my API with new status order. So status order is the first step of my workflow. So let's go and you can see the others done it. It got created the kitchen prepared it and now the delivery truck is delivering it and he said, okay, I delivered. So, I'm going to start this again with a few commands so that you can see everything in action. So I've added, in my handlers, I've set up a sleep time out with dots. And when it's ready, I just patched

the order on my API.

Okay, so that just raises further questions indeed. Because here, I really wanted to show you guys how easily things come together when we use Symfony and API Platform. The thing is, um, we could have used the messenger and a lot more configuration possible. And it goes the same with the workflow: it can be really much more complicated than this. What I actually don't really like about what I did here is to use transitions through the API. But yeah, this kind of things can surely be improved. Um, also about the transport, I used here RabbitMQ on the messenger transport. Uh, I've actually worked on the, on a new transport with redis. And uh, I've opened up a pull request on Symfony. So yeah, lots of thing that can be improved in this workflow.

What's Next?

Um, actually what comes next now is that within API platform also, we will have the Mercure support. So I don't know if you guys know about Mercure yet, but it's something Kévin Dunglas worked on that, that gives you the ability to receive live events when you update a resource. So for example, here, when I updated my pizza, I could have received a live event in the kitchen just by adding one simple configuration on my annotation, meaning mercure true. In API Platform we also have, it's, it's being prepared, it's the MongoDB support. And we also have people working on Elasticsearch read support within API Platform. So lots and lots of new stuff. We work everyday on API Platform to provide you with the best tools possible.

So yeah, this was it. So I want to thank the SymfonyCon for having me here. Uh, obviously, uh, everyone who contributed to API Platform because I really learned a lot by just reading you guys and working with you. Uh, also, Les-Tilleuls, which will be hiring me in February. And yeah, this talk will be available on GitHub soon, uh, at this URL. And you can follow me on GitHub and Twitter. So now if you have any question I will be happy to answer them.

Questions

Does someone have a question? I think this works.

Okay, I think it works. I was wondering more about how can you customize the API Platform because it seems like it's like this, but perhaps not.

How you can customize API platform?

I mean, is this behavior that you showed us, is it the standard behavior? Can you do other things?

It's not a standard behavior because I used other components that comes with Symfony. But really when you have Symfony 4, with the autoconfiguration and everything, it's so easy: you just add the good interfaces and everything works. So, so I don't know.

Okay.

Someone else back there? Just send it and then. It's like non-breakable mic.

Okay. Uh, for me it's questionable for the monolith application, uh, to use some additional services like RabbitMQ and Redis for the messenger. So I'm wondering if the same workflow which you just showed can be achieved just in the monolith with use of just Symfony events.

So you mean working with this but with Redis on, in back?

No, without Redis and without RabbitMQ: just with internal Symfony event system, just in a monolith system, we are working in a once call, right? So no additional services required usually in this case.

So actually, um, when you transport the messages, you can use whatever you like. I mean, yes, it can work, with a bit of work, you'd have to dig a bit through the components, but yeah, I don't see why it couldn't, could not work.

Someone else. Nope. Looks... thank you for your attention.

Chapter 15: The Patterns Behind Doctrine

Tip

SymfonyCon 2018 Presentation by Denis Brumann

How does Doctrine talk to your database? What are Data Mapper, Unit Of Work, and Identity Map? These are the questions I want to answer in this talk. We will look at how Doctrine ORM implements them and what they are there for. Finally we will look at how they compare to Active Record and what the benefits and drawbacks are to help you choose which one fits your needs best.

A lot of things, um, I hope it's not too fast. But in any case I will still be around afterwards, so if you want to talk about any of those topics I will be talking about, we can talk about that later on in more detail. So first of all, because I already know that in this room it's really hard to see the slides if you're in the back of the room. The qr code goes to joined.in to the talk. I already uploaded the slides with some additional pages in there that you just have to skip over. And yeah, you can just go to the talk/c1b8c is the talk id and, and you can just get the slides and you watch them on speaker deck as well if you don't want to follow on the screen or things are hard to read.

Hey Denis!

So, um, with that out of the way. Um, hello again, my name is Denis Brumann. I work for SensioLabs in Germany as a software developer in Berlin. And you can reach me via email denis.brumann [at] sensiolabs.de and also on twitter and GitHub with my initial and last name.

Disclaimer: Doctrine 2 Vs Doctrine 3

And before I start I want to make clear that everything I will talk about is about Doctrine 2 not Doctrine 3. It's not out yet, but it's already under active development. The master branch already points to Doctrine 3. Things might still apply in some way or another, but, but it will probably not be the same. For example, the unit of work is probably not stick around. Um, so keep in mind this is how things are right now, but it's still helpful to know because if you want to do this switch to Doctrine 3, you probably want to know how Doctrine 2 works under the hood to basically know does this change affect me that they introduced. And do I have to look at anything that maybe could cause a bug in my code.

Introducing The Domain: Order & OrderItem

So we will start by just looking at a simplified domain that we basically we'll follow through the, through the whole talk. And I only have two entities in the domain. So it's kind of easy. It's probably easier than all the entities setups that you have in your projects, but it will be enough to to look at all these things. So first of all, we have an Order that just has an id. It's an auto-increment id, very basic stuff that you will probably all know. We have a timestamp from when that order was created and then it's basically just a list of OtherItems which we'll look at later. And everything is set via getters and setters just like most people are familiar with. There are other nicer ways of dealing with things, with like rich domain models, but where it's more expressive, but we will just look at something that's kind of ordinary.

And the OrderItem is the same. It has the same auto-incrementing id. It's linked back to the Order. Um, that's a necessity because OneToMany requires ManyToOne. We will look at that. It has a name for, for the item that we want to store in our Order. And it also has a price which is a string, which might seem weird at first. But if you ever worked in an e-commerce system using floats, it's really a pain. It just causes rounding errors and, and, and it's just horrible to work with. So one workaround is to use ints, so integers. So basically instead of using the euro amount, use the cent amount: 100 cents are one euro and that's it. And that makes things a little bit safer. And I pushed it to the next level and used string and you will see later on why we do this.

So let's look at the code. So you probably all have seen this. It's a very basic entity. We have the usual annotations. In this case we also add a custom repository class. You probably have seen that as well. We give our table a specific name. For the OrderItem it doesn't matter as much. For the Order, it's more important because order is a reserved word in most dbms. So that kind of messes up things. But when you work with Doctrine, um, so it's nice to, to give it an alternative name. As I already said, the auto-increment integer, that's pretty much what you all know. Then we link to our Order. Um, the interesting thing is that we basically tell the JoinColumn that this OrderItem always needs to be attached to an Order because it doesn't really make much sense to have an OrderItem just loosely hanging around in the database. So we just ensure this with the

JoinColumn(nullable=false). And the rest is just basic stuff. We get things, we set things. You probably all have seen this, you can auto-generate it. So it's really simple.

With Order it's kind of the same thing. We have an auto-incrementing id, we have the relationship this time a @OneToMany that maps back to the OrderItem $order property. And if you think about the, the table layout, it kind of gets clear why we needed the other relationship from the OrderItem back to the Order because obviously the, the order ID is stored inside the OrderItem. It links back to that. So if we were just to have this relationship with, on our Order, that that wouldn't be a reference to: okay, where is information stored? It's in this different table. So the, the OneToMany relationship is the only one where you actually have to have a bi-directional association. In other cases you can just have a unidirectional association.

And the other important thing here is the cascade={"persist"}, which come, will come in later, but just keep it in the back of your head that this is important. And yeah, for the timestamp I just use the DateTimeImmutable object. It's one of the types that Doctrine supports. We initialize the items that we want to put into our Order is an ArrayCollection. You probably have seen it 100 times and more. Um, in my case, I return the array instead of the ArrayCollection. This is just something I like, but there's no specific reason to do that. Um, and it doesn't really come up later, so I just want to point it out that this is a quirk I have, you don't have to do that. Um, what is important is the addItem(), we basically, because we have a bidirectional association, we have to make sure that both the Order and the OrderItem know what object they are linked to. So in this case, I set the Order um, to $this on the $item and also add this item to our current Order if that makes sense.

Um, and yeah, this is where the string part for the price comes in. I use the BC math extension for PHP. If you use modern php where you have strict-typing and everything, you can't just pass in an integer or float, it has to be a string, otherwise you will get a type error and that's why I use the string all the way through and it will just add those strings, strings up from the items and then you get the final total price. So it's really simple math. It's just doing plus current total plus the next item.

INSERT

Okay. So those are the basics. Um, I know that was kind of fast, but I, I kind of hope that you got the structure in your head, you know, what we're talking about in terms of what kind of objects we're dealing with because this is all we will use throughout.

And yeah, just assume that with our very simplified e-commerce system that we have now, we want to do a checkout. We select items from a page - I don't have everything built for this case - it's just, let's assume it happened. And then we pass things to a Checkout service. I don't use the service suffix here, but this is just a basic service class. Um, I pass in the EntityManager because we want to store the Order when we do a checkout with all the OrderItems. And this is what the actual checkout method looks like. In my case I just get some $cartItems which are basically products that, uh, in a DTO entity or whatever that is used on the frontend for displaying my product items in the cart and checkout. And now I want to transform them to, to an OrderItem.

It's basically to make sure that when the product changes, I change the price, my OrderItem is supposed to still have that price because I will still want to charge the customer the original price they used when they used the checkout. And then I just do some mapping. I map the name and the price from the $cartItem to the OrderItem. I add that OrderItem to my existing Order that I created up there in the top, and then I just save that Order and I flush the entity manager. So you probably have all seen this.

This is really basic stuff and still, there are some really interesting things going on behind the scenes. Because if you think about it, first of all, we are only persisting the Order but the OrderItems will be saved as well. So Doctrine, for some reason knows it has to save those items into that separate table. Um, and it also knows that has to do insert instead of just an update. So let's look at why this happens? How does Doctrine know, which items to save?

Unit of Work

And that is really interesting and it works based on the unit of work. This is a really huge object and, and it's kind of a basic pattern. So, um, the, the example on the right is basically what Martin Fowler has in his pattern catalog. I omitted a few methods that we don't really care about, but basically this is an object that stores our changes for the database and, and keeps them in memory until we flush and we actually want to write them. And then this unit of work, will figure out what writes it has to do: if it's an insert, if it's an update and which order things have to happen. And that's really a lot of stuff going on there and it's a huge class right now, which is probably why in Doctrine 3 that they want to split it up a little bit. Um, and it's also kind of ugly to look into it. Um, so it's fun to dive into it if you want to do it. If you have to do it because things go wrong, then it's just a pain.

So I want to save you from doing that and just enjoy looking into it for the fun of it. Um, and you can already see that there are some states for, for insertions, for updates, for deletions. We have the entityChangeSets, so what changes need to be written to the database. And more importantly we see some of the methods that look a lot of similar to the methods on the EntityManager. Um, so for example, we have a persist() method that looks basically the same. The remove() method, the

flush() method that kind of looks like the commit() method. So that, there's some kind of overlap. And that's no, no that's intentional because basically the EntityManager will pass its called to the UnitOfWork and the UnitOfWork will store all those things.

So when we pass in the Order, the UnitOfWork will figure out: ah those other properties that map to a database table, I have to basically save the data that's in there for later change. And I will also have to remember that this object that they wanted to persist, I have to basically write that to the database as a new object based on the state.

And the reason why this thing works with the OtherItems as well as because we have this cascade persist in there. Because when we persist our OrderItem, what will always happen is that the EntityManager will also look through the associations on our entity. So on our Order it will look at the the items property and say, "Oh this is an association, the items in there I probably have to save as well". And then just loops over them and sees either if we persist them manually or if we have the cascade persist, then obviously Doctrine knows everything that I put in there I have to store in the database as well on an insert.

And yeah, this is basically what happens. So we pass in the Order and then Doctrine figures out, the OrderItems in there, I have INSERT them as well and have to write to the changesets. And this is roughly equivalent to adding a second persist in our foreach loop for each item. There are some subtle differences, but basically it works out the same. And if you don't do either of those things, so neither the cascade annotation or persisting all the objects manually, what you will get this error. Because, obviously, Doctrine knows that some objects should be stored in the database because we have the association there, but it doesn't want to write this unless we tell Doctrine to do so. Because it would be kind of bad that when Doctrine thinks:

> yeah, I know Dennis is kinda lazy and he probably forgot to do the annotation. I will just save it for him.

And that could be fine, but if it just writes stuff to the database that I really didn't want to have, then yeah, I'm in trouble. So Doctrine says no, I won't touch that. Figure it out yourself. Do you really want to persist this? Then either call persist or cascade or just move it out of the collection. So this is something you have to be aware of when you work with Doctrine that unintentionally non-persistent entities that come up in the changeset, Doctrine will not touch them. You have to figure things out. It will give you a nice error like you saw before where it tells you what to do, but still you have to check if: is this really something that has to go into the database? Probably yes, and then you have to act on it and do that. And also you have to add this cascade annotation to make sure that this happens automatically for you. And obviously only on the entity that is supposed to cascade. So on the OrderItem we didn't cascade for the Order, because I decided that I always want to save the Order with the items but not an item and then create a new Order for that item. Because in my e-commerce context, that doesn't make sense. If you have this context where this makes sense and your item can be the primary object, basically, then you can just do that: just add the cascade and it will also save the new Order through the item as well when you persist one item. And Doctrine will also figure out in what order to do that.

FIND

Okay. So much INSERT, there was already like really, really lots of stuff in there. And it's basically getting worse, but at least we have nice pictures. So let's think about like we're inserting new orders, we do our checkouts and, and we now want to have a backend where we want to look at our orders. And we want to have some additional details. So in our order list, we don't just want to show the order number, we want to see how many items are in there, what will be the total price for that. And what you can see on the bottom is that this will issue 2 database queries which might seem odd to some. Um, because yeah, it's just, we just wanted our order list and that could be one query.

And the reason this happens is you don't have to look at the full Twig template, but this is basically the bottom part that that shows how many items are in there and what they cost - the total price. And this is our Order, method, getTotal() price that iterates over the items. So I'm not calling the items themselves, the, the, the method does, but still Doctrine recognizes: ah, those items I need to get somewhere. And it will just issue the second query for those items.

And this can get bad if you have more than one item in your overview or one order in your overview, because for every Order it has to fetch the items. So if we have five orders, it issues one query for getting all the orders and then another query for every order to get its items. This is often referred to as the N+1 problem. So Doctrine will basically issue more queries than you might think it will. And yeah, this is basically what it looks like in a profiler. So you can see that on the bottom you have the query for the order list, fetching all the orders and then for every order of 5 times you get the same SELECT statement, um, for the items fetching all the items for that order.

Lazy Loading

And the reason this happens is lazy loading. You probably have heard that before. And how it works in Doctrine is it uses a so-called proxy pattern. So when you call getOrderItems() or even just accessing the items property on your Order, Doctrine has not loaded this beforehand, because we only wanted the orders in the beginning and our controller. Instead it will see

that those items are not loaded, load from the database. And once that is done and will return those other items and it will keep them in there. So it will not do the same query on the same object over and over again. It just remembers: ah these items were not fetched the first time and that's why I'm fetching it now. And then subsequent calls will get those items we previously fetched.

And what's kind of responsible is the, I already mentioned the proxy, the proxy objects that Doctrine generates - at least if you're not in debug mode, so, um, it might create the folder but you will probably not see files on there, so just do debug false and you will see those proxy files and we can look at them. They are huge and ugly and, and I never had to look at them in any real cases. It's just basically to show you what they look like. Um, you can already see that it's just an object over our original Order. That's also why we can't have, entities final, because something has to extend them - the proxy object. And then have an initializer and then you have tons and tons of code that basically is unreadable and, and you don't have to, it's automatically generated and you don't have to care about it. But you can already see that if I want to get those items, some initializer is looked at it, and it will invoke a method. And this initializer is basically responsible for doing the querying in the background. And then it will just get the items as our method on our parent object basically described. So, um, this is the magic behind it, so to speak.

And this is how it looks as UML. I hope I got the arrows right the direction and the arrow tip. I, I'm, I'm not a computer scientist, so I have never learned this and I never get this right. Um, but yeah, this is basically what it looks like. You have a wrapper on your Order object on the right that has the same methods, it just has some additional stuff on there that you don't really have to care about. You don't have to access yourself. Doctrine does this under the hood for you.

Fetch Modes

And this is fine if it's actually what you want. So for example, if you fetch orders and you don't want to access the items, it's really good that doctrine doesn't fetch the items every time you, you get this order. But when you want to have the items and it issues these additional queries, then you kind of want to address this in some cases where it makes the site slow.

And what you, what you can do is add another option to the annotation and say: this relationship I want to fetch eagerly, so always fetch those items. Um, obviously this has the same drawback as fetching things lazily does because now everything is always fetched. And as I said, if we have cases where we just want to work with the Order without items, we now do a query that we don't really need. So this might not work as well, but, but it's a nice and easy way to address this issue to, to not lazy-load.

The other kind of more involved approach is since we have our own repositories and we can write our own DQL queries, we can just tell Doctrine to join the items in certain cases where we define the method and we just call this method. So in this case, findOrderWithItems(). And the interesting bit is the join part. You can see that this is DQ, it works on the object level. So we're not talking to the database table, we're talking about our objects. In this case, the Order object that has alias o, it has some items property - we already know that - and this item's property, everything that's in there, in the association, that's what we want to join. And that's also what we want to select because if you don't select then it will just join it and throw it away when hydrating the object. That's basically if you want to ask something from the items without actually fetching the items, then this can be useful.

And so this is a more involved query because you actually have to write the query other than just doing find(), but, but it can already help you that I can now figure out my cases where I want to do Order with items and Order without items.

Custom Object Hydration

Another approach that I personally like even better is custom object hydration. Um, so take for example, a DTO object that looks like this. It's probably really hard to read. So we have an OrderSummary and it describes all the information we want to see in our Order list. We have the Order id that we want to display, the created timestamp and then the item count and the item total. Um, and then we have just getters, we don't want to change this data because this is basically a read model on our database and we just want to fill this model with the data. And we can still use the DQL query. Let's go through it, like not in chronological order. So first we see that we fetch all the stuff that we need for our OrderSummary and the interesting part here is that we count the items in the database and also we use the SUM method in the database. So we don't have to run the PHP query stuff. We can actually do this in the database which is really fast for doing the kind of integer operations. Then we still fetch the data as if it were the Order object. So we say every data that you would get from the Order object, just just get it. And then we extract the data we want. And then we just create a new object with our DTO in there. And basically it will map the data not to the Order object but to our DTO. And this will also perform only one query and it has all the data in there and we don't even need all the item information that we don't need and we just need the stuff, we get the stuff that we need. And this is what it would look in plain SQL. So probably a little bit simpler, but yeah, the, you just have to translate that back to DQL. Or you can even use sql and use the ResultSetMapping to map it back to the object instead.

And so, just to remind you, this is the queries that we had before. And this is what we get now: we have one query you, can see in the bottom, it kind of looks like the SQL query we just saw, Doctrine just uses different identifiers. And it's also a lot

faster, like 3 ms, roughly 3 ms. And 3 ms doesn't sound like a lot, but obviously it adds up. If you have more than six items, like 1,000 orders.

And yeah. So basically your takeaway for this is: Doctrine loads, associated entities lazily by default, which can be good, but it can also be a problem for querying. So you can do custom queries with a join and then select those items yourself. You, you can do a custom hydrations, either into a DTO object or if you just need a scalar value, like a count, you can just get this as well and you don't even have to wrap it into an object. And this can improve read performance tremendously for you.

Identity Map

And something that kind of relates to that, to optimizing querying, is identity map. This is also a pattern that that's wrapped inside the UnitOfWork. So whenever you call the EntityManager find() method with the object class name like Order::class and then the id you want to fetch. What will happen is, the first thing that the EntityManager does is look into the UnitOfWork, into the identity map to see if someone, well actually, if we performed this query earlier before and we already have the object in this identity map. And if we do, then it will just get the object from the identity map and don't even issue a query. If not, then obviously it will just do the database query as before the, the typical find() method and it's fine.

So, um, just to, to basically see how this works. This is very simplified code with just some, some dump in there to see the results. So we do the same find() twice and in between we look at the UnitOfWork and at the end we looked at the $order and the $sameOrder, um, are actually the same object.

And yeah, this is what it looks like. So the identity map contains the class name and then an array where the ID is the key and the object itself is the value. And this is how Doctrine looks it up. Like, okay, I have this class, I have this id, the user wants that, so I have to pass this object on. And on the bottom you see that yea, both objects are actually the same identical object. So every change you do to the object is basically kept and the when you do find again, then you get the object that you already had in memory, it doesn't issue an additional query. Um, and you can see that, on the bottom as well, there was only one database query in this case. And this is basically for our Order object with the find items, it will, at the same time also fetch those items as well and put them into the identity map as well. So if you want to fetch an OrderItem later on by its id, then it will also not issue a query because it already got this item through the Order with the OrderItems. And obviously the downside is that it keeps a lot of data in the memory. But also it's probably doing one request, you won't have like dozens and dozens of entities that it has to keep in the identity map and, and has to work with.

And yeah, so basically that's another important point when you want to improve your read performance, you have to look out that when you use the repository's find() method, it uses the criteria array thing, which is not the same as the EntityManager find() method. So it will actually perform the query twice. It will still fetch the data from the identity map, but it does the query because it can't figure out that, even if you just fetch the id, the find method or your custom query, the EntityManager, the UnitOfWork, don't know how to introspect it and figure it out. But if you're using EntityManager::find(), then you can actually have fewer queries than you would expect.

UPDATE

Okay. So we looked at INSERT, we looked at the find() method. Let's also look at updates. Um, so let's assume we have an object, we create it new. And then we set the ID. Some people might assume that since my object now has an ID, I can just update this object if it's already in the database. And the thing is, obviously, this will not work, because the UnitOfWork keeps track of an entity's state. And since it didn't know the object before, it, it doesn't know what to do. It assumes it's a new object. And so it wants to insert that new object. Because internally, Doctrine keeps track of entities not by the ID but by the object hash. And so Doctrine has a set of states that it kind of looks at and, and tries to sort entities into. Obviously, for new entities, the new state, the managed state, when UnitOfWork knows of this object, it's already managed somehow, that that's both new objects, updated objects and deleted objects and detached objects. So objects we don't, no longer want to maintain via the UnitOfWork and via the EntityManager. So all the changes we do to that object, they will not be persisted, they will not be saved. And removed objects, or something that's basically said that: once you do flush, we want to delete this.

This is how it looks on the code. Um, there are some nice explanations in there. And so basically what happens is, since we created a new object, not via the UnitOfWork and the EntityManager, but by ourselves, the UnitOfWork doesn't know what kind of object is this, and it says: ah, this must be new, I will give it the new state and I will insert it. And on the other hand, if we fetch an object via the EntityManager, the UnitOfWork now knows that this entity was fetched before, it's already part of the UnitOfWork inside the identity map for example. And that's why, when we do this, we no longer need to persist obviously, because it's already in there. Now it knows that I can update this object: this is not something new, this is something I have to update. And so basically, for you, what you have to look out for is, just, kind of, keep in mind that the UnitOfWork manages the state for you. And you don't, you have to make sure that everything you want to insert is part of the inserts sets and so on. So yeah, it's, it's just, it's a lot. I know last session, but we're almost done.

Active Record

So finally, which is kind of unrelated to everything before but, but I promise it in the abstract, so I figured I have to deliver it, is a kind of comparison to, to active record.

So I don't know how many of you know that Doctrine uses the so-called "data mapper" pattern to map things. And another famous approach as active record. Actually Doctrine 1 used it, Propel was another famous object relational mapper, that uses active record pattern. In Ruby on Rails you have the, I don't know what it's called. And obviously Laravel is famously using Eloquent, which is also using the active record pattern. And what the active record pattern does is it wraps an object in a way that everything related to the database is part of this object. So usually what happens is you extend the base model, base record or whatever, and in this base record you have all the database-related stuff. You can query things, you can save things and then you create your new entity, with some restrictions as to how the entity is created. And and then you can do all the database operations on there. So basically the models allow for, for querying the data and you don't need special classes like the EntityManager or repository or anything else, you just do everything for the record.

So this is how it looked in doctrine 1: the abstract Doctrine_Record class. This would be what you would extend. And you can already see that there are some states in there, so it kind of still does all the same things that Doctrine 2 does with the keeping the entity states, for example. So there's this kind of the same feature set, but everything is part of this basic model. And sure there are things that work differently but there's even a proxy thing, so it kind of looks the same. So just to give you kind of look at how you would act on an active record kind of model. And I didn't put the, the entities on there as a record, because based on the library they will look differently anyway.

But roughly, this is what would change. So first of all, since we don't have the EntityManager anymore, all operations are on our model, we just call an $order->save() and this bubbles down to the record and this is where everything is implemented, it uses a connection. Same with an update. Basically instead of getting the EntityManager to find our object, we use a static method called for find() for example or query to get a new Order object. And obviously the flush() has to go as well. We still do the $order->save(). and so we don't need any additional classes, which might seem nice. We actually saw that the code kind of gets a little bit more compact, so that's less stuff to do.

And just just to keep kind of a record of what are the benefits and drawbacks of it. Um, because it, it really depends on what your use cases. Um, so if you look at the data mapper, um, what's really neat is that basically you don't have to care about all these additional classes doing all these things. Everything is inside my model and I don't have to to remember to inject anything. I can just work with my object and do everything on it that I want to do, like saving stuff which can be nice. It's, I don't have to use injection in the service to get the EntityManager, I'll just pass the object and say save. And that's it.

And obviously this comes with a downside that, that my object has to fit some conventions that this record gives me, to, to basically be able to figure out what stuff to write and what not. And with Doctrine you do it very implicit, ah explicitly. You put the annotations on there, you have the yaml file. But with an active record, you basically have to figure out what do they want me to do? Where do I have to store my properties, how do I give it the table name? Um, maybe there are some limitations, like usually you can't use the constructor because this initializes some, some basic database stuff which can be annoying, when you want to do like, rich domain models for example, and you want to have this constructor, you have to remember to, to call parent construct. And yeah, the, the obvious benefit is that it's less code to write. It's just easy: save, no EntityManager, no custom repositories, you can just go ahead and do things.

Summary

And just to wrap things up. So what did we look at? First of all, um, the, the Doctrine EntityManager uses underneath the UnitOfWork which does all the heavy lifting and storing the entity changes until they are persisted. It takes care of the commit order and everything. The proxies for, for finding stuff are useful that it basically allows us to lazily load stuff if we want to, and if not then we can just bypass it by writing custom queries. And the identity map as well for finding stuff is really useful when it comes to minimizing queries and to make sure that we, when we fetch an object from the database and it's something that we worked on before that, that it still has the same state: we work on a consistent object and not like one OrderItem of the name x and one OrderItem of the name y, and then we want to save it, they don't really fit together or the last one wins.

And yeah, that's basically it. So I don't know how much time I have left. I think I rushed through it quite a lot more than I would have to. Um, so yeah, plenty of times for questions I guess.

Questions

Yeah. Wait, there's a microphone. Cool, thanks, thanks for catching that, so I don't look bad.

(unaudible) Doctrine to handle concurrent persistent in different processes, the same database?

Um, so, so again, uh, how does Doctrine handle different repositories?

No, if Doctrine is running into 2 processes, php processes?

In different PHP processes. Yeah. So, um, when you have different PHP processes, you will obviously have different unit of works. So there, things will, will not apply, in terms of, you will not have the identity map from one process work with the other process. Um, I'm not sure if there's some common caching you could use, like the l2 caching things like that. I haven't looked into that. Um, but yea, so those things are really, if you work on one php process and, and you just want to access your data there, which is kind of the most common use case. I can't really tell you how it works with multiple PHP process. So, um, everything is good. Um, I think there was someone in the back who had a question or not.

Okay. Uh, since we all have some time, I don't know how eagerly you want to go to, to the jeopardy, I kinda do. But, um, I, I have some code if you kind of feel that this, this was like rushing through everything and just getting a glimpse here and there. If you want, I can give you a quick rundown of the code for, for example, the querying of how it looks in a real project to make it more hands-on and a little bit more approachable. Um, I see someone not? Okay, then let's just do that for, for five minutes. And whoever else wants to leave for, for the, the Jeopardy or just to get some drinks or whatever, feel free to do that.

So only two minutes. Okay then, then we will do it really fast. I, I'm used to that. So I already did this with the talk. So, um, let's look at our list orders controller. So in this case I used an invokable controller. So one controller only has one action, the invoke action. Um, oh, it doesn't, wait a second. Well, since the time it's up anyway. Let's do it differently. Whoever was interested, you can come up and I will, I will show you on my notebook and since I can't get the screen to work. And everyone else, thanks for staying here, for, for listening to my talk. Please give me feedback. Um, and yeah, have a good day and hopefully see you around.

Chapter 16: When Testing Makes No Sense

Tip

SymfonyCon 2018 Presentation by Miro Svrtan

If you look at the stage of any conference in the PHP world, people are preaching testing, testing, testing ... If you on the other hand look at the community, the percentage of people writing tests is really low.

As a person who went from 'How can I ask for more time/money/resources for testing?' through 'ask for forgiveness instead of permission', to person who writes tests a lot, I still believe testing doesn't make sense. No, it doesn't make sense for all and everyone, often enough it makes no sense for me too.

This talk will explore that fuzzy line when you have to shift your mind from one side to the other: in both directions.

Everybody's got their caffeine? I know it's the last session of the first day, so, everybody needs it, right? No? Somebody can live without coffee. I need some help with how to do that.

The Back Story

So, first off, I see a lot of young faces in the crowd, so, do you know what this thing is? That's how old when I started developing professionally. So, almost 20 years now. So, I started off doing small websites because at that time people just wanted to have some internet presence. And far away from any kind of rocket science, even by those days, and for most of you today, it would be like, yeah, really what's the problem there? But that was the time that we didn't have cool things like frameworks, templating engines, database abstraction layers or ORMs. We had to handle all of that ourselves, but this was actually the fun part of the job. The problems started when you had to make it work for things like Netscape or Internet explorer 5.5. And if any of you are doing front-end these days and think you're screwed, trust me.

And, I got so uninterested in development that I actually went back to university doing some freelance on the side, earning some money. But about 10 years ago I found a team, I found a company, I found a really great job. It was a small product. It started off, let's say, like a dog house - really, really small. As we kept working on it, it kept growing in both features, customers, money. So, I was good with, let's say, carpentry.

So, in few months we had a shed, which meant more and more people could fit in. They wanted more and more, more money was coming in, more customers, more content, more everything. So it's grew again. You understand that you can't use wood to build skyscrapers, right? You have to change your base. So yes, I had to learn how to use concrete to get a big, better stuff done.

But what actually, my code looked like, something like this. You know, and to be honest, probably, in this case, this was done by somebody and then by somebody else. And for most of it was us, us. And, okay, it was ugly, but it worked. But what our customers wanted was bigger and bigger and nicer and not now concrete. It's aluminum, it's glass, it's shiny.

But as we kept growing, we kept failing. Because, hey, I didn't have the experience needed for that project. I used to work in small websites, small apps, not 100 million user visits a month. And what customers actually paid and what I thought was, oh, this is what we're doing. But in honesty, most of the developers don't understand that businesses want a nice feature, another nice feature, they really really want something to just glue them together. By design, because businesses don't know always what they want.

So, this project has took about five years of my life. And the next project that I came on, yeah, it's going to grow like that, right? It's going to be that small product is going to be the biggest thing ever. Unfortunately, what I was suffering from was Stockholm Syndrome. Because in reality, not every project or product becomes big.

So, let's say that you want to buy a dog house, like a real dog house. You have a dog and you want to buy a dog house. How would you feel if I came to you and told you that the price for this nice small dog house includes something like this? Because it's going to be a skyscraper so I want to do it before and avoid all of the problems. Right? Found yourself any time in that situation? So, Donald Knuth said: Premature optimization is the root of all evil. And, that's why I want to talk to you about when testing makes no sense.

My name is Miro Svrtan. You can find me on Twitter using this handle. So why am I talking about buildings and testing? What's the connection? So, who here is testing? It's an easy question, right? Okay. Who here isn't testing? Again, so, this

was a partially loaded question. Sorry for that. But actually all of us test. You write some code, you open your browser, you check that it works - it's also a testing. So, I hope nobody just writes the code and pushes it to production. To be honest, I did it few times. But not as a normal way of working.

So, let's say that we have a small website. No Mars rover thing, just a small website. And we have a bug. How much time do you think after somebody reports it will take us to fix it? Five minutes? An hour? Half a day? How much time does it take you to deploy? Probably 5 to 10, maybe 15 minutes. Right? So, why would we test and invest in testing when we can fix something quickly and deploy it everywhere?

But the problem was that about 7-8 years ago, a lot of web developers became phone developers. Been there, done that. A few of my colleagues went that way, but they didn't have the mindset of testing. Which is pretty nice for Google store because it takes about 3 days after you fix it for people to get it. But what if it's an iPhone? iOS app? 6-7 years ago it would take from 2 weeks to 3 weeks, sometimes even a month to fix something.

What if you're doing desktop applications? You have a bug and now all of your customers have to be somehow contacted to download the new version because, okay, maybe Mac users do have App store, but Windows users not so much. How hard does it then to fix a bug? It might not be a critical bug, but it might be something that bothers your customers, your users.

Hey, we have hardware. But that's at least easy, right? Because, this is a router, so it's connected to internet, right? So it's easy to fix it. If we have a bug, we can just ship it to all the routers. What if the bug is in the piece of code getting the update? Are you going to send all of your staff to the whole continent, to the whole world to fix people's routers?

When should we write tests?

What if it's... what if it's a washing machine? Hopefully, it's not connected to internet, but you find out that you shipped some of it and you can't wash cotton on 60 degrees. So, you now have to call all of your mechanics, all of your service departments, to contact the customers to flash the washing machine. So, when should we write tests? When it's a high cost to fix? Because, maybe, sometimes it's cheaper to test than to try to fix it once it happens. Maybe when the cost is larger.

Let's go back to the website. So, should we ever test websites? We can fix it easily, right? Yep. Right? Anybody? Nobody? Everything test? But banks do use websites as well. I mean I hope you don't experience a bug where you send the money instead of receiving it. The other way around, probably, you'd like it. Right?

So, let's maybe look at it as when it brings value. Because value is not just the money, it's how much your consumers love the product, how much your clients are happy with your work. Maybe for you, if you're a development company, maybe because of, maybe, it would bring value in the sense of people not leaving the company often because they would be more happy? There's a lot of cost when a disgruntled employee decides to leave because then you have to get somebody in. It might take about 6 months to get the new person in. So, let's define the value as benefit minus the cost.

So, what I'm trying to say that if, let's say, you have a tomato farm that produces a lot of tomatoes, you probably shouldn't use the same tactic - you might use on your own tomato garden. You maybe like your tomatoes a lot, but would you, um, would you create a whole lab? Would you test it every day? Probably for tomato farm you would because if one goes down - you might lose everything. Everything, if you get the disease. In case of your tomatoes in the garden, you can just go to a farmer's market or a supermarket and get some others. Okay?

So, value is benefit minus the cost. Of course, I do understand that some people might invest a lot of money into their tomato gardens because it's a hobby and we tend to spend a lot of money on our hobbies, but let's look at it as a business. The benefit minus the cost. So there is some, before I go further, there are lots of different things about testing. So what I'm going to try to focus on is writing test, automating the tests, the ones that help you ensure that the app does what was expected? Okay?

So, who here writes tests in that way? Okay. Who tests everything? Okay. 6 to 7 hands. So, actually, can you raise your hands up? Who tests everything? How much security testing do we do? Performance testing? Visual regression testing? That all is testing, right?

So, why I decided to do this talk was people came to a stage like this and said we test everything. And a lot of people understood that, yeah, to be a good developer you have to test everything. And they started testing, but they started testing the wrong things. I've done it that way. And, 6 months ago I did this talk in Amsterdam and I spoke to another speaker afterwards. He wasn't at my talk, but he was interested in what I was talking about and he told me: Hey, yeah, well no, you shouldn't be talking about when not to test because we test everything. I said: Okay, so performance? Oh, no. Security? Some. Visual regression? Nah, we actually TDD everything. Wait, you're using Symfony and you TDD everything? Yes. I was really interested because I found TDD to be very wrong for Symfony, the Symfony part, like using the Forms, using the Doctrine and stuff like that. And I was wondering, okay, how do you do that? Oh, we don't. So, wait, you just told me you test everything and you do it TDD way, but you don't do the UI, the infrastructure and stuff like that? Nope, that's the UI. So if I

wasn't really interested in that part, like how do you do that? I would think, yeah, I'm the stupid one because I don't test everything. Please, don't be.

Fallacy #1: It makes my project more expensive

There are a lot of fallacies with testing. 6 or 7 years ago, when I wanted to start writing tests, one of the biggest problems I had, I was a team lead at the time, was... So similar feature was about 100 hours. Now I need to set an estimate of about 150 hours because, well, writing the code, writing the tests, right? So it's gonna take more money from the clients. They're not going to be happy. What I didn't think of - do you charge for manual testing? So, do we say: Hey, it's gonna take me 7 hours to write the code and X amount of hours to run it in the browser as I click save on the code every time. We think of it as the same thing, it's part of development.

And, if you have to explain to anybody, try to explain to them the bugs cost too. So most of the companies should have, should solve the bugs for free. Right? I mean I know of companies whose whole business plan is to charge for bugs, for bug fixing. But you might notice that that's a pretty bad business model because somehow they want to produce more bugs because that means more work for them. Right?

Anybody here has a car? A new car maybe? Or somebody's buying a new car? Recently? So, let's say you want to buy this car. You go to, I think it's an Opel dealership, and you say: Hey, I want this one. How much does it cost? Oh, 20,000 euros, let's say, I have no idea, please, don't get me on the numbers. Okay. So, it's tested, right? Yeah. So can I get the version that's not? It's going to be cheaper, right? Because why would I pay for testing?

So products don't come out without those things. I mean, when you buy food, you expect it to be tested, right? You don't want to be poisoned. Try to ask for forgiveness, not permission. Just try to do it. Try to do it in a small way. Start building up your tests. Every new feature, add more tests. You don't have to do everything the first time.

Fallacy #2: It takes a lot of time

Another fallacy is that it takes a lot of time. That might be true for a first timer. Because there is a cost of learning. I mean, I guess, most of you that had a car that are driving had to go to school for driving, right? It took like 20 to 40 hours. Right? So, you don't take that into the cost of every time you go to the shop. Right? It's learning. Yes when your first project, it's going to cost you, your company, your product team, whatever. But it's learning. You wish don't learn.

Fallacy #3: It takes extra time

It takes extra time writing the code, writing the test. Right? So, if I rock, if I write 10 lines of code, I need to write some lines of tests as well. So, that takes time, right? Manual testing takes time as well because you have to Alt+Tab or Ctrl+Tab something, check your browser. So, if you write your tests as you're writing your code, you don't have to do that manual check. You can just run the test and it will tell you that, what you expected, works.

Fallacy #4: Ship PoC to production

So, for some years I thought that if I implement something that's ready for production, let's say, you want to implement a new social login or a new payment gateway and you get one PHP file where you see that it works, then that's it, right?

Phase 1: Exploration

But actual feature lifecycle is the part where you explore. You look how that system works. It could be a library, it could be a new database, it can be a third party integration, it could be whatever. That is the part where you don't test. You should explore, you should enjoy, you should understand what you're doing.

Phase 2: Modelling/architecture

Then you start modeling. Again, we're not testing, you're using tests but you're not testing - you're modeling. And this is a very confusing thing, when I try to explain it to people. If I'm writing tests, right? How is that not testing? So, if you are in a car, it doesn't mean that you are the driver. Okay? Uber and Taxi doesn't work that way. Okay? So, yes you could be using tests just like you use a car, but that doesn't mean you're driving it.

Phase 3: Development

The third part is development and that's where testing needs to be done. A lot of it. Because you're not sure about all of the edge cases that could happen. Here tests have a great value.

Phase 4: Production

And usually, when I go to production, I delete some of the tests. Because when I was writing them in the previous phase, I wasn't sure which of the tests will have value later and I don't want my tests to run for hours just because during the development phase I wasn't sure if that can break or not. I'm not saying delete all of the tests, but be... feel free to delete some of those that you do not need.

Fallacy #5: 100% code coverage

100% code coverage is something that I had a lot of problems with. And I hear from the others. If I don't check the code coverage, how would I know if something is tested or not? So you want to go and have 100 percent because in that case you know your app is tested. So, what usually people start is, well, let's get the code coverage as high as we can, as cheap as we can, which means testing getters and setters. It can be fun for like one or two entities, five-ish is Ballmer's peak. Oh, need of Ballmer's peak. About 10 - I'm happy if everybody doesn't quit their company.

So, should we test something like this? I really don't see the value. When I moved to using commands and events, command and handler events, command handler patterns, I found that I spent a lot of time on testing those setters and getters, once again. What now I usually do is if I do use code coverage, I just put all of the commands and event namespaces into ignore. And I just don't care. I want to achieve the higher percentage, but I don't want to know about those DTO classes.

Fallacy #6: We have "tests"

We have tests. I've heard some people complaining a few years ago about they wanted to participate in open source project. And they implemented the changes because implementing those changes was rather easy, but making all of the tests work was the hard part. You don't want you and your fellow developers to be in the position that, to change the code, is so hard because of the tests. Because people in that case will not test. I mean if you have a car on the bottom of a lake, what's the value of it?

Fallacy #7: Writing tests later is OK

I mean, and that most often comes from: Let's write tests later, which is most often coming from management. And as a disgruntled developer, I might come, I might build the whole feature. I know everything works and before merging it, my manager comes and says: where are the tests? We're not merging that without tests? No. Everything works. I don't care. Everything works. No. You have to write those tests.

What do you think? What kind of tests am I going to write? I'm just going to look. I'm going to split my screen. I'm just going to look at the code on the left side and replicate it on the right side. Because that's going to get shit done. So, yea, I'm just gonna add a simple case for this, maybe.

Fallacy #8: TDD has something to do with testing

Who here knows what TDD is? Anybody practicing it? Okay. If I say, if I say test driven development - is that TDD? No, sorry, another loaded question. Test driven design. Not development, design. You're supposed to use TDD to build your architecture, to model it. You can write your tests first and then your code - that's not TDD. TDD is that small red, green, refactor, red, green, refactor, red, green, refactor.

Anybody's remembering this tweet? Anybody's remembering the shit storm coming after it? Where everybody came with: Haha, you see, testing makes no sense. You shouldn't test anytime. Why were you forcing me to test? People don't understand the TDD and testing are completely different things.

Uh, do you all know who DHH is? So, he's an author of Ruby on Rails framework. He works for a small company called Basecamp, used to be called 37 signals and they have a few small products, really, really small products, really cool products, but really, really small products, that they have been building for about 10 or 15 years. What kind of modeling would they need when they know what they're doing? There's no modeling in it. They just need to test that what they want works. It's not really modeling. And, just to try to explain to you, that's one of the rare companies that would come to their clients, when they would ask for features, they told them: No. We have our product, we love it, as simple as it is. If you want more features - use Jira. Use this, use that. Thank you. Here's the door. So, don't forget the environment that this comes from. They don't build that many features.

Fallacy #9: TDD all the way

If you ever looked into TDD, one of the things that even the people who invented it said, you have to do everything the TDD way. Only in the last few years they started saying: Well, we were wrong. Because trying to build everything TDD way takes a lot of time. And if you're yet another payment gateway, yet another CMS, yet another E-commerce and you know how everything has to look, what part of modeling do you have to do? It's yet another house. Maybe has another color, but that's

about it.

I'm going to guess that some of you changed the company a few times. Have you ever come... have you ever changed it to company that has good tests? Whoa. Really? Wow. Three hands. Wow. So, my previous employer had such a great coverage that I started shipping the next day. I came on Monday and started working on Tuesday. Because it was easy to onboard me because I could have gone through the tests to understand what and why are things happening and even if I screw it up, because it's my second day, people, the CI says: Hey, this doesn't work anymore. It's much easier to add new features because you know that you didn't break older ones. You know that the older ones still work.

So, what do we had? This, the previous exam? We know this suck. We have to add a feature now. The feature is that if you're not from, um, from that European Union country, I'm using Portugal since we're here, you shouldn't charge a VAT if it's a company. So, maybe this is the point where you should test. Switching from rather simple if, to another.

What if you have to refactor something? So you have all of your tests. They run, you just change the code and all of your tests say: Hey, it still works. Still works. It's okay, everything's good.

So let's make this bit more readable to avoid nested ifs. So this is what happened. This is what I have. Can anybody spot the bug? Where? Here? Nope. You always have to charge VAT for that country if you're selling something in your country. Any others?

So, I just wasted two minutes of your life trying to understand if this was okay or not. So if we had tests for this, you don't have to look at this. You know it works. It's easier to rewrite things. And running tests means that we can do some fun while they're running. I mean at least for us in PHP world because we don't have compile times, which I'm something that I'm actually missing. But that's a long story.

Cleaner code. Trust me, it's really hard to test shitty code. If you write your tests as you're writing your code, you'll will write cleaner code. Because for a 15 line code, 15 lines of code, you might have 300 lines of test. And then you add an if, which means 600.

But maybe if you're building a small app, something that you have built often and that you understand. I don't think that testing brings any value. Let's say you are bringing, you're building a one-off app. Again, you can easily manually test it. That's okay. You know it works, but what happens a lot is, that simple app becomes a bigger one and a bigger one and a bigger one. When that one-off app for one concert in Lisbon becomes an app for all the concerts of that group in the world or all the concerts in the world for every group, you can't use the same thinking. When you're exploring something, if you're trying to learn a new language, a new library, a new framework, a new something - don't waste time testing. Enjoy it, learn how to use it.

Last but not least: job security. If you're a key person in your company and you have no tests, you cannot be fired. You can ask for whatever you want - you'll get it. I want a new car, I want the best Mac, I want 5 month vacation - you'll get it. What people keep forgetting is that at some point they might look for another job because they're not happy where they are or they have to move to another city or they get the shitty manager because their previous one decided to go somewhere else. You will not find a senior position that says that doesn't have listed testing experience, testing knowledge. I didn't trust people when they started telling me that 6-7 years ago, but for last 3 years, no senior position ad that I saw, came without testing experience. So, it's gonna be really, really hard to move to find a new job and it's just getting worse and worse and worse. So if you're like, my grandma had a few chickens. No fancy lab to test them. Please, don't think that you can use the same principles to have a chicken farm. It's different mindset. Thank you.

Chapter 17: Microservices Gone Wrong

Tip

SymfonyCon 2018 Presentation by Anthony Ferrara

Microservices are the latest architectural trend to take the PHP community by storm. Is it a good pattern? How can you use it effectively? In this talk, we'll explore real world experience building out a large scale application based around microservices: what worked really well, what didn't work at all, and what we learned along the way. Spoiler alert: we got a lot wrong.

Good morning, bom día. So today I want to talk to you about a little bit of a story. I want to do a very different talk than most other conference talks that you have seen or that you would have here today. Rather than telling you what to do or telling you what technologies to use or how to do something well, I'm going to tell you a story of something that I've royally screwed up. We all make mistakes. We've all gone out and built systems. And a team that I had the opportunity to lead a number of years ago, we wound up building a platform. And I'm gonna tell you a little about the background. But we made a bunch of mistakes and rather than let those mistakes die out and let me and the team be the ones that learned from it. I really want to take all of you on a little bit of a journey and tell you about some of the things that we did right some of the things we did wrong.

So today is really going to be a journey about what we built. Why we built it. How we built it. Where we ran into significant trouble, and where everything worked well. And this is my puppy Adda who's going to help me on some of these transitionary slides.

Background

So let's jump into the background. I walked in to a growing team. They had recently received a major round of venture investment, grew the engineering team from five to 18 within a year, it would later be about 30 people. And it was a very good mix between php developers, a couple people with Go experience, a couple of really, really good frontend engineers, data engineers, et cetera. They really cared about software quality. The people that they hired in were very, very good engineers.

But they were dealing with a legacy system, and we've all dealt with legacy systems in the past. This was quite an interesting one. The one I really want to highlight here is this bottom one. There were over a thousand cron jobs that ran as frequently as every two minutes, which corrected data in the database. That should tell you a little bit about, of an idea about the state of the system.

It was so hard to find and fix bugs that rather than fix them, they just patched over them and band-aided it. And that's not a knock on the developers. Like this system was around for five years, was built very, very quickly, was scaled with different varying amounts of talent. You know, a classical legacy problem. But the business needed more. The business needed stability. They needed an application that worked for them, that scaled with them. When I walked in, they had just gotten off of a nine month product freeze, which means that engineering said, for nine months we're going to do nothing but solve technical debt. We're going to do nothing but try to clean up the system. It was another three months before the first meaningful product release happened from there. And it was incredibly, incredibly painful to work with.

To Refactor? Or Rebuild?

So we were left with a decision, do we refactor or do we rebuild? The team had spent about nine months, like I just mentioned, trying to refactor and kind of not really having very much success with it. And so what we decided to do was not to rebuild. We went into a room - head of product, head of UX and about five or six other individual contributors and myself - and we went into a room with four whiteboard walls. And we started to ask ourselves, what does this system do? What are the core things in this application that it assumes. And I'll give you an example right now. Your system - any application that you work on - probably has a unique column on the user table for email address. Emails are unique within the system. That is an assumption. However, in most well-designed systems, that assumption is isolated to that user system. If you wanted to change that, if you wanted it to allow multiple emails to register for multiple user accounts, sorry, a single email to register for multiple user accounts, you just remove that unique, change your login, change your registration and you're done. This application, it was all over the place. Things inside of systems that had no business knowing about an email address relied on the fact that emails were unique. And so in order to change that assumption, we would have needed to touch about 60 percent of the code.

So what we did is we went on the four walls of this whiteboard room and we filled every single wall with a core assumption of what our platform was. And then we asked product: which out of these do you want to change significantly in the next 6 to 12 months? We wound up with, out of four entire walls, exactly three that were not going to be changed significantly.

And so what we realized is that it's not a refactor. It's not a rebuild, it's actually a V2 . It's actually a completely separate product that we want to build. At the highest of high level, it solves the same business problem, but how it does it is drastically different. We could have taken the time and refactored that in, but it would have taken three or four years to actually get to the end goal of where the product wanted to be. And so instead, we set us, set ourselves a four month goal to get an MVP of the V2 up and running. And this is the story of that MVP process.

Architecture

So stepping into the technical architecture here, when we started building this MVP, we had to have some kind of a guiding framework. And what we settled on was this, and I'll walk you through each one of the pieces. But the basic concept here is everything that runs on a server only does API calls. The frontend is the only thing - this frontend server here - is the only thing that actually knows about HTML, that actually knows anything about a browser.

Everything else talks through this gateway to a service over REST. So API-first development. The API gateway is the first thing I want to focus on because it's one of the things I think we got really, really right. This was in the neighborhood of 2015 and Amazon had just released their API gateway project two days before we decided to settle on Tyk. Tyk is an open source project, it's written in GO, uses MongoDB on the backend, it integrates well with console, with a lot of modern Dev ops tools. But basically what Tyk allows you to do is configure REST API endpoints with JSON. So I can hit an API and say, create this new endpoint, here's the backend server that it's going to deal with, I want you to handle OAuth for me, so terminate OAuth and just give me a user ID. I don't want to know any of that stuff. Handle rate limiting, handle quotas, do all of this stuff. And what it actually, one of the really cool parts is it had the support for middleware where we could actually have a very slowly-evolving frontend REST API, that our frontend servers built on, that our clients build integrations against it, etc. While our internal servers were more RPC based, a lot faster moving and didn't have to worry about backwards compatibility nearly as much. There was still some challenges there. But the API gateway pattern definitely is something that I'm actually really proud of from this that we actually built out, worked really, really fascinatingly well.

RabbitMQ & Event Sourcing

So stepping back out, the next really key part is RabbitMQ. Again, this is 2015 timeframe, the stack that we're building on is mostly php. We had some GO services, et cetera, and at the time Kafka did not really support PHP well. Go did not really support PHP well, sorry, Go did not support Kafka well. And so we decided to go with RabbitMQ. If I was doing this decision today, I would 100 percent pick Kafka for a reason that's going to be apparent. We were using this as a pseudo event sourcing database, meaning every time we made a change in the application, we would emit an event describing that change. Theoretically you could replay those events in order and get the system back into the state. I said theoretically because in practice it didn't really work that well. The, one of the big mistakes that we made was none of the services relied on Rabbit to set the state. So whenever they emitted the state changes it was kind of just advisory. So we would run into problems where the JSON wasn't fully populated or messages were just completely blank by accident and they weren't caught for a while.

if I were to do this again, I would absolutely use a similar-style system, but I would make it event sourcing first. Meaning that event list *is* the single source of truth and the services mainly use that, have a query against it, have a API where they can look at those events. But, so this was something that really was difficult to get running, caused us a lot of frustration, but in the end actually gave us a lot of insights. So it was kind of a mixed bag because having everything talking to a single archive meant that we had one single picture over everything that was happening in the application from the event stream. So while it was a pain in the neck, it also did help us a lot.

Service Layer

The other thing I want to talk about at this layer, is this service layer. So we divided our services into roughly three categories, domain services which are meant to sound like domain objects because that's kind of how we were thinking about them. So our business entities, all of our business logic, all lived in these types of services. They communicated over HTTP and they all had their own persistence. So they all have their own databases and caching systems. We also had asynchronous services that purely listened over RabbitMQ to do things like long running jobs, batch processing, we did a lot of video transcoding, et cetera. So all of that stuff was handled via asynchronous services. And then we finally had these things that we called meta services.

It's kinda hard to explain what a meta service is. So I'll give you an example in just a second,.but one of the challenges, we approached this design as normal object oriented design: your domain entities, you split them apart, you find your boundaries, you create your services just like you would do it in PHP, whether it's Symfony or Laravel or Zend or whatever

framework you want to do. And those services talk to each other. We modeled our microservice architecture off of very, very similar principles. There is something that we didn't consider though.

HTTP Calls will Fail

How unreliable a service call is in relation to a method call. So ask a question, ask yourself a question. How often do you expect a method to fail randomly? And I don't mean the method to not return because that would be somewhere around one in a billion. If you look at Symfony in a normal default configuration running in production, your front page may take 10,000 method calls to render. And how often does one of them fail randomly? Maybe one every 100,000 requests? But we're actually not talking about what happens inside that method because that stays the same when you go to services. We're actually talking about the method call itself. It's so infrequent that you've probably never even thought about it. You've never thought that: hey, I have this object, I know it's a valid object, I'm going to call this method on it, maybe that's not going to work. Whereas in a service architecture, you absolutely, positively have to. HTTP calls, if you're very, very good at operating an HTTP service, you're maybe going to get five nines, five nines of uptime, 99 point nine, nine, nine percent uptime, translates to one every 100,000 requests will fail randomly. And there's nothing that you can do about it. So compare those two numbers, one in infinity versus one in 100,000. This is what got us into a lot of trouble.

So the question is, how small should you build your services? We started with the idea of one week. One week should be about the amount of time that if you have a solid specification for a service, you should be able to go from zero to a working service. That way, if we made a mistake, if we had a significant issue, we could literally throw a service away. I've heard some people talk about that number as a rough benchmark. And at this point in my career and after this experience, I will tell you that is an absolute mistake.

Here's why, this was a rough model of our domain. We did an e-learning platform that would deliver lessons via a web platform. And so we had users, we had assignments which assigned lessons to users, we had a history which showed what lessons a user interacted with. We had content associated with lessons and we had assets associated with content. And the way we modeled this as a system architecture was roughly every one of those entities became its own service.

Now it's actually a bit more complicated than this. This is a little bit simplified to make the point: each one of these services did have more than one database table. It did have more logic built into it, but this is the rough concept. And so this raises a question. If you are building for the frontend, how would you get everything that you need to service a request or to render a page that shows an assignment? Today and in fact, back then, one answer could be GraphQL, right? GraphQL is phenomenal at stitching it things like this together. The problem was this need isn't just had on the frontend. Backend services needed to look at things and aggregate as well.

So that's where these meta services came in. They had domain knowledge. So it knew what an assignment was. It wasn't just stitching things randomly together. It knew what it was creating so we could actually add some business logic in there around that. But most importantly, it acted as a pain function for developers. If we got our backend model wrong, we would feel it when we built that meta service. And so by forcing us to maintain a service to fill in the gap, it forced developers to realize that: hey, this other model was bad. Or this other model had problems.

So let's ask another question. How would you get a list of lessons ordered by the author name of the content within those lessons? Think about where that data lives. You want to order by this user service over here, but you want to join against content and then finally return lessons. That would be an absolute nightmare to do over REST. I mean that's, you basically would have to query every single row from every single service and try to stitch that back together. It took about six months to solve that problem once we sat down and tried to do it, which is where I think the real failing of this architecture and going this small on services. We ignored what the business domain was. We ignored the bounded context and went way smaller than was necessary. And to be fair, we didn't know that this type of requirement was going to exist when we built it. So keep in mind, keep things as wide as you can and only really cut those services when those boundaries are clear and easy. By the way, the way we solved this was with a service that used ElasticSearch and basically kept its own model of everything in here and became a generic search service, which yeah, was challenging.

Infrastructure

Let's talk about infrastructure. I just told you something that we got horribly wrong. Now I'm going to tell you about something that we got ridiculously right. I think at least. We had been running Mesos, Apache Mesos for a while at that point because we were using Spark. And Apache Mesos is basically very similar to Kubernetes but for arbitrary jobs. You can run a farm of servers. You give Mesos jobs and it figures out how to run them and it runs them across the cluster. So we had a lot of experience running Mesos and at the time Kubernetes was really not a thing. I think they had just announced it. It may have even been alpha, but none of the cloud services supported it. And so we decided to go this direction, running Marathon, which was the Docker scheduler on top of Mesos.

Took a little bit to get running. But once it was running, it worked phenomenally well. And so I want to take you through the life of an actual request just to show how powerful this was. The very first thing that happened when you called an API was it

would hit an external elastic load balancer, an Amazon ELB, which are very, very, very reliable, but kind of slow to reconfigure. Whereas Marathon would want to reconfigure things every couple of milliseconds at times. And so the ELB would talk to a ha proxy instance, which, was a little bit less reliable but was very, very, very fast to update. Those requests would then go into Tyk into our gateway and then Tyk would call an internal service inside the firewall to an internal ELB, which would then go through the same process and hit our service. This looks heavy, but including Tyk, including OAuth termination, including rate limiting, including the REST deserialization in any middleware that we had in here, this entire process took about 10 to 15 milliseconds. So really, really, really fast and gave us near infinite, well not near infinite vertical scalability let's not go that far, but gave us a good bit of vertical scalability with this, or horizontal scalability actually.

And then if that internal service wanted to talk to another API, it basically just bypassed that external step and just talk straight to that other service through that ELB. So what we wound up having was a system where if we wanted to add nodes, we could drag a slider and within 30 to 50 milliseconds have every single machine running a new service and the system would reconfigure itself. We were deploying on average, I think it was about 500 machines per day where we would spin down old machines and spin up new machines and we had almost 100 percent uptime during the time we actually ran this, that I was there. It turned out to be very, very reliable once we got it up and running.

Touching on logging really quickly. Logging is insanely important when you're building a distributed system. This was one of the core things that we did initially using LogSpout to collect, to stick things out into DataDog. So StatsD as well for application metrics. And we also used something called Zipkin. Zipkin was simultaneously one of the biggest pain in the rear ends that we worked with as well as one of the most powerful tools that we had. Getting it running at least at the time was an utter nightmare. Once it was running, the data that we got was incredible.

Basically, when you have a service that calls other services, you pass along a request id on that other call. And Zipkin, looks at that data and is able to correlate requests. So you can see for one service, it may have taken 100 milliseconds to serve that request. You can look at every single sub request, every single part, no matter how far it fans out into your system, all from one graph. You can see the network effects, the configuration that happens when you have one service fail, how that affects other services. Zipkin took us a very long time to get up and running and we paid a significant price. A lot of things would have been a lot easier to debug had we have gotten that up and running sooner.

The final piece on the infrastructure side, we implemented something called, that we called the service.json file. This basically lived inside of every single services repo. It described the name of the service, which would turn into a DNS name. It would describe what other services this one required to run, what database it needed, and so we could actually spin up databases, run migrations against them, configure the credentials 100 percent automatically. Same thing with health checks and the APIs that that service exposed, as well as all the things that Mesos Marathon needed to run it properly. And so with one file to get a new service into production, all we needed to do is go into CircleCI and say, build this project. And it would automatically deploy everything into production when you merge into master. Worked actually really quite well.

One thing I'd point out does anything that I just talked about look familiar? This is basically Kubernetes. That's what we wound up reinventing and looking back on it, it feels a lot like we reinvented a lot of the wheel. But the reality was we were just a little bit too far ahead of where things were coming and I don't mean that in a good way. But we wound up reinventing a lot that was ultimately coming down the pike for major open source projects. And today just use Kubernetes. You don't need to build the rest of it and you can get the same exact benefits.

Local Dev Experience

So moving along, that's how it worked in production, at least in theory. Local dev was a little bit of a different story. So the initial intention and what we built at first glance was a command line tool. So you would check out a repository for service and you would run this command line tool in it. And it would read that service.json and figure out what you needed to run your service, configure a docker compose file to spin up all of the other services to run all the migration files to get your databases all set up and everything like that, and get everything up and running so that you had a local dev that basically exactly mirrored production, in theory.

The problem was nobody actually used it. Every engineer ran their own service natively on their machine. And when they needed to run a dependency run another service that they had to talk to, they would either mock it themselves by creating a little, you know, Node.js script or a little PHP script to simulate that endpoint, or they would talk to another developer to get the other service running on their machine. And this wound up having a big problems because the amount of times that they would update that destination service was not really that frequent. And when they did, they weren't really in the habit of running migrations,.and so when we went to integrate this in production services weren't used to talking to each other. It became an absolute nightmare.

And so we stopped and we asked why. Why weren't the developers using this tool that built and worked? That was built and actually worked? And the answer was actually pretty simple. There's two layers to it. The first is that tool was slow. It took on average, because of the number of dependencies that had to spin up, and it basically started from a docker, from a fresh slate every single time you booted, took about 20 minutes to get up and running. And you figure you do that once a day, once

every couple of days. But that also means anytime you want to reset the state of the system, you have to wait 20 minutes. That's kind of a ridiculous amount of time to wait. It was also really unreliable about half the time it wouldn't start,

But there is another key point. The way we had built the team in the way we had built this tool was such that it was somebody else's problem. There was somebody who is designated on the team to build and maintain this tool. Developers had agency over their service.json file, but they had no agency on whether or not their system ran in this tool or not because ultimately the tool didn't matter. What mattered was prod and as long as prod worked, nobody really cared. And so that was a really, really huge fault and failure on my side. And one of the big things I learned as a takeaway is you've got to, especially in a services architecture, you have to get the local environment right and consistent because that will be the difference between a system that works in a system that doesn't work.

Getting it Running

And speaking of that, I want to talk about actually getting this thing off the ground. So we spent about three weeks building the first couple of services, authentication, user service, and a basic content service. And we had all three of them running in local dev and we went to put it into production. And it took a month from that point before the first successful login on the frontend happened. A month, from working code to serving traffic. That's just absolutely ludicrous.

And so we started to ask ourselves, we had retrospectives after retrospectives and there is a whole bunch of reasons for it. One of the key ones being the local development environment was not stable. So therefore getting into prod was actually the first time these services ever talked to each other. And so the services were built in isolation. It really caused a lot of pain. Some of this was just normal growing pains, you know, you're building your infrastructure to kind of an ideal, you expect health checks to behave exactly a certain way and in practice there's a little bit of fudge room for that. And so there's always little kinks to iron out, but it was really, really challenging.

A little more detail on a couple of these. I'm not going to read these each independently. I'll get to the coordination in a second. But getting the local state and staging environments into a known state was insanely challenging. So what I mean by known state is, let's say you have a bug in production. How would you replicate that in a local environment? If you're dealing with a monolith, maybe depending upon your security requirements, you clone the database? Or if you're actually doing things well and GDPR compliant, you actually have a system in place to anonymize that data? Or even better yet, you're able to recreate it directly on your local without having to copy any data.

That was literally impossible here. Every service had its own database. There was data in all sorts of places and Rabbit MQ queues, in S3 that the system used. And so to get the system into a known state was literally impossible. The only way people could actually do it was by going into the interface and clicking things in the interface to create content and to create assignments and all this stuff. So both from a debugging standpoint, but also from a build standpoint, it was insanely, insanely challenging. We had an idea for a tool to solve this problem, which was basically to detail the state in a yaml file and give it to a tool which would then coordinate with all the services to get everything into a known state. It seemed like it would have worked to solve the problem, but again, just a matter of time, we didn't actually get a chance to finish it.

Dealing with Change

The high-level coordination is a really interesting challenge. How do you deal with change? So change is always a natural part of software development cycles and of getting things into production. You know, we're never going to get it right the first time, whether it's feedback from clients tell you that you built the wrong solution, whether it's an executive stakeholder that walks in and goes, I demand that you add this feature because I'm just executive and I demand things. Or maybe it's because we actually literally screwed up the first time that we did it and we misunderstood the problem from an engineering standpoint. We built the wrong thing. So change is a natural part and, you know it's going to happen.

Here's an example of a real change that we faced. This is a rough abstraction of one of the hierarchies in our system where a program has many topics, a topic has many lessons, a lesson has many cards, and a card has many assets. Don't worry too much about what these things are. Just think of them as entities. What happens when the business comes in and says, I need a new layer?

In a normal monolithic application, this would be trivial. You would add a new database table, right? Maybe a little bit of a migration and 90 percent of the time things will just work and maybe you spend a couple more days cleaning stuff up and adding the UI elements and stuff like that. It took a month to do something like this because, remember, everything is a separate service. How would you make a change to this hierarchy?

Well, it turns out, you first have to make the change in anything that depends upon that service. And you have to make the change such that it can accept either the old or the new way. And then you need to add that new service. You need to run all the migrations to get all the data into that new service and then you have to change all of the other services again to get rid of that duplication, to get rid of the acceptance of the old way of doing things or the new way of doing things. And so basically what would have otherwise been a very simple refactor turned into massive coordinated surgery. Typically these type of

refactorings required half to three quarters of the team working for weeks at a time. And these are things that any application goes through, like these are not complex things.

And so what we should have done, in retrospect, what I would have rather done is this: take our domain model and group it by bounded context, group it by the things that are similar that need to know about each other and put these other services out in other areas. Asset service is its own thing here, it could be part of lessons, but considering video transcoding and stuff like that, there's a lot of things that are unique just to assets. And this is ultimately what we wound up doing. We did wind up throwing away three quarters of those other services and building what some people would call microliths.

Lessons Learned

So I want to wrap up a little bit here with some lessons that we learned and some key takeaways here. First thing that I would recommend is don't do microservices. Now this is a little bit of a joke because you can very clearly see what we did at least initially was not really microservices. It was a distributed model. But I would strongly, strongly say, do not build a non monolith unless you can invest in operations, unless you can invest in automation and tooling and have every single developer own and be responsible for that tooling, it's not gonna work out well. Or at least, it didn't work out well for us.

Start with big services, especially when you don't understand the problem. You may think that you understand the problem, but unless you have solved that problem in production, start with a big service because it's far, far, far easier to take something that's big and complete and split it into, break it apart because you know where your challenges are. You understand your domain, you understand the problems. But to try to stitch two services together, especially when you have dependencies in other parts of the application, becomes insanely challenging.

Automate absolutely everything. And I don't just mean deploys. I'm talking about your infrastructure changes. If you're not using Terraform, use Terraform, etc. Like make sure the backups are 100 percent automated. Make sure that how you get the application into a known state is automated. Automation and autonomous systems are absolutely critical, especially as complexity increases.

This is, I think, the biggest lesson that I learned. Anytime that we're really dealing with distributed systems, but a lot more so applications in general. Normally the way we write code is we write the happy path first and then we test the sad paths and we write the sad paths. And even TDD with the red green cycle is designed to kind of do this. You don't write the failing test for your failure case first, you write your failing test for what the business problem is. Then that fails and then you write the code to solve that. Think failure first. Start off with: what happens if this thing breaks. And write the code to solve that first, because things *will* break. Your database will go down, you will have corruption, you will have a network failure. No matter whether you're building a monolith or a distributed system, things will go wrong. And changing your thinking to start thinking about what's going wrong, what's going to happen, what's failing and how will I gracefully handle that, becomes absolutely critical at maintaining a scalable and highly available system.

And then finally, this is I think one of the things that we thought we understood in the beginning, but we didn't. SLO's are a concept that came out of Google's SRE book, which is their service-level objectives. Basically, what does the business care about for this service? We measured everything in our services. We measured requests per second. we measured memory usage. We looked at number of database queries per call. Like any technical metric you can think of we probably were tracking it. But we weren't really tracking with the business actually cared about. Yeah, we thought we were. We thought were, you know, number of pieces of content accessed per second, but that's really not what a business cares about. The business cares: was a lesson consumed? Did the learning happen? It was an e-learning platform. Did a service respond in a timely fashion and not a timely fashion from a technical definition, a timely fashion from what the actual end user cares about. And so by defining these SLO's when you, before you build your service, you actually have a metric that you can test your service against.

The biggest takeaway that I have from this entire experience, and I've talked about this a little bit on twitter and I know there's different opinions on this, but I've learned that complexity is the number one enemy in software development. Everything that we do: when you look at refactoring, when you look at object oriented design, are ways of managing complexity. And quite often what we do is we take complexity from one part of the system and we spread it out into the rest. This file is easy for me to read. This code in this method is easy for me to read. Therefore, it's simple. Therefore my application is simple. When in reality you didn't simplify anything. You just moved the complexity from one place to another. And so managing complexity becomes the absolutely most important thing to building a reliable system.

The way I would phrase it is this, it is insanely simple and insanely easy to create a system that is so complicated that you cannot understand it. And if you can't understand it, how can you run it? This is basically the story of the system that we built and where it failed, where it broke down. And so I want to say thank you and I think I have a minute or two for questions if anyone has any.

Come up to the podium. Nope. I'll throw the cubes. Ok, to anybody? Or to questions? Okay. Here you go.

Thank you for awesome presentation.

Oh, okay. Who wants the cube? Who wants a mic? Up here.

Can you hear me? So, uh, I wanted to ask you a question like: normally when you're trying to split the system into smaller services, you often end up splitting the complex logic into complex.... You can't hear me? Can you hear me now?

Yeah, it's, I can't really hear you over the background noise.

Uh, so I'm going to try my best.

Yeah, that's better.

Okay. So when you're trying to split the large system into smaller components or microservices, you often end up making the service simpler but the communication harder and more complex as you said. So how would tackling such a problem be? Be like, identify, be like making a service more responsible communication adaptation for data and then forwarding data and formatting it? Or how, how would you advise people to tackle that?

So I think that's where I get back earlier where I said failure is the key and thinking about those failure modes when you're splitting that apart. Think of the happy paths when, if that service is offline or if those data communication channels fail, how would that behave? And try to work out the system such that they actually do behave correctly, when those things happen. Or they can gracefully. Unless I misunderstood the question, it's kind of hard. Come up after, we'll chat afterwards about it. Alright, thank you.

Chapter 18: Security: Handling User Access with Symfony the Right Way

Tip

SymfonyCon 2018 Presentation by Diana Ungaro Arnos

We often overlook a central security requirement that any application needs to meet: controlling users' access to data and functionality. Usually, we handle user access through the combination of 3 security mechanisms: authentication, session management and access control. We will take a look at the Symfony's Security component powerful tools and see how to use them to handle user access the right way.

Good morning everyone! Did you drink your coffee already? So, because you know, if you feel a little bit sleepy I can scream really loud, and I maybe do that sometimes during the presentation, please don't be scared. I promise I'm a really nice person, especially when I'm sober, which is right now. So we are fine. Uh, I must say sometimes I say some maybe bad and ugly words and if you feel offended by it, (xxxx) you. Oh no, sorry. If you feel offended by it, okay. I can ask, you can come and ask, I can ask you a, you can ask me for apologies later. And no problem, I can give you hugs and everything is fine.

So, well, um, I, I must say also that I'm from Brazil, I'm Brazilian, so, uh, I'm sorry if I mess up some words in English because, well, it's not my mother, my mother English, it's really fine. That's not my mother language. So I'm really sorry about that too. And maybe no, (xxxx) you again.

So I'm here to talk about security. Um, what I want to talk about is, uh, the Symfony framework that have, um, amazing tool called the Security component. You may use it with your whole application running on Symfony or you can install it by small parts, by packages using Composer.

And while I was studying this component, it is extremely powerful, but I've been searching for, you know, examples on the Internet. Um, examples of code, I, I read, uh, lots of lines of code on GitHub. Um, I talked with people I know about the component, the component, and I realized people, like, open them the documentation, see some lines of code and it's always Ctrl+C, Ctrl+V. People don't understand the concepts behind, um, some security problems and security solutions. Okay? Um, if you don't know the concepts, if you don't know how things work, if you don't know the context you're dealing with, I'm so sorry you're a (xxxx) developer. Okay. Um, and this is not a problem. Everyone is a (xxxxx) developer on some level or something like: I can do frontend. I mean, css hates me, so that's why I stand in the back end stuff. And especially for starting, if you're new you, if you're a junior developer, it's good to be a (xxxx) developer, you know? Wake up every morning, look in the mirror and say: I'm a (xxxxx) developer. Because this will make you try harder when you go to work, when you try to study. Okay.

Hi Diana!

So I'm going to play around here with some concepts. And I will present some, some of the functions and tools you can use for, from the Security component to handle user access specifically. We have some stages of user access, but before I enter that, that's me. Um, there's this... Oh! My light doesn't work. So you can find me everywhere with Diana Arnos. That's my name. Uh, I like Sec, Music, I am tech leader from the startup in Brazil and I am evangelist of the user group of php and São Paulo Brazil and the Brazilian chapter of PHP women that, yes, we have a Brazilian chapter of PHP women, okay?

User Access

Well, let's talk about user access. When we think about user access, well, it's easy right? You just have to put some username, you have to match up the passwords and let the user in. It's like, oh, this is you, so you can get in. But is this really that simple? How, how do you think of when you imagine the user entering your system, what things our user can do? You know, it doesn't matter what kind of user we are talking about. If there's one thing we know, every user is really good at, the user is good at (xxxx)ing up things. The user will click where it shouldn't. He will try a URL it shouldn't. He will end up at the page you didn't even imagine it existed, especially for dealing with legacy, if you're working with legacy code.

So we have a few steps to think about, a few situations when we talk about access. You have the user and you must, you can notice I'm a really good artist, you know, and then you have to do the auth. Uh, it's interesting to notice that when we talk about auth and we use only the abbreviation, it's because it's the same for authentication and authorization, because it's, they are separate things. Okay. Not many, not many people think about that, but they are separate things. Okay, so you'll have the

authentication step.

After the user is logged in, or it's authenticated, it will have access to many features, whatever they are in your system. Those features enable the user to access some data. It may be only visualization, it may be only to edit or create, whatever. Every data on your system is accessed by the user through some features. Yeah, if that doesn't happen, you're having a problem, around, like... I will try to not talk about sql injection because you know, it's a very common mistake people usually do, especially developers that they don't pay attention to what they're doing. But I'll try it hold myself, it's not what I need to talk about, but I really want to punch everyone in the face when talking about that. So you have session, and you have an interaction with user authentication session and data and feature all the time through all the system and actually everything at the same time. It's much more complex than we usually think about. And then you have the authorization process: that's involved around everything. Basically the authentication process is... and the user data you have, they're combined throughout the system to know, okay, does my, does this user have access to this data or to this screen, to this button, whatever. You know, when we talk about not even only user roles, but what can the user access, it may be not only a simple visualization, you know, maybe as a low security-level user, couldn't see the names of the high directors of the company, for example.

User

So we have, everything begins with a user. You can't have an authentication if you don't have the user. And let's suppose we are using Symfony here. It doesn't matter how you create your user class. The question, you may use, you may cod it by hand or you may use the MakerBundle, which is something very fun to, to play with, but you have our user.

User Provider

But then we have something really important for all the authentication and authorization process that will be used by the Security component: that's a user provider. What does the user provider do? It gets, um, like, let's say, support methods that handles your user data in ways authentication can do with it. You have two basic methods that must be implemented by your user provider and, but you can also add a few more, once your start playing a little bit more with the Security component and specially Guard authenticators that we'll talk in a second, okay? So you have a user provider. The user provider does basically two things,

Reload from the Session

I must say, it *must* do at least these two things. First reload from session. There is a way to disable that and I'll talk about that, but what is the reload from session stuff? It serializes the User object and at the end of every request, every request the user gets, the user object gets serialized in session. And after, every time it begins a new request begun, or begins, sorry, uh, it deserializes the object. But it's not only that, it deserializes the object from the session, then it makes a refresh from the user based on the data on the session from whatever your user data is: it may be an API, please don't do that, let's do REST, but it may be an AP, it may be the database and it may be, I don't know, memory, because it can do that too. Please don't do that, but you can do that from memory too - configuration files. And then it compares: it's like, okay, I have this user, then I'll get this user from my database. It compares both and sees: Oh, if it's not the same user, it de-authenticates the user: it makes it login again. There's some internal methods that verify this that's outside from the, the part of the user provider. You can play around on that too.

But it's a security measure to guarantee that, well, if you have some kind of middle-man, middle-man, it's a very nice name, you know, a man in the middle attack, okay, if you have any kind of man-the-middle attack, um, you have many kinds of men in the middle, but one of that, one of the most dangerous attacks on that is, right playing with the user session. Cause, you know that that things start there. So let's suppose I am the man in the middle. Then I will try to get the data from your session and then I'll start playing around because the server will think I am you. And still, still the, the, the component doing that - verifying if there was any change between the original user from the user database and the data on the session, there's still ways you can play around and (xxxx) this up. Because, well, if there ain't no time to updated the user or the data didn't change anything, you can still have problems with that. And you have ways to deal with that here too. Because, you can use a class that's a, an interface you can put on User class and you can implement them a method called isEqualTo(). So, instead of using the, the, the, the default method from Symfony, you can write your own. But be careful, great power, great, brings big responsibilities.

Load User

And then you have the second method you must implement that is the load user. Uh, you may look and say: but they are the same thing! No, they're not. The load user does not do anything with the session. The load user, every time you have a feature that needs the user data, it will be calling this method. Why would you implement the load user? Well, like, if you don't want to use the default methods and you are using like an API, the built-in user providers that Symfony comes with does not give you the methods needed to get a user from an API. So you can implement this here. Okay.

Custom User Providers

And here, and this is one of many use-cases that can do, you have to get the user from an API. And now I feel a little bit better talking about that because ya know, you guys were watching Anthony's talk and he was saying to please don't use microservices, and I was going to say if you're using a user micro-service. So I can say we're using a big user service, okay? We're going to recover the user data from the API. These are the methods you're going to mess around. loadUserByUsername(), funny thing is, that's, the username is not necessarily the username. It's the thing that comes if you have... It, it's the data that comes when you use the method, getUsername() from the user. Oh, but if I'm using the User class that's default from Symfony, you will get the username. Okay. But you can also make your custom User class, so you can change your username, I dunno maybe your identification of some kind. But then you can create and code all the data that comes here and this gives you really power over your application. They get, this gives power to: Okay my user is logging in, I am authenticating this user. How am I to authenticate this user? And uh, I know that most people doesn't care about that. People like, okay, I have used the full functions. Don't do that, please.

Well, everything, as everything you can do on the Security component, you configure it in the security.yaml: that's right: config/packages/security.yaml. Uh, when you make your custom provider, you can set them here and you can just add the id, user provider. Uh, I didn't show the, that piece of code because I was showing the things you needed to, to code, but at the end of the class, there's a method that supports class, that's called supportsClass() that will show the framework you're using the class as a custom user provider. But it's a, it's a, it's a common line. You can find the line of code on the documentation itself.

Authentication

After the user: Ok, I have my user, I have my user class, I have my user data and I have a user provider. The user provider will always be used during the request and authentication and authorization methods. Okay. I did this step. So now I'm going to talk about authentication.

And we have authentication providers. Uh, I don't know, most people, first time they're trying to mess with, uh, the Security component, they are trying to do things from out-of-the-box, because you can do that. It'll just put: okay, I created a project, I have my user, I can use the MakerBundle, everything's fine. So, you need to fix or do something different with the authentication. And then you start, like jumping from class to class and you're like: oh my God, this authentication is magic because it's not in the controller, it's not everywhere.

The authentication on on Symfony runs before the controller is called. It's almost like a, a middleware idea, it's not exactly a middleware, but it runs before the controller. And for that, you can create authentication providers. Symfony already has some authentication providers built-in. But if you read the documentation, they will ask, they will say it's better if you create your own. But here, you don't have custom authentication provider, you have what's called a guard authenticator. But let's suppose we're going to, you have... When you create a user provider, you have one user provider for your class. Here you can have lots of authentication providers. And every authentication provider is, they always runs before each request. Always. And so you may have, you need to pay attention to the order things are happening. How many providers you have on authentication. And you must take care to, don't let one mess up the other. Okay?

Umm, okay, I won't use the built-in, please don't use the built-in, I repeat this many, many, many times: please don't use the built-in authenticator. And then, you have to do your own. You can create a guard authenticator. It runs at the, it's the, on the same, the same context; they run every time before each request. You can create lots of guard authenticators. But, this class, when you create a Guard authenticator, it gives you full control of the authentication process. You can, you can analyze and, you know, deal with data since the beginning of the request to the end of the request. You will use an interface that brings seven methods you need to implement, but you can put more, and you can do a lot more.

There's an example. Um, I'm really sorry I messed up the PSRs here, because you know, but it's just so they can fit all in. These are the seven methods you need to use, at least, by extending the AbstractGuardAuthenticator. Okay? Uh, why, um, why would you, need most authenticator, to create one by yourself? You know, like if you want to use a JWT authentication, API tokens authentication, you don't have those built in on Symfony. Anytime, basically everything you need to do with an API, you have to create an authenticator, a specific authenticator.

So, here you come from, since the beginning, you know, this function here, the start() one. It's basically, it tells the authenticator what to do when, okay: this user isn't logged in. Or this user is incorrect. It runs the start() authentication. Uh, supportsRememberMe(). this function, remember me, you find on the user provider class or the user class, it's basically to get the user from the session and keep it logged in. You may disable that if you want to. You may check credentials, um, what we'll do an authentication success. If you look at this class and if you look up on the Internet for examples of its implementations, you'll find that some, a little bit of similarity with programming by events, you know: when it does logged in, when the request starts, when the request ends, when it's not the same user, when they users change. Okay?

And to configure that, you can do that, or again, on the security.yaml. Here is a firewall. I'm not talking specifically about the

firewalls because it's a very extensive subject and it's very fun to play with too. But let's suppose we all know what we're doing with the firewalls. And, pay attention, if you don't know what you're doing, you're going to mess up your whole application. But down there, in my main firewall, I have some authenticators. And why is this authenticators? You can have lots, as I said before. And then you just put the namespace of your personalized authenticator. It's really simple when you do that.

Uh, I would like to call the attention to the bottom lines here, because there's an option in your firewall, in your configuration that's called stateless, cause I'm talking a lot about recovering user from session. There is a serialization that goes on session of the user, blah, blah, blah, blah. When you'll turn on stateless, you can work with REST API's, because there's no state on your session. And I must say it's one of the best ways to do that. I try to, to think like that, because you can avoid a lot of security problems by doing that. But you can do stateless: true. If stateless is true here, you don't need to implement that method that, that's that load users from session. You can leave it blank. We can leave it whatever message, because when your application is configured for stateless, they won't be calling that method any time soon. So.

User Roles

And then, you have, I can say, talking some of, like, it's like a sneak peek on user roles, because you know, it's again, a very extended subject, but before I talk quickly about user roles, I will like you to think about, you're maybe thinking, man that girl is talking (xxxx) the whole time, why I'm sitting here, blah blah blah, user here, blah, blah, blah, blah again and blah, blah, blah, stateless. But to get to here, because every, everyone loves to talk about ACLs. Okay, I have to handle access. I have user roles. But notice, pay attention to how many steps you must think before you get to user roles. If we're not thinking about that, you may have problems. That... you *will* have design flaws in your application and that can be, and will be, exploited by malicious, I would say malicious attacker, but an attacker is malicious anyway, so.

Um, user roles: this is a quick configuration, You'll do that on your User entity. Um, you have them, this method getRoles(). Um, the big thing here when you talk about user roles is that you can have a hierarchy of roles, and, you won't get this: Okay, I will make this admin role and then I will configure this admin to have access to this page, that page, will visualize this data, that data, but actually you begin configuring your roles from the bottom to the top. So when you talk about the admin role, you just have to say that the admin role have all the others. Uh, another interesting fact, when you login or access your application, without being logged in or logged in as anonymous user, uh, the, the, the component have a very interesting concept that, uh, it's like, it's not exactly that, but it's like the anonymous user, it's kind of a user role. So even if you're, okay, you're coding then, and you're and you're trying something and you have the error on dev mode and you have the profiler bar under your application, if you are like, anonymous, okay, I'm not logged in and I'm, I'm on the poor mode of the browser. I mean the anonymous mode of the browser, and you answered that and you go check if you're authenticated, because you have the user information on the status bar, you'll see that it will show you that you are authenticated as an anon. And basically, uh, the user role anon don't have any permissions. So, every time you are not logged in, the application and treats you like you are logged in with a user to have no, no permissions at all. And this is a very interesting fact because when you're coding your application and thinking about the security issues or access, or permissions, you don't have to think on the "no" option. It's like: okay, I will remove this and that and this guy can do that, no, he's logged in as anon. And you can even mess with that and give the anonymous user access to some features of your system or your application without being logged in. And you can do that simply by using a (inaudible) the configuration, setting up some paths or URLs or patterns that you're, the anonymous user can have access to. Okay? And you can mess around with that too. You can, you've: okay, you have access here and I'm using the Guard authenticator and for any anon user, it can only see this kind of data, not that kind of data.

And this is where you would configure. The guy here: access_control. You have your firewall configuration. I was, as I was talking, the anonymous user there in the main. Then you have access_control, so, you can play here. This, this makes your life a lot easier when you will filter access on your application. You may give that: Okay, I will enter, um, I don't know the welcome screen after the user has logged in, and I have a dashboard, but I have a specific dashboard, I don't know with financial information that only the admins can see. So when I have a low user accessing, I can author that information right on the template, on Twig. You can call the user role that, it's like app user access.

But let's suppose you're not thinking of, on that kind of detail. You can do something quick, or quicker, or faster. That's the better word to say that. So you have it here: access_control and then you can say right here, like what's the role? Which role have access to which pages? Or you can think about path, you can think about URLs or hosts, that's the option you can use, and then you can make everything just from the configuration file. If I have a simple application, if I have, a with only like editing, I dunno employee's information, you can set it up through here. And every time we need to change something: Oh God, the director's said that the guy from the next department have to see the page "B", now, and he can only see the "A". You don't have to change on lines of code, you can do right here. But of course you can only do right here easily if you pay attention since the beginning: user, user provider, authentication, guard authenticator. How does, how do I handle my requests before I reach the controller? Because with, sometimes we are used to doing authentication on the controller. That's not one of the best ways to do that. So. And that's how you can play with user roles. Okay.

Um, I could talk a lot more about that, specifically security, I like to talk about that, but unfortunately I won't have time to talk about all the stuff that comes from the security component. But I would like to say thank you. It's a, thank you for being here. Thank you for listening to my bad English and I'm here to answer all your questions. Not only here, but I'll be here all day. So come and talk to me on Twitter and Instagram because I'm kind of hipster developer. So if anyone has questions, or you may, I don't know, curse me? I can hear that too.

Thank you. Thank you.

Chapter 19: Webpack Encore: Tips, Tricks, Questions & Best Practices

Tip

SymfonyCon 2018 Presentation by Ryan Weaver

With Webpack, your JavaScript & CSS code can have superpowers you've only dreamed up. And with Symfony's Webpack Encore, you can get all of this with almost zero setup time!

In this talk, we'll quickly learn the basics of Webpack Encore, then, turn to the lessons we've learned over the past year: answer popular questions and explore common problems people run into when moving to Encore. We'll also dive into a host of lesser-known best practices that you can follow to make sure your frontend coding is as streamlined as possible. A modern frontend build system: all in the time of one talk!

How many people have seen the SymfonyCasts videos? I was going to explain that this is really my voice and the miracles of editing. It's really incredible what we can with that audio. Okay. I am Ryan. Hello nice to meet you. I'm the the lead of the documentation team. Also, a writer for KnpUniversity... wait SymfonyCasts.

Hey Ryan!

Changing a domain is harder than you think. I'm basically, if you've been around Symfony for a while, you probably know me, I'm a Symfony evangelist / fanboy. The husband of the much more talented and much more popular at the Symfony conferences, Leanna. How many people have met Leanna? Okay, yeah, okay. That's so great, because that means, there's many of you have such a great opportunity today to meet Leanna, give her a jumping high-five, like this style.

Leanna, can you raise your hand, just so everyone sees, right here? Yeah, so Leanna's right here, and you can find her afterwards. She's going to come up afterwards and wait for her high-fives. And the father to my much more handsome, charming son Beckett. So now, I have a kid, so now I get to show you pictures of my kid in the beginning. Whether you want to or not, so that's my son, Beckett.

Webpack: Totally New Way or Frontend Dev

Okay, so let's talk about Webpack and Encore. We are going to do just a little bit of explanation in the beginning, then talk about some lessons that we've learned over the past year and a half. And also, some new features, because a new version of Encore came out, only one month ago, and there are actually some really exciting changes that we've made in that version.

Okay so first of all, Webpack itself. What is Webpack? It's a very simple concept, it allows you to package all of your JavaScript and all of your CSS into one file each. And it works via require statements, so you can have one file app.js, require some other files, sort of like we used to do in PHP, require some other files, and that's it. It's just going to turn all of those into one app.js, and one app.css file. That's kind of cool.

Or we can explain Webpack as a totally new way of programming. This is actually what makes it hard. You have to completely unlearn everything you've learned about doing frontend programming. Because we've spent 15 years learning how to program in JavaScript with global variables. That's insane.

How many of you regularly use global variables in PHP? There's one brave hand. No, we don't program that way. How many of you do it in JavaScript? Everyone that does JavaScript, right? Because it wasn't possible before. You have to unlearn that. We basically have to retrain ourselves to program in JavaScript the same way that we program in PHP. Once you do it, it's an amazing experience.

Encore Basics

So, Encore is just a wrapper around Webpack. It's actually a Webpack configuration generator. It's a Node library, but we recommend installing it via a Composer package. I'll explain why later. There is a little bit of PHP in Encore, but it's mostly Node. Of course, it has a nice recipe when you install it: composer require encore.

Oh, actually see, things changed. That's slightly out of date because we should not have the —dev there anymore. I'll explain why later. There is a small bit of code that you do want to run at run-time now.

composer require encore, okay, it downloads a few things. This is what it gives you. It gives you a package.json: that's the composer file basically for Node. Two files: a CSS file and a JavaScript file just to get you started. And that's basically it.

So, here is what the package.json looks like. It just requires two packages: Encore and the webpack-notifier is just cool because it makes cool desktop notifications when your build finishes. It's like ding, your build is done! It just makes it more fun. That's it.

The webpack.config.js file is where you configure Encore, how you tell Encore what you need. What you need is actually very simple. You just need to tell it where you want the output path, where you are going to put the final build files, and then you are going to point it, it's called an entry, you are going to point it to a single JavaScript file.

I'm saying go read that ./assets/js/app.js file. The first key app? That could be anything: that will ultimately control the name of your *output* file. If that first key, that first argument was foo, we would end up with a foo.js file.

But we're pointing it at app.js. What does app.js look like? Okay, here's a simple example. We can requires some JavaScript files, or we can even require CSS files. That's what we have here: we're actually requiring CSS from JavaScript. I'm going to talk about that in a little bit: that's very, very important. Then we actually need to install our dependencies. I really should have taken a screenshot of ... This is basically me running yarn install. Yarn is, there are two package managers in Node. Yarn is one of them. This is basically me running yarn install.

For some reason, yarn lets you be so incredibly lazy that you can just say, yarn and it installs for you. This is me running yarn install and the end result is... boom! We end up with tons of files in a node_modules/ directory, which is the vendor/ directory. At this point, we have basically just required two packages from Node, and installed them. That's it.

To actually run Encore, you're going to run yarn encore dev. There is also a yarn encore production you'll see later: that's the mode that minifies all of your files and does other optimizations. Of course, there is a --watch so you don't have to run every time. Every time you modify file, this is going to re-output your files.

The end result of this is that you get two files. Actually, you guys are probably good at counting, you get four files. The only important ones yet are the app.js and app.css. What happened was: we told the Webpack to look at our app.js file. It follows all of the require statements: all the JavaScript we required, all of the CSS we required, and got all of that JavaScript and CSS, and it finally writes one app.js, and one app.css file. Then we just need to include those in our template, completely like normal.

In some ways, I want this to seem very underwhelming, very boring. Okay, cool, you pointed at this file, and it outputs these files. Excellent. But this is massively important. We could stop the presentation right now, because this gives you the ability to have require statements in your JavaScript! Which means if you have a file and you want to maybe split it into two pieces, re-factor your code like we would do in PHP, before, we would maybe not do that. Because if you split your JavaScript into two files, you have to add another script tag, or you need to go into some build system and remember to say, oh, now concatenate this file too, make sure they are in the correct order, all that kind of thing. It just didn't make sense. It was difficult to split your files.

With Webpack, you can actually program correctly. You don't think about the packaging of your files, you just think about: if I'm in one file, and I'd like to use another file, I require it. Very much like, again, I don't mean this to make it sound like the require statement is old, but in PHP, very much like we used to do in PHP. If you needed something, you required that file instead of just expecting it to be globally available.

CSS as a Dependency of your JavaScript

I'm going to put up a couple of best practices guides through using Encore. The first one, as you've seen, is you need to think of your CSS as a dependency of your JavaScript app. The same way the, that if you were in a JavaScript file, and you need another JavaScript file, you add the require statement. That's a dependency. Do the exact same thing with CSS. You say, in this JavaScript file, in order for the page to truly work, I need this CSS file. So, you require it.

One Global Entry

Another best practice is, and this is exactly what we've seen so far, is that you should have one entry, that means one source JavaScript file that's going to live in your layout. It's not a rule, but this is a best practice. This is exactly what we're doing right now, we have an app.js file, and that app.js file is included in our base.html.twig. It contains all of the global JavaScript and global CSS that we need for our site. Later, we'll talk about page-specific JavaScript and CSS.

Refactoring with require

The game-changing feature is that require statement. The ability to have that require statement. Which by the way is called, as soon as you start using require statements in JavaScript, suddenly your JavaScript files are called modules. When you hear module, it's just basically a JavaScript file that works in this system. This is just a simple example of how I would maybe re-factor my code. You have the app.js on top, and then I'm just going to require another file and use its function.

The key thing is, in the second file, is the module.exports. In PHP, when you used require statements, you would just require a file, and just receive whatever is inside that file. That's not the case with JavaScript, or Node. You actually need to export something. You need to say, I want to export this function, or I want to export this object, or something like that. We're exporting that function, and then we require it to import it.

Notice there is no .js on the random word, and that's just because JavaScript people really like making you be lazy. You can put the .js on the filename, but you don't need to. That's why you see it not there.

require() versus import()

Or, to make things a little bit more complicated, interesting. You see the require statement a lot, but the require statement is not cool anymore. You're supposed to now use this import keyword, and this export keyword. The reason I'm telling you this is so that you know that they are exactly the same. They are just two syntaxes that do the exact same thing. The import and export is actually more of a standard, so that's the one you should use. It has a little bit more flexibility, but really, they do the same thing. I'll typically use import and export. It's all the same thing.

Module-based Architecture... just like PHP

The best-practice now is to basically program like you would PHP. To organize your code into modules. In Symfony, we would organize our code into service classes to be well-organized, you can even unit test them. It's the same thing in JavaScript now. It's like: okay, I have this function or this object with some useful functions on it, let's just put that in its own file so that it's organized better. We can reuse it, and then we can just require it from wherever we need it.

Here's a slightly more complex example. This is a file that's going to just call a function, but it's requiring this displayRandomWord. Okay, let's look at displayRandomWord. That's just another function. Sorry, that exports a function, but it imports another file, and that last file is actually another function which actually gives us the random word. Just an example of the type of thing we would do in PHP typically: isolate these things into small pieces.

Ah, but there's a problem. We've already talked about this. CSS dependencies. Actually, let me go back real quick. This works great, but now I'm going to re-factor my code so that the displayRandomWord() function is now going to output HTML, and that HTML is going to have some classes on it. And dang it, I need to bring in the CSS file to style those classes. What we could do, because, remember our app.js file requires app.css, we could just go into app.css and add our class there. That would make it into the final file.

But we can dobetter than that. A better way is to actually isolate this into its own CSS file and require it directly from that module. What I'm saying here is: you'll typically see your main JavaScript file, app.js, require a CSS file. That doesn't mean that okay, great, let's put all of our CSS into that one file. No, we can actually break things down. We can have some very, very deep module - you don't even know what page it's going to be used on - but if that one module needs some CSS, like this, then require the CSS file. Then, no matter who uses this, you don't even care what page it's going to be used on, the CSS for that module is going to be included in the final output CSS.

Again, it's programming correctly. That's all it is. You're actually defining all of your requirements and dependencies in each little file.

Each Module is an Isolated, Unique Snowflake

The way you need to think about it is that each module, each file, is its own unique snowflake. Its own unique environment. When you look at a file, guys, I'm going to keep saying this, it's just like we program in PHP. If you went into it a file in PHP, you would never make any assumptions about what variables are available to you, or anything like that. If you need something inside of a class in PHP, you would make sure that you add a use statement, or use dependency injection, or something. It's the same thing with these modules. If you need something in a module, require it. Don't think: oh, it's okay, that CSS has already been required by this other module, which will be on the same page. That's bad. Put everything you need inside that one file.

To make things actually even a little more complicated, this is so cool, I love this, now I've taken that same CSS file, which is required by one of my JavaScript files, and now I need to point to a font file. Okay, what happens when we do that? It just works. Because Webpack is going to notice that you're referring to that font file, it's going to move it into the build directory for

you, and the final output CSS is going to have the correct path to point to that font file. In fact, to Webpack, this looks like a require statement! It actually sees this, and it looks, to it, like you're requiring a font file. And so it says: oh, how do I handle font files? I move them into the build directory and make sure the path is set. The font file is a dependency of your CSS file.

In your CSS files you can use the @import syntax. In the top of some CSS file, you could say @import another CSS file. That's going to be read and parsed in exactly the same way. You could be in a CSS file and @import a SASS file, and that's going to go get processed just fine, and get processed through SASS.

Page-Specific JavaScript

Alright, so cool story. But, so far, we've only been talking about one file, one app.js file. More like we have one JavaScript file that we include on our entire site.

A lot of us still have multiple pages, so you have page=specific JavaScript. You have your main JavaScript and CSS, but let's say on your checkout page you have some pretty big JavaScript you don't want to put into your main JavaScript, because it's only needed on this one page. Okay, so now we're going to create a second file, checkout.js next to app.js. Same thing: we'll import the CSS we need. We'll import any JavaScript we need, write some code. This is just fake code, but you guys get the idea. Then we're going to go back into our Webpack config file. Remember, we haven't done much in this file yet. And we're going to add a second entry. An entry is an important concept in Webpack. Let's see here, and every entry results in one output JavaScript file and one output CSS file.

The result of this, since we added an entry called checkout, is that we end up with a checkout.js with all the JavaScript we need, and checkout.css with all the CSS we need. Then we just need to make sure that we put that on the checkout page. Inside of our checkout template, we're going to add a script tag for the one file, and a CSS link tag for the one CSS file.

The guidance on this is that, and of course, this is not an absolute rule, but if you're looking for general guidance, this is how I do it: is to treat each entry ... Well actually, let me say this a different way. Each page that needs its own JavaScript is going to get its own entry. It's going to mean, in practice, that every page is going to have the main entry, the app.js file that's in your layout, and if you need it, one other entry for that page-specific JavaScript. That entry file should contain all of the JavaScript, and all the CSS needed to power everything on your checkout page.

The other thing is, that each entry, this is how Webpack wants you to think about it, is its own standalone application. The Webpack, the authors of Webpack do a lot of single page apps, so they almost think of every JavaScript file as a single page app. You want to think of your checkout.js as its own app, which means that you need to, once again, require everything you need. If you need CSS require it, if you need some external library, require it. It's its own isolated environment. It's different than before when we think about: I'll include some script tags, and then these script tags will refer to some stuff that these script tags did. They're really two isolated environments.

jQuery: Things get Weird

Alright, so I want to talk a little bit about jQuery in case you're using it. Because jQuery is actually where things get weird. Actually, jQuery is also, because things get weird with jQuery, it's also a great example to understand how Webpack is doing some things internally.

Okay, so you want to use jQuery. This is one of the best things about the new way of developing. If you want jQuery, you just add it: yarn add jquery — dev. Just add it: just like we composer require stuff. We don't need to download, go download something. We don't need to find a CDN. You just add it. Then you just require it or import it. That's it.

This is really great. Outside libraries are just actually very, very easy to use. This is not the problem with jQuery. This would work just fine. Anytime you import a module that does not start with a ./, it's importing it from the node_modules directory. If you are importing a local file, you've noticed I'm probably using ./ before: that means a file next to me. If there's no ./, it's coming from the node_modules directory.

jQuery is just like any other module, or React, or Vue. These are just modules: and you just require them, and they work like normal. Here's a key thing though, what's the difference between importing $ from jQuery and a script tag that you put in your layout for jQuery? The answer is ... everything. Even if that is literally pointing to the same file! Because, inside jQuery, and this is common to many, many modules, this is not a jQuery-specific thing, inside jQuery, it *detects* how it's being used and changes its behavior.

In Webpack, there are no global variables. Okay, just like PHP you can cheat and make global variables. But if you're programming things correctly, there are no global variables. When we include a script tag on the page, what does that do? It gives us a global jQuery variable. If you require it, it detects that, and it uses the module.exports: it returns a value. It does *not* actually give us a global variable. The key thing here is that most JavaScript libraries are going to change their behavior based on how they're being imported. It's just something that you need to understand as you're going from the old way to the

new way of doing things.

A few minutes ago I talked about how you want your entry files, your application and your checkouts.js to function like two separate applications. This is interesting. If we import $ from 'jquery' in app.js, and its script tag is included first, and then in checkout.js we don't import it, but we just start using it, is that going to work? No. It's not going to work. Beautifully. That's going to be an "undefined variable $ in checkout.js": it's an uninitialized variable. In PHP, we would never expect that to work. It's not going to work. That is excellent.

This actually is little bit more interesting. Well actually, it's the same thing. Is this going to work? What I changed here is: okay, can I use the $ in maybe a template? I just need, hey, I'm getting lazy, and I need to put a script tag in my template. Is that going to work? Nope. Same thing, there's no global variable, so that's not going to work either.

One of the effects of using Webpack is that you will now have no JavaScript code in your template. Yes, we were always supposed to do that but we got busy, and lazy, and cheated etc. It's really not possible because you don't have any global variables. Now yes, you can cheat. In our tutorial on Encore we talk about that. When we upgraded SymfonyCasts to this, we actually set jQuery as a global variable, a true global variable, so that our legacy JavaScript would work. So that way, we didn't need to update everything all at once. Yes, you can work around this as you're updating your application.

jQuery Plugins: Weird gets Weirder

jQuery plug-ins are actually where things get truly weird. The weird thing about jQuery plug-ins, so bootstrap, I'm talking about the Bootstrap's JavaScript, it has some jQuery plug-ins in it, like tooltip.

The weird thing about jQuery plug-ins is that they don't return anything. They modify jQuery and *add* something to it. They work fine with Webpack, but they're clearly from a universe that existed before webpack. Because it's just weird. It's weird to just import bootstrap and suddenly we have a tooltip() function. It doesn't look obvious.

Internally, how does it do that? jQuery plug-ins, again, behave differently based on their environment. This is very important. If you think about the normal script tags, if you have a script tag for jQuery, then a script tag for bootstrap, how does bootstrap know where to find jQuery so that it can modify it? It only has one option, expect it to be a global variable. It just looks for jQuery as a global variable and it modifies it.

Fortunately, most modern libraries, most modern jQuery plug-ins now have code that looks like this. Where again, they detect their environment and they go: okay, I'm in a Webpack environment. What do they do? They actually require jQuery! One of the properties of the module system is that if two files require the same file, they get the same thing. It's a bit like Symfony's container. They'll get the same one instance of jQuery. It will actually modify jQuery.

Now, not all plug-ins are written correctly though. Here's a example. This is just a silly jQuery plug-in I found, tagsinput. The cool thing is I can say yarn add, so I can just add this to my project by saying yarn add jquery-tags-input. Great. I'm going to import the JavaScript file. That, in theory, will give me a tagsInput() function.

Also, this is a really cool thing. It's very common to have a JavaScript library that also has CSS with it. So normally we have to add one script tag and remember to go add the link tag. Now I just require the CSS file, and you're done. In this case, when you have the name of the module then / a path, it's just literally pointing at node_modules/that directory/ the path of the CSS file. Sometimes you have to do a little digging to be like, what's the path to the CSS file? Once you find the path, you just require it. Done.

This doesn't work. We did the exact same thing as we did a second ago with bootstrap, but it doesn't work. So you get jQuery is not defined. The reason is that, inside that module, it basically has code that looks like this. It just goes, jQuery. It just, hey, jQuery. It just expects it to be a global variable. And it's not anymore.

This is still written the old way. It has no code in it to make itself work in a module environment. It's broken. This is probably the most common and consistent question we get about Encore. It's not even an Encore thing, it's just a Webpack thing. It's important to understand how these jQuery plug-ins work, how they load, so that you can debug what's going on.

Ok so how do we fix that? Magic. We go back to our webpack config file, and we add one new line called .autoProvidejQuery(). This is incredible. And that function is an Encore feature, but anything in Encore, we're just leveraging features in Webpack. This is just leveraging a feature in Webpack. So what does this do? It rewrites the bad code. It literally, as it's going through all of the JavaScript files that you are working with, if it ever finds a jQuery variable that has not been initialized, it replaces it with require('jquery') and puts that in the output.

Yeah, right? It's incredible. This fixes it, and all of a sudden you can use these old things. Brilliant, we fixed somebody else's bad code! Be careful, because now we can write bad code too. This is now possible, so this is the downside of .autoProvidejQuery(). So, don't turn it on if you don't need it. It now means you can go into any file and just start saying $, or jQuery. You can instantly start programming the old way with undefined variables. And Yay, Encore will also fix our bad

code. If you turn it on, try to not do this, even though you can.

This is important, if I'm in a Twig template, and I put an inline script tag, that still does not work. Because .autoProvidejQuery() does not give you a global jQuery variable. It rewrites any code that's processed by Webpack. If you just have some random Twig template, there is still no $ variable. This is just a repeat from earlier, even if you can cheat with the autoProvidejQuery() plug-in, remember that each file is its own environment, so remember to require it.

Entry != script Tag (in the old way of thinking)

This is another thing, so a couple of lessons from the past year that we've seen people do. How many people have had applications that look like this? Yeah, yeah, me too, yes. Because this is how we used to program. Require shared, and that creates these five global variables, and these 10 global functions, and then vendor.

Okay, now we have a jQuery global variable. We just build on top of each other by adding more and more global variables. Then we have some script on the bottom that actually uses these. I have seen people basically take their old setup and just move it directly into Webpack. I understand, every file that I have in my old application is just an entry file, right? Because this is going to basically result in the same output file. Yeah, first one requires a file, vendor is going to go actually require juery and react. Okay, good, this works, right? This does not work at all. Because what happens with that vendor entry, that vendor entry is going to require jquery. Good for you, you just basically gave yourself a jQuery variable and never used it.

It's like calling a function, setting a variable, and then going to lunch. Because that doesn't set a global variable. That's why I don't follow this model, I follow the model of one entry per page. Think about your checkout page as its own JavaScript application. What is all the code that needs to go into this to build this JavaScript application?

addStyleEntry(): It's a hack

Another thing, another feature that we've had forever in Encore is the addStyleEntry(). That's a way for you to say, oh, I want an output file, but it's just CSS. I just need a CSS file. That's a hack. We added that as a way to make people more comfortable, but you shouldn't use it. By shouldn't, I mean it's not like a bug, buggy or something, it's just that if you're using that, you're probably not organizing your code correctly. Because your CSS just shouldn't be just some random, hey, I just need a CSS file. You should be thinking what: JavaScript file is this CSS a dependency of?

In our app.js, we require the app.css file: this is the CSS that's needed for our layout, so we think of it as a dependency of the JavaScript file that's in our layout.

Splitting Files / New splitEntryChunks()

Okay, now this is where things get really interesting and we're going to talk about some of the new features of Encore. I realize I'm running a little low on time, but I also realize there is a break afterwards. So sorry if I keep you for a few extra minutes.

Alright, so Webpack says:

> hey, it's cool, just require stuff, I'll handle all the details, don't think about it. Really, it's like don't think about it. You just write code correctly, I'll take care of the rest. I'll build your files perfectly.

Great Webpack, except, wow, those are pretty big files, those JavaScript files, yeah, 168 kilobytes. This is the development mode, so it gets smaller in production. But jeez, those are big. Because... we have so many jQueryies. Because we required jQuery from our app, and we require jQuery from our checkout, so what did webpack do? It put jQuery in both. Perfect! Except for performance on your front-end. How do we fix that?

This is not the way we fix it anymore. I wanted to show you this because you'll see this. This is the way that we fixed this problem for the first year and half in Encore. Because this is the way that you fixed it in Webpack. Again, we're always leveraging Webpack features. Webpack 4 came out earlier this year and changed a lot of things. This is one of the features that is still available in Encore, but is not the recommended way to do it. Yeah, I'm not even going to talk about it, but this is a feature that existed before. It still exists, but don't use it.

Now what? What we do now? I said don't use that feature. Okay, so introducing Webpack Encore 0.21.0. Wow, what a great version number for a big release! Wow, congratulations! This brought a lot of new stuff. Webpack 4 support. I'm going to talk about a few of these. I'll show you a couple of these, so I'm not going to mention all of them here. browserslist is a really good one. That allows you to put in your package.json file which browsers you need to support, and different parts of Webpack will know how to rewrite your code to be compatible with exactly those browsers. So, if you're using newer JavaScript features, or you need some vendor prefixing on some new CSS stuff, then it will automatically do that for you. Really, really cool. Webpack is always, and Encore has always done that rewriting, but it was more difficult before to control the exact browsers

that you need for that. Yeah. The rest is not important, or we're going to talk about.

The Runtime Chunk

Real quick, this is not that exciting, but I want you to understand what it does. There's a thing called a runtime chunk. By the way, "chunk" is what Webpack calls a JavaScript file more or less. When you see chunk, that's kind of what it means.

This is a function that you can call now in your Webpack config file. Okay, cool, what does that do? It actually results in one additional file being output. A runtime.js. It now means that you actually need two script tags when you enable this. Why do we do that? There's two reasons. One is actually a little bit for performance. I don't want to get into the weeds too much. The runtime.js file contains some code that helps Webpack work. That code tends to change more often than your code, so by isolating it into its own file, your other files will change less often, and therefore can be cached longer by your users. It also has a side effect that when you have the runtime chunk, if you required jQuery from somewhere in app.js, and somewhere in checkout.js, they get the same object. Whereas, if you don't have that, your entries are so isolated that when you require jQuery, it's actually two separate objects. That may or may not be important to you.

Oh yeah, I actually have an example of that. Will this work? I require bootstrap in app.js. Down in checkout.js, is there a tooltip function? We required bootstrap further up, right? Right? It depends. With enableSingleRuntimeChunk(), yes, because they're actually using the same jQuery object. With disableSingleRuntimeChunk(), no, they would be more isolated.

It's not... what's wrong... enable is more convenient. Disable is actually a little more pure, because I said keep them isolated. It's up to you, I just want you to be aware of the decision there.

splitEntryChunks()

Okay, cool. Back to optimizing our build. We're not going to do this createSharedEntry() thing anymore. We're going to go back to just, we have 2 entry files, or maybe 10 entry files. Cool. Now we're going to say .splitEntryChunks(). This is all you need to do to optimize your build. You don't need to think about anything. Just call that function.

Okay, what does that do? That, when you run Encore now, you're going to get a little bit more output. And, if, you've probably never seen this before, but notice it says we have an entry point called app, and we have an entry point called checkout. Then it lists a bunch of files after each one. You'll notice after app, if you look, there's actually one, two, three, four JavaScript files listed after it. In order to get the app entry point to function, you now need four script tags. I know, yeah, I just got some looks like hmm?

What happens with splitEntryChunks(), is when you enable that, you ask Webpack to look at all of the files you're requiring across all your entry points, identify code that you're duplicating between the two, and try to figure out the most optimum way to split that into multiple JavaScript files.

It's really incredible how it does this. It takes into account your file size. If you have some shared code, but it's maybe like five kilobytes, it's not worth the extra web request to isolate that into a separate file. You can configure all this, so I'm just telling you the defaults. If it's above 30 kilobytes, I think that's the threshold that it decides that this is better in a separate file. It also distinguishes between your code and vendor code, things in node_modules, because it assumes that vendor code will change less often, so it tries to isolate the vendor code into its own file. Because that will change less often. Your users will, by isolating it, your users will be able to cache that longer because it's not going to change all the time.

It's actually a pretty incredible algorithm it's using to figure out the optimum way to do this. It also recognizes that your browser will make, it's either three or four parallel requests to the same host at the same time. It will split it to a certain point, but it realizes that if it keeps splitting it, then your browser's actually going to be waiting to download the first few JavaScript files before it starts the other JavaScript files. Yeah, it looks kind of crazy at first, but it actually works really, really well.

Of course, the problem is how do we know what script tags to include? Also, including four script tags, that seems like a real pain. Also, as you can tell, these are going to change. If you start doing something different, it's going to split it in a different way.

So, introducing the newest bundle in the Symfony family, WebpackEncoreBundle. Amazing! Amazing! Wow. Probably the smallest bundle in all of the Symfony ecosystem. It gives you two Twig functions and that's it. It's super cool. Now you just say, encore_entry_link_tags(), you give it the name of your entry, and it's just going to include all of the link tags, because the CSS is also split. Then same thing with script tags. It will include all of the script tags you need for that. When they change, will just get output.

How does that work? Magic? No. It's actually really boring. Oh, and same thing: checkout, include the checkout. It works because Encore dumps a file called entrypoints.json that looks like this. Just dumps exactly which files are needed for every entry point. That bundle just reads those and it works. This means that you just enable .splitEntryChunks() and you don't

care. Things get split into pieces. It re-optimizes itself as you do different, keep coding and make more builds. Your code just works.

100% Cache Busting / Versioning

Also, I don't talk about here, but if you want versioning on your assets, so like when you build they have hashes in their filename, you get that out of the box. In the Encore recipe, we have a line in Webpack config that says enableVersioning(). All of your files output with a hash in the filename. And you don't even know or care, because those hashes end up in this file. And so as long as you use the Twig functions, then you get free versioning. If you change a file, then it's going to change the hash in the filename. You don't even think about versioning anymore. You just get it for free.

Long-Term Expiration Caching

That also means if you know a little bit about it, you can enable long-term expires headers for your assets. You can configure your web server to say, for all of my JavaScript and all of my CSS files, really, anything in this build directory, let the user cache that for a full year. Because I don't care. If the content changes, we'll change the filename. That can really massively speed up your frontend performance.

Dynamic / Async Code Splitting

One last thing I want to talk about, this is another feature that's not technically new with the new version of Encore, but we made it work without any configuration, is dynamic code splitting. It looks like this. Normally, we have the import function, the import call on top, the top of our file, we import all the things we need. Sometimes you have a situation where you have a large piece of code, and you only need that code under certain situations.

The perfect example is if you have a React or Vue router, and it's like when you go to this URL, render this component, go to this URL, render that component, if you go to this URL, render that component. If you just code that, normally, that means all of the code from all of your components will get packaged into one single file. Maybe not that big of a deal. It depends on how big your application is, how much you care. Instead, what you can do is you can use import like a function. It basically looks like an AJAX call, because it is an AJAX call. Import like a function, and then you give it the name of the module that you want to import, and then you say .then(), that's a promise, .then() and you pass it a callback. The argument passed your callback is that module. And then you use it.

In this case, since we're importing external linker, that code will not be included in the app.js file. Instead, the moment we need it, it's going to grab it via AJAX, and then when it finishes, it's going to go execute it. It also includes the CSS file, because our JavaScript down there required CSS. We don't care. It's just going to work.

In Conclusion

All right, so now putting all together, the things I want you guys to take away from this, the best practices are: create one entry file in your layout that's going to contain all your global JavaScript and CSS. Treat each entry you have like an individual application, my checkout application, my product list application. Organize your code. We can do that now. Finally. Finally we can actually organize our code properly into pieces. Remember to treat each module like its own unique environment. Require everything you need just from the very specific place you need it. Require the CSS you need from that tiny little module that actually needs it. Like I said, make your CSS a dependency of your application.

Okay, and I forgot this one: require jQuery like anything else, even if you are allowed to cheat because you use autoProvidejQuery(). And, use the new splitEntryChunks(), because that's going to make your life much easier. By the way, I showed most of the presentation, I showed using normal script tags and normal link tags. Then we re-factored to the bundle. In reality, you just use the bundle. When you use Encore, you use those functions, then you don't have to think about versioning, or splitEntryChunks(), or anything. It makes sense to turn that on.

All right, thank you, guys! Thank you for letting me go a little longer. Questions? I think I get to throw this. Cool. I was like, there has to be at least one question, because I haven't been able to throw this yet.

Yeah, that was amazing. That was on me. That was a bad throw. That was a good job over there.

Why are you wearing no shoes?

Why am I wearing no shoes? Well, now it's like my thing. A long time ago when I was presenting, I had shoes with heels, and I got on the stage, and it was like clunk, clunk, clunk, clunk. So, now I wear no shoes so I can be silent and slide around the stage. If it's summer, I wear no socks also.

Question over there? Or was it the same question?

Okay, my question is I'm not familiar with JavaScript, and I was wondering why you add —dev everywhere.

Yeah, yeah, that's a really good question. Why when I'm doing the yarn add, like the composer require, why was I adding —dev? It doesn't matter. Technically, it's just a technical detail. Technically, if you use Node on your application for Webpack, but you also actually had a little bit of Node that you actually ran in production. I don't know, your code actually went through and used some Node code, then in theory, by adding Encore and those things as Dev dependencies, when you're deploying, you would actually, that's, you would run Encore, and your dev dependencies would be there, and you'd get that final built public/build file. But then when you finally got to production, and you needed to install your Node dependencies for whatever weird thing you're doing at runtime, you could actually ask yarn to only install your not dev dependencies, which would be whatever libraries you're using for your actual runtime code. But you don't actually need Encore on production. It's technically correct as a dev dependency because that's not something that you actually need on production to execute your code. In reality, it doesn't matter because none of us, almost none of us are going to want to go to production and install our not Dev dependencies of Node because we have Node running alongside our PHP application. It's a weird edge case. Yeah, good question.

Nice, oh good, you guys are responsible for throwing it now.

Hi, thank you for presentation. I actually have two questions. First you said that require and import are equally the same, but they're not. You showed that there is asynchronous fetching in import. It's a big difference between the require also model splitting you can include only this part you need. The second question is how this jQuery fixing tool affects building time.

Interesting. Yeah, first part, yeah. You guys saw, I said require and import were basically the same, but import is a little bit better, but I didn't really say why. One of the reasons, as you said, is the import has this asynchronous, the import function thing. You can actually do that with require also, but the import is the cooler way to do it. The other thing which we didn't talk about is import and export, you have the ability to export multiple things from a module. With the require and the module.exports, it's always module.exports equals, the one thing that you want to export. But there's technically, with the export syntax you can export like 10 functions from the same module. Sorry, the other question was?

About jQuery fixing pack, or making them modules.

Oh yeah, the jQuery fixing thing. I don't think the jQuery fixing thing affects performance, or at least affects performance very much. The reason is that Webpack is basically already doing all that work to go through and parse every file and look for the require stuff. We also go through a tool called Babel, which actually reads your JavaScript code, and then outputs old JavaScript.

And changes imports to require, yeah.

Yeah, I don't think it affects much. Your builds can get big. It has to do with the size of your project. If you're doing things like... this alone won't make your builds slow, but if you were doing, if you were using SASS for CSS, and you wanted to import bootstrap as SASS, that alone is not going to make your build slow. But, if you start doing lots of those types of things, all of a sudden you're saying, hey SASS, go parse this giant SASS file. That's the kind of stuff that's going to make your build slower.

Thank you.

It's good you guys all sat together with the questions. It's very convenient.

I'm not into JavaScript very much, don't know very much information about it. In my company, we already moved to Webpack a couple of weeks ago, but I don't know why every time I get a new branch, without accessing or editing any JavaScript or CSS files, I can see the package.json already changes. I don't know why.

The package.json changes?

Yeah.

Yeah, I don't know about that. As he changes branches, the package.json file is suddenly modified. I'm just going to say that in case somebody else knows what weird thing that happens, but that might be something specific to your project.

Okay.

Oh good, he can tell. Good.

Okay, so basically depends on the environment, so if some of your team, then you can have some differences in output. You can find a stackoverflow on basically how to fix it. You can disable it.

You better fix it in your project. So then you have no changes between operational systems. In our company we have fixed it recently, and it's pretty easy, you can just find it on stackoverflow.

Yep. All right, think we're past time. If you guys have more questions, come up and say hi. Thank you, guys.

Chapter 20: My first year with event sourcing (in Symfony)

Tip

SymfonyCon 2018 Presentation by Tim Huijzers

Over the last couple of years, I have heard of Event sourcing but didn't really know where to start until I did a tutorial at DPC '17. After having some basic information it was time to start a Hackathon and after that something production worty. In this talk I will try to give the best information to get started and to know some of the problems you can face if you begin event-sourcing.

Chapter 21: Symfony 4 Internals

Tip

SymfonyCon 2018 Presentation by Tobias Nyholm

Symfony is a request and response framework. But what about all that magic that happens around your code? Why isn't autowiring slowing things down? And how is it that Symfony components can be so decoupled but sill play so well together?

I will show you the Symfony internals and its architecture.

This talk will go over the architecture of Symfony. We will follow the request and the response paths throw the framework. We will do some stops at the components that are more awesome than others.

This talk is perfect for you who is working with Symfony or Laravel and want to understand what the framework actually is doing for you.

Chapter 22: What’s a Typical Symfony 4.2 Application Like?

Tip

SymfonyCon 2018 Presentation by Nicolas Grekas

Symfony's configuration system has improved a lot in recent years. Version 4.2 brings a lot of minor improvements that all together give an even better developer experience. Let's step back and look at how we can tackle all of the most common problems using the new approaches! This will include tricks for configuring services as easily as possible, handling different environment configuration and the hugely important topic of environment variables and secrets. Best of all, we'll discuss some strategies to migrate your existing apps so you're never going to be left behind.

Chapter 23: Modern Application Built from Scratch: API Platform FTW

Tip

SymfonyCon 2018 Presentation by Anderson Casimiro

Scenario: You had an awesome idea. A brand new business. So you have to test it‘s adherence to market… as soon as possible. How?

Fortunately we have Symfony and API Platform. This session will cover main topics to build a real world project on top of REST and GraphQL APIs with modern authentication while delivering clients and documentation. Barely touching PHP code for that.

Chapter 24: Integrate (Vue)JS Components in a Symfony App, add E2E Tests with Panther

Tip

SymfonyCon 2018 Presentation by Kévin Dunglas

Thanks to the new capabilities of the web platform (web components, Progressive Web Apps…) and the rise of modern JS libraries (Vue, React, Angular) almost all modern Symfony applications must leverage the frontend ecosystem. Symfony 4 embed many gems that make it easy to integrate modern JavaScript within the framework, including the first component entirely written in JS: Webpack Encore. In Symfony 4.2, another component that is super convenient for apps containing JS code has been released: Panther, a PHP library compatible with BrowserKit, that drives real web browsers to create end-to-end (E2E) tests with ease. During this talk, I'll show you how to cleanly integrate modern JavaScript code with Symfony and Twig and how to test such applications using Panther. The examples will use VueJS, because it’s probably the easiest JS framework to get started with as a PHP developer, but all the tips and tricks will be applicable with other libraries such as React or Angular.

Chapter 25: Profiling your Symfony Application

Tip

SymfonyCon 2018 Presentation by Michael Cullum

A 1 second delay on an e-commerce site can result in a 7% reduction in your conversion rate; but profiling isn't just important in e-commerce when limited resources are available. Profiling is really important and there are now a host of tools to help you profile your application through means other than naive profiling. In this talk we'll look at how to identify bottlenecks in your application using a variety of tools; and how we can use those profiler results to improve our Symfony applications' performance.

Chapter 26: A Year of Symfony

Tip

SymfonyCon 2018 Presentation by Sarah Khalil & Nicolas Grekas

Let's review what happened during the last year: basically we'll see and/or discover nice new features that appeared since the last year.

Chapter 27: Magic

Tip

SymfonyCon 2018 Presentation of magic, prepare to be amazed!