Title: **Rapid prototyping with the help of Docker**

Summary:

Imagine you have a web application and you want anyone in the company, or a 3rd party contractor, to be able to quickly prototype new view layouts or different UI frameworks. You want them to be able to get up and running quickly. You don't want them cloning the source code, setting up databases, working with configuration files, and so on. In this talk, you'll hear how Docker is being used by Dovico to speed up development of Timesheet.

Level: **Beginner**

Duration needed: **35 minutes for the talk**, 10 minutes for Q&A

Presentation will be on: **Thursday, February 23rd at 3pm**

**Setting up for the talk:**

* Be **logged into Dashlane** for access to Teamset Beta creds
* Be logged into the demo Timesheet database
* Have Dovico.com up ***(default tab)*** in the same tab grouping as the logged in Timesheet
* Make sure the docker images are running and each open in a tab:
* 5000 – menu config ***(default tab)***
* 5001 – Blazor
* 5002 – freemium migration ***🡨 Have the local storage cleared***
* Make sure the quick assign container (port 5003) doesn’t exist
* Have the Blazor code solution open so I can show the base tag
* Have the Simulator code solution open (D:\ConFoo\_Docker)
* Have a terminal open to D:\ConFoo\_Docker\
* Have my notes Printed and on the laptop/my phone as a backup
* **Check things off as I go** to make sure I don’t miss anything
* Have a **countdown timer** from 35 and leave the phone on so that I can see where I’m at
* Have **water** and take a sip every once in a while.

**The Talk itself…**

* While in the PowerPoint
  + **Welcome slide**

Good morning everyone!

Welcome to my talk on **Rapid Prototyping with the help of Docker**!

Shortly into the presentation I’ll be talking about Docker.

Just in case it’s needed, I created a slide that will give a quick overview of Docker if there are people here today that don’t know what it is or how it works. I’ll skip the slide if everyone’s familiar with it.

Don’t be embarrassed if you don’t know. That’s the whole reason why we’re here at a developer conference… to learn.

Maybe a show of hands… Does anyone *not* know what Docker is? Would anyone want me to give a quick overview of Docker when we reach that slide?

Alright… thank you

***[next slide]***

* + **About me slide**

My name is Gerard Gallant.

I work at Dovico Software and currently wear a few hats.

CIO is a role that I’m expanding into but I’m also a senior software developer and architect.

Writing a book was a bucket list item so I’m pretty happy that I was given the opportunity to write a book with Manning Publications called “WebAssembly in Action”.

If you want to reach me, you can find me on Twitter or LinkedIn.

***[next slide]***

* + **Outline**

In this talk, I’m going to discuss the challenges that we faced when we looked into modernizing one of our products.

I’ll start with a quick overview of how our product was structured so that you understand why a solution like Docker was needed.

***[click]***

Then we’ll look at how we’re currently using Docker

***[click]***

And, finally, we’ll look at how we might leverage containers even more going forward

***[next slide]***

* + **Where we were**

Looking at where we were…

Our main product, Dovico Timesheet, is a monolith. It was started back in the day using Active Server Pages and then we jumped to ASP.NET when that came out.

There are a variety of different technologies that have been used over the years so it takes time for new developers to get up to speed and it takes time to update the software in general.

If we just wanted to try something, the software needs to be rebuilt, tested, and deployed. That process isn’t quick.

We needed a better solution.

* + **Creation of Modules**

I started looking at our architecture to see what we could do…

***[click to show api]***

We had built an API so that apps and customer integrations would be possible but the API was created about 10 years after Timesheet was started so Timesheet itself doesn’t use it.

While looking at the architecture, an idea occurred to me…

***[click to show module link]***

* What if we allowed for client-side only views that could talk to the local API?
  + Having it client-side only removes the potential security issues of code accessing files, memory, or session state data on the server that it shouldn’t
  + The client-side code would only be things like JavaScript, CSS, HTML, images, and WebAssembly
* The API calls themselves would be the same as what an app or integration would use but the module would talk to a Timesheet endpoint instead. The Timesheet endpoint would just relay the call to the local API instance.
  + This simplifies things for the module developer and adds security for users because the module doesn’t know about the user’s credentials. The Timesheet endpoint will specify them before passing the call along.
  + This is also secure because, even if the module tries to access certain data, all API calls are restricted based on the logged in user’s permissions. The Timesheet endpoint is passing the local API the logged in user’s token so only information that person has access to will be returned.
* Finally, instead of a config file that forces all companies to have the same menu system, I created a table in the database for the views. The table indicates where the main file of the module is.

This is great!

All we need to do is add a module’s files to a folder under the solution and add a record to the database table and presto! A new view lights up without needing to recompile or deploy Timesheet!

Now we can test new ideas for features, try different UI frameworks like Bootstrap or React, and even test module ideas with specific customers by only turning on that view for them.

It’s a step forward but the issue that we now faced is that a developer version of this setup takes quite a bit of effort to configure ***before*** you can even start working on a module…

* You need to install a database engine, or pull a docker image, and then restore 2 databases
* You need to pull and compile the source code for 3 different solutions
* You need to set up 3 IIS applications
  + - There are 4 shown here but the API Proxy isn’t used by modules because Timesheet it talking directly to the local API
* Some configurations are needed in the databases
* And each app’s config files need to be adjusted so that everything talks together properly

If you’re fortunate, everything works on your first try and you don’t need to spend a bunch more time trying to figure out which configuration you forgot or messed up.

This is a lot of work to set it up even for one of our experienced developers never mind a summer student.

It’s also a showstopper if we wanted to open this up to third parties because we wouldn’t want to hand over our source code.

So…

How can we make the setup easier so that we can just start working on a module and how can we allow 3rd party development without giving them our code?

* + **What is Docker? *🡨 skip this slide if everyone’s up to speed on what Docker is***

Before I start talking about Docker, I’ll give you a quick overview of what it is.

Docker is a platform that lets you build, deploy, run, and manage containers.

A container is a package that contains an application’s code and all of its dependencies, so that it can run quickly and reliably from one computing environment to the next.

Like a virtual machine (VM), containers are isolated and act as if they have their own file system, CPU, and RAM. Unlike a VM, however, they don’t have an OS. Instead, a container shares the kernel of the host OS.

Not having an OS reduces the container’s size. It also allows a container to start faster and be much more efficient because there are fewer system resources needed to run it. Because of the smaller size and fewer resources compared to a VM, additional instances of the application can be run with the same hardware.

* + **Can Docker help?**

We had started to use Docker internally for development needs.

For example, I no longer install a database engine on my laptop. I just pull a Docker image with the database engine version I need.

Rather than passing around developer databases, I set up a docker image with some dev databases that are preconfigured to make getting started with building our solutions easier.

Could docker help us here?

***[click to show the Windows container]***

Based on our software stack, we would need a Windows Docker image because Timesheet uses the ASP.NET Framework which needs IIS and both technologies are Windows-specific.

It’s not the end of the world because this is only intended for developers but Windows images tend to be quite a bit bigger than Linux images.

Unfortunately, I discovered that MS discontinued support for the Windows Docker Image of SQL Server and only Linux is supported now.

***[click to show the Linux container]***

This throws a wrench into my plans because you can’t have Windows and Linux containers running simultaneously on the same machine.

This could be a multi-OS setup ***[click to show the OS labels]*** but that’s still complicated and not every developer is going to have a second machine to network with especially if they’re working from home in this new remote/hybrid world.

At the same time, it’s not that hard to copy files out of an image. Down the road I’d like to open this up to 3rd parties so that they can create modules too. What if they realize a full version of Timesheet is in the Docker image? They could copy it out, set it up on their own servers without us knowing and without paying us for it.

These are some pretty big hurdles to overcome and I wasn’t sure if containers could help after all

As I was thinking about this though, something dawned on me

***[click to next slide to show the current architecture]***

* + **Timesheet Simulator**

Really… we only need a way to develop a module so that it works exactly the same way on the developer’s machine that it will in production.

We don’t really need the full version of Timesheet with the databases and everything else in the docker container.

What if we had a Timesheet Simulator that just talks to the normal API endpoint but the module interacts with it the same as it would with Timesheet in production?

***[click to show the simulator and its interaction with the api proxy]***

With the simulator, you create enough of the Timesheet UI so that you can see how your module will look in Timesheet and enough of the setup that you don’t need two sets of JavaScript (one for development and the other for prod where you only find out if you messed up an endpoint or something once deployed)

The simulator approach is what I decided to look into.

***[Switch to the Dovico Website]***

On Dovico’s website, **Dovico.com**, You can sign up for a 30-day free trial.

When you log in, you’ll find the logged in user’s **API token** in the Settings view *(hover over the button on the top-right for the menu)*, Data access token field at the bottom. Copy that to notepad.

You’ll also need the consumer secret which identifies this specific database. You can find that token in the Company view, API tab, and Data access token field. Copy that to notepad as well.

Now you have what you need to start using the Timesheet API.

As I’ve been moving around in Timesheet, you’ve probably noticed that the nav bar stays the same and it’s just the section below it that changes as you navigate between views. When you mouse over the menu bar, menu items are shown.

Before we dig into the code and start using the tokens that we’ve copied, let’s take a look at a some modules via the simulator.

***[move Timesheet to be side-by-side with the modules for comparison – Menu Config the tab selected]***

The simulator has fewer menu items but, if you notice the address bar, this is running locally yet there’s data being loaded.

**Show the API Info view** and then switch back to the main view

This module was developed using the simulator and went live in Timesheet in November allowing administrators to configure their menu system by turning off certain views or just hiding them by default.

Just to show what’s possible, so long as the code runs client-side, it should run as a module. ***[switch to the Blazor module]*** This example is a Blazor WebAssembly app that I ported from an Java app I wrote years ago. It uses C# code and draws to an HTML canvas showing the status of an employee’s time entries over the past 30 days.

***[switch to the freemium migration module]*** Timesheet had a free option for a bit and then it was decided to stop that approach. Another product we’re working on ‘Teamset’ has some similar features as Timesheet and has a free version so we offered those on Timesheet the option to be migrated if they wanted. I was originally going to make this available as a module so that Sales/Support could migrate those who wanted to go to Teamset. There were only about 10 who wanted to migrate so it was decided that I’d be the one to do the migration. I didn’t bother fixing up the UI and didn’t even deploy this to the servers. Instead, I adjusted the manifest file to point to the Timesheet customer’s database and I ran the module locally.

I’ll be showing you the Simulator’s code in a moment but the reason why I’m showing you this module is because it demonstrates that the simulator (and Timesheet) can pass a query string received to the module.

Let’s take a look at the code…

***[switch to the simulator’s code]***

For the simulator, I created a solution from the **ASP.NET Core Web App** template and I chose to **enable Docker** but that’s optional. You can build the Docker image as a separate manual step if you want.

I made a few tweaks to the **Program.cs** file:

* The first change is that I added some mappings so that Kestrel will allow Blazor files to be served.
* I added some code to read in the contents of a manifest file
* And, finally, I added in a route map so the APIRequest endpoint can be called.

Middelware\**ModuleInfo.cs** has a class that accepts the deserialized JSON data and that data is added to the HttpContext of the current call for use by other areas of the app.

Under wwwroot, I added a modules folder that has a few folders under it.

* ApiInfo is a view that gives some basic information to the developer on how to work with modules
* ManifestNotFound is the view shown if the manifest file isn’t found
* UserModule is the folder that you map your local output folder to in the docker image

Under Pages, **Index.cshtml.cs**, I adjusted the code behind file ***[show code behind first]***

I updated the Get function to check if the HttpContext object has information from the manifest file and some default values are used if not.

Regardless of if the manifest file was there, the menu will still have two items so the developer can toggle between them which is why the query string check happens.

The query string that is used to switch views is stripped out but, if there’s still a query string present, that is passed to the iFrame. There may be times where it’s necessary for an external source to pass information back to Timesheet in the URI. With this step, that query string is passed to the module.

We won’t switch to the user module view unless the manifest file was loaded so we grab what the developer specified for a start page because it might not be index.html.

Finally, we add in some data for the view to use when generating the HTML ***[switch to the Index.cshtml to show them]***

Show the index.cshtml file

Under **Pages\Shared**, I modified the **\_Layout.cshtml** to have a nav bar that look like Timesheet’s menu system and the css in **site.css** was adjusted as well.

The bulk of the work in the simulator happens in the **APIRequest.cs** controller where it takes the module’s request, adds in the tokens from the manifest file, and relays that on to the main API.

The bulk of the simulator’s work is the APIRequest endpoint that receives calls from the module and relays them to the API.

There are several routing attributes defined for the different API levels that are possible.

We grab the full path making sure it ends in a forward slash so that the next lines of code don’t break.

We then split the URI at the APIRequest/ portion because everything before that point is the Timesheet API and everything after that point is needed for the API call.

We then check to see if the caller uri starts with <http://localhost:5000> for example and if not, we add that to the beginning of the caller URI.

We build up the request using the same GET, PUT, POST, or DELETE method that was used to call this endpoint and point the request to the public API.

Add in the Accept header and application/json value if it was specified by the caller. The Timesheet API was built when JSON was just becoming a standard but XML was widely used at that point so, if you don’t specify an Accept header, XML is the default that’s used.

We need to do a big of a hack here. With the actual Timesheet endpoint, it’s able to talk directly to the local API and specify the root uri to be placed in the return data. We don’t have that option with the public API so the public api root will need to be replaced with the caller’s root uri once the data is returned. It’s not usually noticeable but this can cause the simulator to respond a bit slower than Timesheet would.

If the caller made POST or PUT call, copy in the body and the content type if one was specified.

Fire off the request, copy the response back to the response that will be given the caller.

Grab the body that was returned, replace the root uris and then write the body to the caller’s response.

***[Show building the image]***

**docker build -t timesheetmodulesimulator:latest .**

I usually use **docker ps -a** to show all containers that have been created on my system, including containers not running, but you can also filter the list to just those that derived from a specific image. In this case timesheet module simulator latest:

**docker ps -a --filter="ancestor=timesheetmodulesimulator:latest"**

***[Show building the Quick Assign container]***

**docker run --name module\_quick\_assign -p 5003:80 -v "D:\Projects\TimesheetV14\modules\module-quick-assign:/app/wwwroot/modules/UserModule" -d timesheetmodulesimulator:latest**

***[show the simulator running it and talk about it a bit]***

**http://localhost:5003**

This Quick Assign module is one I’m building for Timesheet. I built it using Bootstrap 5 and wired it up pretty quick. I’m not a designer so it took me more time to make it look like the other views than it did to create it. Unfortunately, this view needed some changes with Timesheet itself. The changes are done and are currently being tested. As I mentioned changes to Timesheet take time but we’re hoping to push out a build in mid-March and then this view can go live.

If everything was already set up in Timesheet, we could have released this after a couple days of testing.

That’s the simulator and where we’re currently at with Docker’s use.

Going forward, I have some thoughts but nothing’s set in stone yet…

* + **Discus what’s next**

Right now with the module system, only client-side code is allowed and all modules are available to every Timesheet customer if we add the record to their database.

We can control what modules are seen per customer database but the files are currently all in the same spot on the server. If your company writes an integration, they might not want that on the server where other companies might be able to access it if someone turned it on in their database by accident.

They wouldn’t see your data because the employee token is used and that only shows them their own company’s data but the idea that someone might be using something you built might not be desired.

This would have to be prototyped first to make sure it’s possible but I’m thinking that a container system might solve this where each company could get a container.

Develop the module locally using the simulator and then upload the files which get dropped into a container.

Not only would your modules be isolated from other companies’ modules but it opens other possibilities like having Node.js, .NET Core, or other runtimes available for your server-side processing. Maybe your module could talk to another API and this would allow you to keep your secrets out of the browser.

* + **Questions**

Anyone have any questions?

* + **Show the slide with the links**