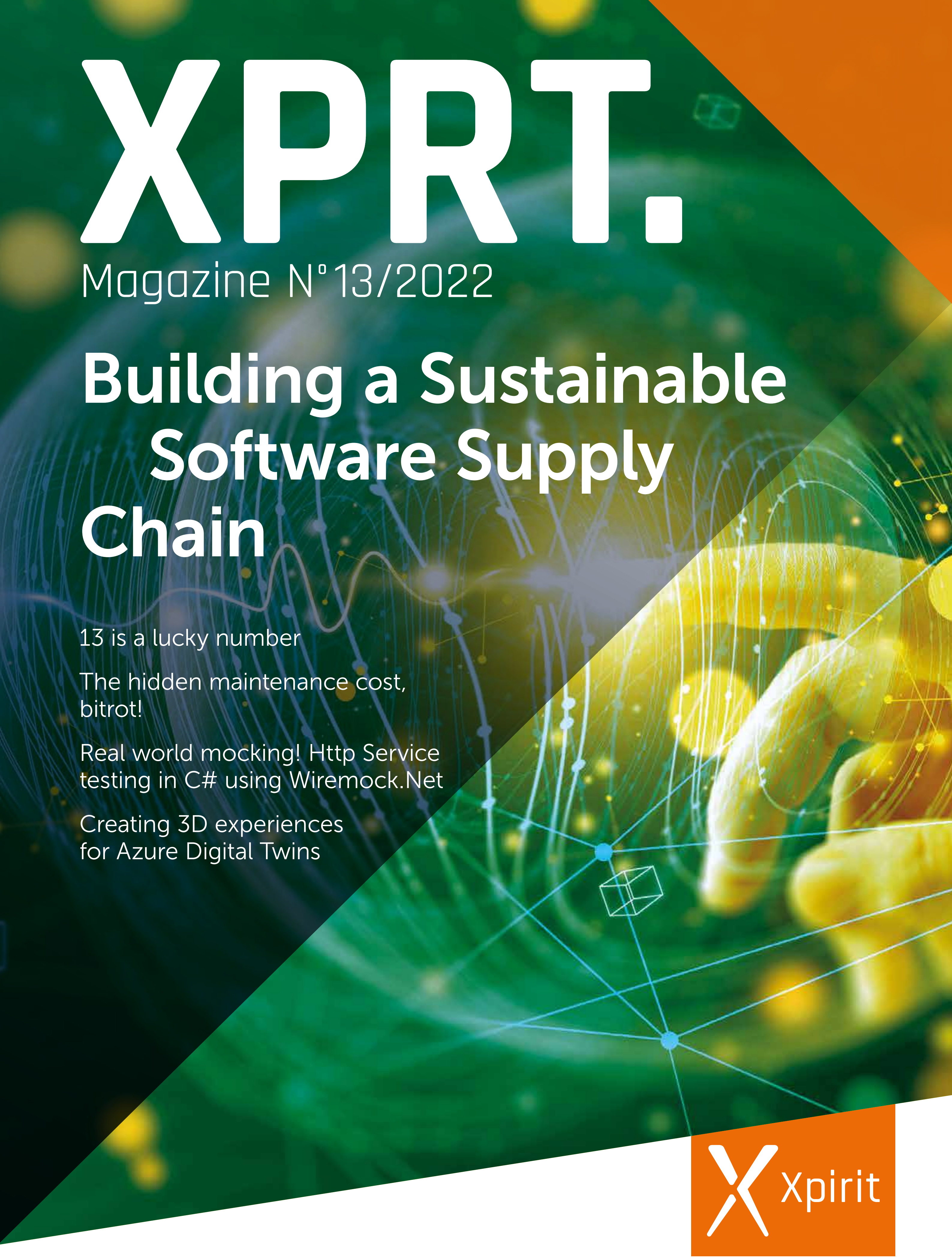
XPRT.

Magazine N°13/2022



Building a Sustainable Software Supply Chain

13 is a lucky number

The hidden maintenance cost, bitrot!

Real world mocking! Http Service testing in C# using Wiremock.Net

Creating 3D experiences for Azure Digital Twins

Xpirit

Together we drive change.

Transforming your business will not work without the right knowledge

- ✓ **Certified Microsoft Azure Fundamentals** (AZ-900)
- ✓ **Certified Microsoft Azure Administrator** (AZ-104)
- ✓ **Certified Microsoft Azure Developer** (AZ-204)
- ✓ **Designing Microsoft Azure Infrastructure Solutions** (AZ305)
- ✓ **Certified Microsoft DevOps Engineer Expert** (AZ-400)

Learning Journey

- ✓ **Azure DevOps Engineer Expert** (AZ-900 · AZ-104 · AZ-400)
- ✓ **Azure DevOps Engineer Expert** (AZ-900 · AZ-204 · AZ-400)
- ✓ **Azure Solutions Architect Expert** (AZ-900 · AZ-104 · AZ-305)
- ✓ **Azure Developer Associate** (AZ-900 · AZ-204)
- ✓ **Azure Administrator Associate** (AZ-900 · AZ-104)

We offer tailer made solutions, contact Max for all options.

Max Verhorst / +31 (0)6 13 46 80 02 / mverhorst@xpirit.com

xpirit.com/training

Colophon

XPRT. Magazine N°13/2022

Editorial Office
Xpirit Netherlands BV

This magazine was made by
Xpirit Netherlands André Geuze, Anne Meijer, Arjan van Bekkum, Bas van de Sande, Casper Dijkstra, Chris van Sluijsveld, Davy Davidse ,Diederik Tiemstra, Dennis Thie, Duncan Roosma, Erick Segaar, Erik Oppedijk, Erwin Staal, Geert van der Cruijsen, Hans Bakker, Hindrik Bruinsma, Immanuel Kranendonk, Jasper Gilhuis, Jesse Houwing, Jesse Swart, Jesse Wellenberg, Kees Verhaar, Loek Duys, Liuba Gonta, Maarten Blok, Maira Duijst - Camu, Manuel Riezebosch, Marc Bruins, Marcel de Vries, Mark Foppen, Mark de Haas, Maria Stepanova, Martijn Tieland, Martijn van der Sijde, Matthijs van der Veer, Max Verhorst, Michiel van Oudheusden, Natascha Former, Niels Nijveldt, Patrick de Kruijf, Patrick van Kleef, Reinier van Maanen, René van Osnabrugge, Reda Fakirmohamed, Reza Atlaschi, Rik Groenewoud, Rob Bos, Robert de Veen, Roy Cornelissen, Rutger Buiteman, Sander Aernouts, Sander Trijssenaar, Sofie Wisse, Suraj Sewbalak, Thijs Limmen, Tiamo Idzenga, Tijmen van de Kamp, Victor de Baare

Xpirit Belgium Pieter Gheysens, Gill Cleeren, Annemie Vandenberghe, Wesley Cabus, Kristof Riebbels, Kristof van Hees, Jasper Van Mensel, Laurenz Ovaere, Wouter Van der Auwera, Jordi Borghers, Stephane Eyskens, Pieter Nijs, Lesly Bernaola, Sorin Pasa

Xpirit Germany Michael Kaufmann, Thomas Tomow, Tobias Mackenroth, Olena Borzenko, Andreas Läubli, Gema Morilla Guirado, Andriy Chevychalov, Michael Contento

Xpirit USA Esteban Garcia, Robert Bremer, Elizabeth Abreu, Natalie Reinfordi, Stuart Celarier, David Sanchez

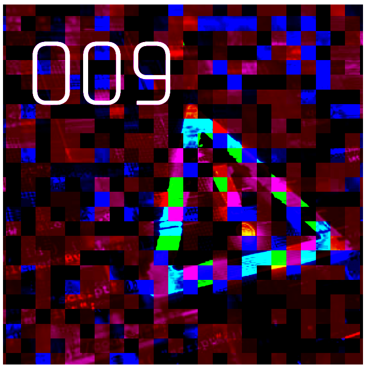
Contact
Xpirit Netherlands BV
Laapersveld 27
1213 VB Hilversum
The Netherlands
Call +3135 538 19 21
mverhorst@xpirit.com
www.xpirit.com

Layout and Design
Studio OOM
www.studio-oom.nl

Translations
Mickey Gousset (GitHub)

If you prefer the digital version of this magazine, please scan the qr-code.

In this issue of **XPRT.** Magazine, our experts share their knowledge about building a Sustainable Software Supply Chain.



INTRO

STATE OF THE ART SOFTWARE DEVELOPMENT

POWER THROUGH PLATFORM

MOVING THE BUSINESS NEEDLE

APPROPRIATE CONTINUITY

13 is a lucky number

Friday, 9th September, was our Xpirit Innovation Day. With around 100 people from Xpirit Netherlands, Xpirit Belgium, Xpirit Germany and Xpirit USA we innovated on all the things we like. The last few times, we combined our innovation day with some global team building with drinks, dinners and a great location. This time we went back "to normal" at our office. Personally, I really liked this. Back to the essence of our innovation day and actually our company. Sharing Knowledge and learning together. Our innovation day has no rules or boundaries, no required deliverables (besides a demo at the end of the day) and is fully focused on learning!

Author René van Osnabrugge

When I walked around the office during our innovation day, I picked up the first magazine we created back in 2015. It looks different, the content has become irrelevant, but in its essence, it is the same. Great articles written by passionate people that write about the things they love. Sharing their knowledge and working on their and our authority mission. Back then, I was extremely proud when we produced this first magazine. And that feeling did not change with the 11 editions that followed. Fast forward to this 13th edition, I am still extremely proud of our magazine, but even more our knowledge driven culture. The only thing that changed is we are now 100 people instead of 8.

All these 100 people that are sharing knowledge and work with our customers have another thing in common. We all help our customers to become a high-performing software company by helping them build an inclusive and secure organizational culture that fosters talent and is driven by sharing and equality so they can achieve the impossible. We call this building an engineering culture. In our previous magazine we talked about what an Engineering Culture is and introduced the eight pillars that contribute to this. It all starts with **State of the Art Software Engineering** where we focus on building high-quality software. The pillar **Smooth Delivery** describes all the work and knowledge we need to get this software in production. **Appropriate Continuity** is all about Security and Compliance and keeping the software running. We do this to help customers to **Move the business needle**. To enable this and get the desired speed, we use the **Power through Platforms** and an **Empowering Operating Model**. We can only enable this to be a **Knowledge Driven** company where we share and build knowledge.

And last but not least, we need to build an **Epic Workplace**. This building of an engineering culture is the central theme that binds us at Xpirit. We all work to achieve this from our own perspective and expertise. And this magazine is another great reflection of this.

We are still on top when it comes to State of the Art Software Engineering. In this magazine, you can find a number of great articles. We talk about Bitrot, Blazor, Wiremock, Frontends and Nullable Reference types. You can also see that we build on the shoulders of giants by leveraging the Power Through Platforms. Our guest writer from GitHub, Remco Vermeulen, writes about GitHub in combination with CodeQL. We share our finding on the security investigation on GitHub actions. We show how Azure Virtual Desktop can really help you to leverage the power of cloud. You can also read an article about Azure Digital Twins and how it can help you to use the power of IoT combined with 3D visualization.

We have an article on monitoring using Grafana and how it can help to assure the continuity of the business. On a somewhat higher abstract level, we talk about moving the business needle, and last but not least, we introduce you to our new member of the Xpirit group, Xpirit USA.

In This 13th magazine, you will find 13 great articles. It is a reflection of what keeps us busy at the moment. A treasure of knowledge and hopefully an inspiration for you, our reader. That said, 13 is definitely a lucky number. Please let us know what you think on any of our channels, and thank you for reading! Enjoy! </>

René van Osnabrugge
Chief Technology Officer

rvanosnabrugge@xpirit.com

Accessible web components by design

Websites and apps in the public sector and governments must comply to the W3C accessibility standards. Everyone should be able to easily find, view and use these websites. For all other websites there are no regulations. Research have shown that 97.4% of the top one million websites don't offer full accessibility. These websites had an average of 50 high/critical accessibility findings on their homepage[1]. This is a lot and most of the time these issues are not hard to resolve.

Author Thijs Limmen

15% of all people have some sort of disability. For visual impaired people it's important that a website has good colour contrasts. People with motor/mobility issues need a website that can be accessed using just a keyboard or custom input devices. For people with learning/cognitive problems a website should be structured, intuitive, calming. People with hearing difficulties would want to have captions on video. Incidentally anyone could be disabled: Sleep deprivation or breaking an arm.

Improving accessibility is not just about making a website available for these 15% of people with a disability. It benefits all people in any environment. For example, having the correct colour contrasts also helps people with perfect eyesight using a website, for example, outdoors in bright sunlight. Having captions on video helps people watch a video on mute in a busy environment. In general, websites that have full accessibility are perceived better. More-over Gartner has researched that Digital products in full Web Content Accessibility Guidelines (WCAG) Level 2 compliance are expected to outperform their market competitors by 50% by 2023. (Gartner, 2020[2]).

Many big companies have their own identity and branding (design system). This design system is in many cases expressed through reusable style classes and accessibility tags which are put on HTML elements. For different pages, many layouts and patterns are copy-pasted. Websites grow fast. It's hard to maintain accessibility and over time accessibility becomes inconsistent, simply forgotten, or not important anymore.

It's important to think about accessibility from the start in your web development process. In this article I will explain about reusable web components that are accessible by default.
You will see how to implement a reusable web component using the W3C accessibility standards and how to test such component on its accessibility features.

After, I'll cover the "by design" part of the article title and shine some light on how you could improve the usability of your website. Usability is about designing products to be effective, efficient, and satisfying (a.k.a. user experience design). Not to be confused with accessibility.

Accessible by default component library

This component library contains all the building blocks from the design system your UX designer invented. If you create components that are accessible by default, then you only must implement accessibility once per type of component.
Jacob's Law from Laws of UX[3] (See Laws of UX paragraph) explains that users spend most of their time on other websites, so they prefer your website to work the same. This also counts for the accessibility of your components. Luckily for all types of components there are well documented patterns about how to make them accessible based on the W3C standard[4].

Always apply the dumb vs smart components best practice when creating components in your component library.
Most component should be dumb and only have little state and presentational logic. Smart components have more responsibility like doing http calls, managing state.

[1] https://blog.hubspot.com/website/accessibility-statistics
[2] https://www.gartner.com/en/documents/3986300/compliance-and-beyond-4-ways-digital-accessibility-gives
[3] https://lawsofux.com/en/jakobs-law/
[4] https://www.w3.org/WAI/ARIA/apg/patterns/

These components usually consist of many dumb components (building blocks) and is mapping state onto them. Keep the dumb components small with single responsibility.

Many people with disabilities make use of screen readers in order to interact with a website. Screen readers will read out to the user what is currently visible, focussed, possible options, etc. aria-tags are used on HTML elements in order to let your screen reader interact with them. In the next chapter you will see how these aria-tags are used in order to create an accessible by default checkbox component.

Tip: Install the WAVE chrome browser plugin5 to easily see if 'aria' tags are being used (correctly), so screen readers interact with your website better.

Tip: Enable a screen reader, close your eyes and try to complete a task on your website. 😉

Accessible by default checkbox component

For this example, I'll cover the checkbox component. In many cases you must implement your own checkboxes, because the regular HTML checkboxes are very hard to style.

The following is the W3C accessibility standard explanation for a checkbox:

- An element has role '**checkbox**', so that screen readers understand that it's interacting with a checkbox.
- The checkbox has an accessible label provided by the visible text content contained within the element with role checkbox. So, people using a screen reader know what checkbox they are interacting with.
- When (un)checked, the checkbox element has state '**aria-checked**' set to true/false, so that screen readers understand that a checkbox is checked/unchecked.

Create a '**dumb**' component in your framework of choice that is used with the following HTML signature:

```
HTML
<custom-checkbox value="false"
changed="onChanged($event)">Unchecked</custom-checkbox>
```

The implementation of this component is as follows. As you see, it contains everything the W3C standard explained.

```
HTML
<div class="custom-checkbox-style" role="checkbox"
aria-checked="{{ value }}" tabindex="0">
  <slot></slot>
</div>
```

- Use tabindex="0" to make the div focusable and interactable.
- <slot></slot> is used for content projection. For the above example, the text 'Unchecked' will be projected here.

Now you never have to think about accessibility for your checkbox component ever again!

Accessibility driven test (Testing Library)

You created the accessible by default component and now you want to test the accessibility features of it. Testing Library[6] is a great tool that can help with this. A web component's unit test should not test the component's instance and its methods/properties. What is visible to the user and what the user can interact with is what matters.

Bad example without Testing Library (Angular):

```
Typescript
    const checkbox: HTMLInputElement =
fixture.nativeElement.querySelector('input');
    expect(checkbox.checked).toBeFalsy();
    const event = new MouseEvent('click');
    checkbox.dispatchEvent(event);
```

- Doesn't test if this div has the role checkbox
- Doesn't test if the checkbox is labelled for screen readers to understand
- Doesn't test if the checkbox has aria-checked
- Doesn't actually click a button
- It's very technical

Example with Testing Library with a user pressing spacebar to check the checkbox:

```
it('should check checkbox when user presses the spacebar'',
() => {
  // Mount checkbox component

  // Arrange
  const checkbox = screen.getByRole('checkbox', { checked:
  false, name: 'Unchecked' });
  checkbox.focus();

  // Act
  userEvent.keyboard('{space}');

  // Assert
  expect(checkbox).toBeChecked();
  // expect value to be emitted
});
```

- Tests that a checkbox has role="checkbox"
- Tests that the checkbox has aria-checked="false"
- Tests that the checkbox that is labelled for screen readers to pick up
- Tests that the checkbox can receive focus (tabindex="0")
- It tests user interaction through spacebar keyboard press, following the W3C standard.

You should add additional tests for the following tests:

- Uncheck when user clicks the checkbox
- Focus checkbox when user presses tab
- Can you think of more accessibility tests?

[5] https://chrome.google.com/webstore/detail/wave-evaluation-tool/jbbplnpkjmmeebjpijfedlgcdilocofh
[6] https://testing-library.com/

Testing library works the same for all major frontend frameworks. So, a Testing Library test written in Angular, can be re-used when migrating to a different framework like React.

Laws of UX

When thinking about Usability there are many rules that could apply. Laws of UX[7] is a collection of easy-to-understand best practices for designing user interfaces. It contains the takeaways from many psychological studies about user experience. It's good to know about the psychological laws the top apps use to keep you engaged in their apps. For example, Doherty Threshold principle about the perceived waiting experience of a user:

Doherty Threshold[8]

Productivity soars when a computer and its users interact at a pace (<400ms) that ensures that neither has to wait on the other.

1. Provide system feedback within 400 ms in order to keep users' attention and increase productivity.
2. Use perceived performance to improve response time and reduce the perception of waiting.
3. Animation is one way to visually engage people while loading or processing is happening in the background.
4. Progress bars help make wait times tolerable, regardless of their accuracy.
5. Purposefully adding a delay to a process can increase its perceived value and instil a sense of trust, even when the process itself takes much less time.

It's nice to reference these principles when developers have created an interface that is not usable, which I have seen a lot.

Tip: Always have a dedicated UX designer available when creating any website. Preferably already in the initiation phase of a project.

Visual Component Testing and Vite 3.0

A big trend currently is visual component testing. Many popular frameworks such as Cypress, Playwright and Storybook are now publishing Beta support for this. Visual tests were usually very slow, flaky and/or hard to maintain, but that has now changed through the power of Vite 3.0[9] (means 'fast' in French). It is a build tool that aims to provide a faster and leaner development experience for modern web projects. It is revolutionising web development and making Webpack a thing of the past.

Now tests are fast, and developers get very good visual feedback about the state of their test. Also inspecting a browser and seeing logs in a browser console for debugging is a lot better than interpreting the Terminal test output and guessing the state a view is in.

I would currently suggest Cypress 10 Component Testing Beta. I think it currently has the best developer experience compared to the other tools and tests are very fast. It comes with a complete tool which makes it easy to navigate through your test suite, re-run tests, debug tests, and step through in-between steps of a specific test.

Tip: Try-out the Cypress 10 Component testing introduction[10].

Practical Usability Tests

Ask a close relative or colleague to test your website. You will be surprised how many things don't work well or are confusing to the user:

1. Give them a goal to accomplish, like registering an account.
2. Just observe them. Do not help, don't speak. They might be struggling, or not.
3. Look at their non-verbal communication. A smile, a confused look, body language. This should give you enough information. Words are not that important.

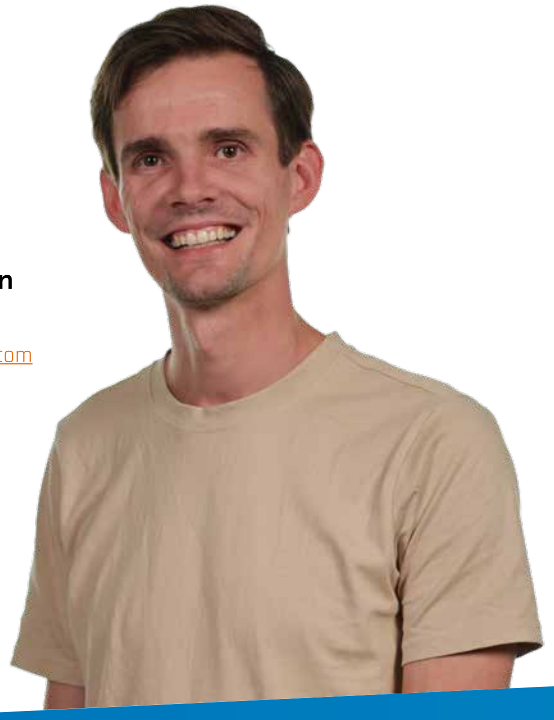
Optional:

4. Film their upper body and the corresponding screen recording to play back.

You should reach out to accessibility user groups including people with disabilities that like to help with testing your website's accessibility.

You could also hire a company that specializes in doing usability tests with actual end-users of your website or people that fit a specific persona.
The reason this works so well is that you are biased, and you went through a specific flow on your site maybe ten to hundreds of times. Do you remember being new to something and you spot everything that doesn't work well?

Let's create a more accessible internet for everyone! </>



Thijs Limmen
Consultant

tlimmen@xpirit.com

[7] https://lawsofux.com
[8] https://lawsofux.com/en/doherty-threshold
[9] https://vitejs.dev/guide/
[10] https://docs.cypress.io/guides/component-testing/writing-your-first-component-test

The hidden maintenance cost, bitrot!

Sometimes it is very hard for customers to understand the hidden costs involved when you build custom software. By a hidden cost, I mean a phenomenon that apparently is back in our industry called BitRot.

Author Marcel de Vries

What is BitRot?

Back in the day, BitRot was caused by the fact that the magnetic media we used to store our computer programs sometimes lost their magnetic information, causing problems reading the data back. With the industry moving to new ways to store data, like solid-state drives, the problem is still there but not predominantly visible anymore. We have algorithms that store our data in such a way that the lost data can be recovered, and from an end-user perspective, BitRot seems to have vanished completely.

0 bits flipped 1 bit flipped 2 bits flipped 3 bits flipped

Data degradation – Wikipedia

Bitrot redefined

Although the original problem has gone more or less away, we are now confronted with a completely new way that bitrot is coming back in our industry. You might disagree with me about reusing the name BitRot for something different, but in essence, the problem we face manifests itself in the same way. If we don't touch the software, we write for even a few weeks, the software deteriorates!

Let me explain what is going on here.

BitRot today is the issue when we don't touch our software for a few days or weeks, and the software deteriorates caused by a number of sources. Let me share a few of the issues you will face when building software today.

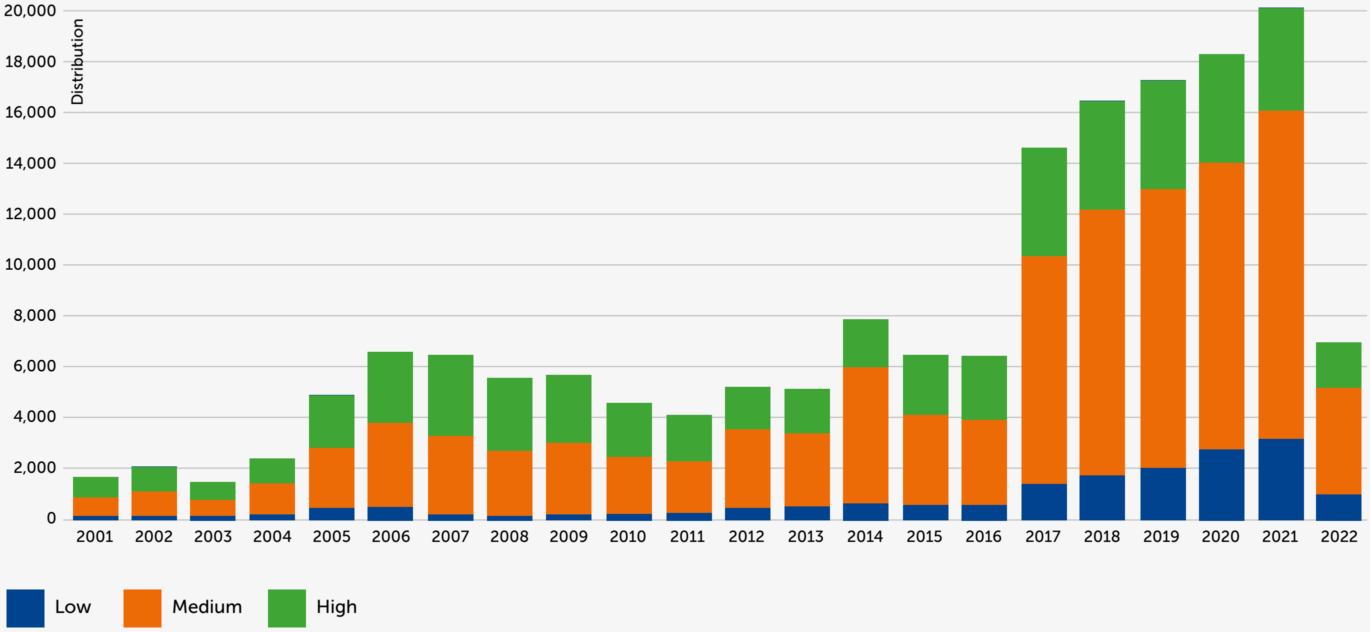
New known vulnerabilities

A known vulnerability is some weakness found in the software you wrote or in any software you used to build your application or website that someone can exploit. You might think, why would my software all of a sudden become exploitable while I haven't touched it? This is caused by the fact that attackers become smarter each day. They find new innovative ways to exploit software. Since the software that is written often contains tons of code not only written by your own company but also open source components, the chance of your software becoming vulnerable itself is a significant risk. This does not immediately mean your software will be exploited as well, but the likelihood that your software becomes exploitable will increase almost every day. This is something you need to keep track of and you need to make updates or changes to your software to keep up with the current state of the industry. The number of vulnerabilities found is also increasing all the time. You can see how fast this is picking up in the following graph the NVD publishes[1]. I added a capture of the chart that shows how many known vulnerabilities are found and that the rate of finding them is constantly increasing.

[1] https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time

CVSS Severity Distribution Over Time

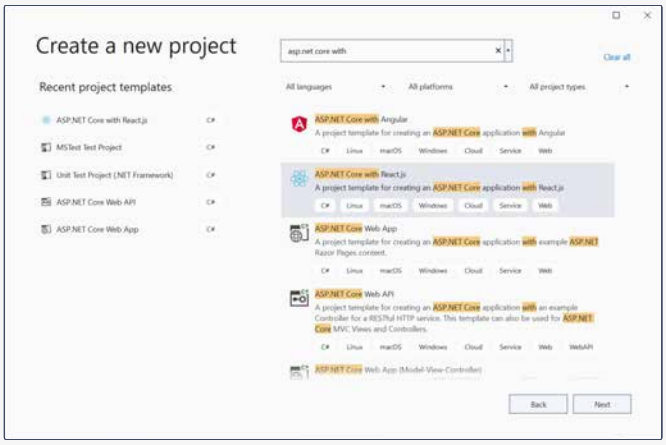
This visualization is a simple graph which shows the distribution of vunerabilities by severity over time. The choice of LOW, MEDIUM and HIGH is based upon the CVSS V2 Base score. For more information on how this data was construted please see the NVD CVSS page.



Updates of used frameworks or packages

Your software is rarely built from scratch. To build software, you will use other software components all the time. Depending on the technology you use to write your programs, you use NuGet packages (.NET), Maven packages (Java), Node Packages (Node/Web Development), RubyGems (Ruby development), etc. These packages are built and maintained by others and, coming back to the previous topic, they also need to maintain their software to keep it free from known vulnerabilities. Also, they want to provide new capabilities and features constantly. This implies those packages will get new versions all the time, and keeping up with those more recent versions is not to be taken lightly.

Let's assume you build a simple Hello World web application using .NET 6 and React. You can see the screenshot of what I did to create the application in Visual Studio 2022:



Create a new project

asp.net core with

Recent project templates

- ASP.NET Core with React.js C#
- MSTest Test Project C#
- Unit Test Project (.NET Framework) C#
- ASP.NET Core Web API C#
- ASP.NET Core Web App C#

All languages · All platforms · All project types

ASP.NET Core with Angular
A project template for creating an ASP.NET Core application with Angular
C# Linux macOS Windows Cloud Service Web

ASP.NET Core with React.js
A project template for creating an ASP.NET Core application with React.js
C# Linux macOS Windows Cloud Service Web

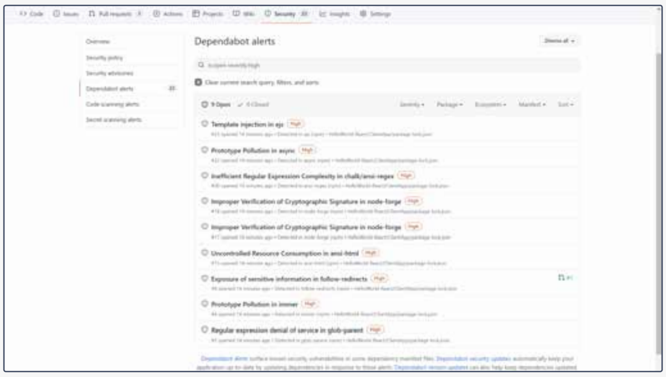
ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

ASP.NET Core Web App (Model-View-Controller)

New ASP.NET with React project

This will result in an astonishing set of 1,487 Dependencies from the NodeJS ecosystem and 15 more from the .NET ecosystem. Starting with a clean template (I updated everything before I created the application in Visual Studio), this already resulted in 23 Known Vulnerabilities of which 9 are at the level of High!



Dependabot alerts

9 Open

- Template injection in ejs High
- Prototype Pollution in async High
- Inefficient Regular Expression Complexity in chalk/ansi-regex High
- Improper Verification of Cryptographic Signature in node-forge High
- Improper Verification of Cryptographic Signature in node-forge High
- Uncontrolled Resource Consumption in ansi-html High
- Exposure of sensitive information in follow-redirects High
- Prototype Pollution in immer High
- Regular expression denial of service in glob-parent High

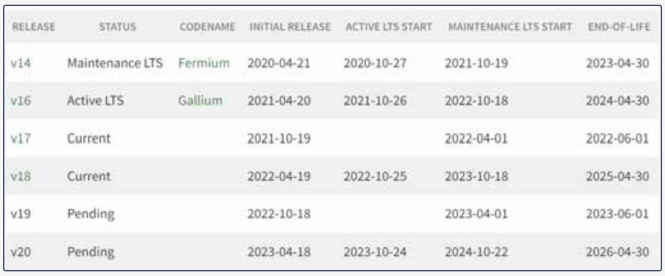
Analysis of new project with Dependabot (GitHub)

Updates on compilers and tooling

Then there are the dependencies on Visual Studio 2022, which has updates at least once every month. We took a dependency on the .NET framework that is updated at least 1x per year and every two years has a new stable release that is supported for a maximum of 3 years. Not to mention all the small updates that might come to fix bugs and vulnerabilities. And finally, we took a dependency on the NodeJS toolset, which is also updated multiple times a year. These tools also tend to make breaking changes. You must constantly keep them up to date because they can also contain known vulnerabilities that might compromise your development environment!

Newer versions of the languages and frameworks

Finally, there are also dependencies on the languages we use. In this example, I used React which is Javascript/typescript based, and C#10 for the .NET codebase. C# is updated on a yearly cycle, and if you look at the versions of Node, then you see you need to update this every six months:



RELEASE	STATUS	CODENAME	INITIAL RELEASE	ACTIVE LTS START	MAINTENANCE LTS START	END-OF-LIFE
v14	Maintenance LTS	Fermium	2020-04-21	2020-10-27	2021-10-19	2023-04-30
v16	Active LTS	Gallium	2021-04-20	2021-10-26	2022-10-18	2024-04-30
v17	Current		2021-10-19		2022-04-01	2022-06-01
v18	Current		2022-04-19	2022-10-25	2023-10-18	2025-04-30
v19	Pending		2022-10-18		2023-04-01	2023-06-01
v20	Pending		2023-04-18	2023-10-24	2024-10-22	2026-04-30

Releases | Node.js (nodejs.org)

Those language and framework updates can be significant if you look at the amount of work involved to actually use the new capabilities. Not using them still makes your codebase deteriorate from a maintainability perspective since the industry is moving on with new ways of utilizing the language and framework features. New team members on a project will have a hard time adapting old programming styles and inefficiencies if you only ensure it compiles.

How can we mitigate this issue?

First of all, the most important thing is that you allocate time to keep things clean. This means allocating a certain budget of time to keep your packages up to date and spending time updating to newer minor or major versions of packages that come available. There is no silver bullet in solving this issue, but taking the time for it is one step in the right direction.

Secondly, you can automate some of this when you are using, e.g., GitHub as a DevOps platform. Here you have the option to enable a feature called Dependabot that will inform you that your packages are out of date or contain known vulnerabilities. When it knows the vulnerability can be mitigated with a new package version, it will even create a pull request for you. You still need to review it and approve the PR, so it becomes part of the main codebase. There are also tools that can integrate with most DevOps platforms to manage automated updates for you e.g. Renovate[2].

And then the final check is, when do I take the dependency? Be very aware of the fact that taking the dependency adds a maintenance cost. Building something yourself will also incur significant costs, but it should be a decision you make and not just a default you accept all the time. Be very aware of the tradeoff and that a dependency incurs a cost as well.

Conclusion

The software you build is in a constant state of decay, and you must allocate a significant portion of time to keep things evergreen and up to date. Waiting for your updates will cost you significantly more time than updating constantly.
The adage "if it hurts, do it more often" applies here and makes the software delivery cadence more predictable and more secure. So make updating packages, frameworks, and languages part of the standard maintenance cycle!

The real challenge lies in making our customers aware of this problem and finding ways to make them aware that they need to maintain software. You can not leave software untouched for a few weeks since, in the meanwhile, the software becomes outdated and vulnerable. And last but not least it will become more complicated each day to make it up to date again. </>

Marcel de Vries
Chief Executive Officer

mdevries@xpirit.com

MVP Microsoft Most Valuable Professional

[2] https://docs.renovatebot.com/

Start dealing with Nullable Reference Types!

Since C# 8, developers can opt in to use Nullable Reference Types (NRTs). With NRTs, developers get a set of 'tools' to avoid the most common exception: the NullReferenceException. These tools include the nullable annotations, attributes, warnings and static code analysis.

Author Pieter Nijs

Starting from .NET 6, NRTs are enabled by default when creating a new project. You can always opt-out if you want, but -to be honest- I think you shouldn't!

And even though NRTs have existed for some time, I still find a lot of people are hesitant about adopting it. Maybe unknown is unloved? I strongly believe that fully embracing NRTs results in better, more understandable, and more robust code! Let me show you what I mean with that and how you can start using NRTs.

It's all about expressing intents

One of the main reasons NRTs are great, is because it gives a lot more meaning to your code. It makes code less ambiguous, more understandable.

Value Types

Being able to explicitly indicate that a value type is nullable or not, is something we are so accustomed to. The intent of using a value type is clear: there will always be a value to expect or to provide, they cannot be null. Nullable value types, on the other hand, can be used in scenarios where it is perfectly legitimate that a value might be null, that a value is optional; there are use cases where it makes sense not to have a value.

Reference Types

Thanks to Tony Hoare 's "The billion dollar mistake" (https://en.wikipedia.org/wiki/Tony_Hoare), reference types can be null. What's even worse is the fact that in .NET values of reference types are null by default and -prior to C# 8- there wasn't any way to indicate that an instance of a reference type should not be null. We used to have Code Contracts in earlier versions of .NET, which could help with defining what should be nullable and what not.
But Code Contracts are end of life and are being superseded with NRTs.
The most common way developers deal with nullability, is by doing null checks where they wouldn't want to have null values and throw an Exception.
The downside? These are runtime checks! If a developer passes a null value where it isn't expected, it's not known until runtime that it isn't allowed.
The following snippet shows a method signature which, when not using NRTs, has many uncertainties:

```
public virtual IEnumerable<T>
Search(SearchCriteria<T> criteria,
SortParams<T> sortParams)
{
}
```

Is it possible to pass a null value as search criteria if I want the entire list of items? Can I pass null as SortParams if I don't want my search result to be sorted? If we don't find anything, I'll get back an empty list, right? Or will the result be null?

By looking at the method signature, we just don't know. We don't know for sure what the intent of this method and its parameters are, or what the intent of the developer was. Oh, and did you notice this method is virtual? Another class inheriting this class, might override this method and could possibly handle the parameters and return type entirely differently. Not to mention that this is considered a bad practice as it clearly violates the Liskov Substitution principle.

When not using NRTs, there is no way - like with (nullable) value types - to express the intent. Is it allowed to pass a null value to a certain method? Will a value returned by a method always have a value, or should I do a null-check? Because the platform doesn't force us to think about nullability, developers tend to not really think about it.
The result? Firstly: NullReferenceExceptions! Secondly: lots of code where the intent is not always clear.

Having NRTs forces you to take nullability into account! It makes your code a lot more self-describing: it's all about expressing intents and defining what's expected, what's possible and what's not.

When working with NRTs, the above Search method becomes a lot more unambiguous:

```
public virtual IEnumerable<T> Search(SearchCriteria<T>?
criteria, SortParams<T>? sortParams)
{
}
```

Both arguments can be nullable and we're rest assured we'll never get back a null value from this method. Just like that, by looking at the method signature, we can immediately identify its intended use.

Let's do this!

Since C# 8 we can enable a nullable annotations context in order to use NRTs. With NRTs, developers need to explicitly indicate whether a value of a reference type can be null, as they are non-nullable by default.

Enabling NRTs for your entire project can be done by adding the following to your csproj file:

```
<Nullable>enable</Nullable>
```

You could also choose to enable or disable NRTs on a file basis, by using the following directives:
#nullable enable
Or
#nullable disable

Once enabled, a nullable reference type can be defined by adding a '?' to the declaration, in analogy with value types:

```
string? text = null; //nullable
string text2 = "Hello world"; //nonnullable
```

Unlike with nullable value types, we are not changing the underlying type by adding a '?': both 'string' and 'string?' refer to the System.String class. The compiler will keep track of the nullable annotation, but the type itself doesn't change! With this extra information, the compiler can do very interesting and helpful code analysis. Take a look at the following listing:

```
string? text = null; //nullable
string text2 = "Hello world"; //nonnullable

if (text.ToLower() == "hello")
{
  //Warning! text is nullable but we haven't checked
  //whether it is nul or not!
}
else if (text2.ToLower() == "world")
{
  //No warning. text2 is nonnullable, we
  //shouldn't check if it is null oer not
}

//Warning! We should not assign null to nonnullable
variable text2 = null;
```

See how immediately the compiler starts to give us warnings. The text variable is nullable, so we should do a null-check prior to calling a method on it, unlike with text2 which is non-nullable. But because text2 is non-nullable, we do get a warning when assigning null to it.

Don't ignore the warnings

Way too often, I see that warnings are being ignored by developers. I, personally, always enable 'Treat Warnings As Errors' on the projects I'm working on. Enabling NRTs without making sure all related warnings are resolved, doesn't make much sense to me. Not everybody wants to treat warnings as errors, but did you know you can tell the compiler to only treat NRT-related warnings as errors? Adding this to you csproj does just that:
<WarningsAsErrors>nullable</WarningsAsErrors>

When enabling NRTs, you really should have this set as a minimum! It forces developers to really focus on dealing with all the aspects of NRTs.

Ideally, you should enable 'Treat Warnings As Errors' in your (future) projects!

Let's fix the above warning by adding an explicit null-check:

```
if (text is not null)
{
  if (text.ToLower() == "hello")
  {
    //No warning ...
  }
}
```

Notice that the error we had previously on the text variable is now gone!

The compiler will constantly track a variable's null-state. In Visual Studio we can consult the inferred null-state by hoovering over the variables. The tooltip will say whether a variable may be null or not null at a given place.

Now, take a look at the following:

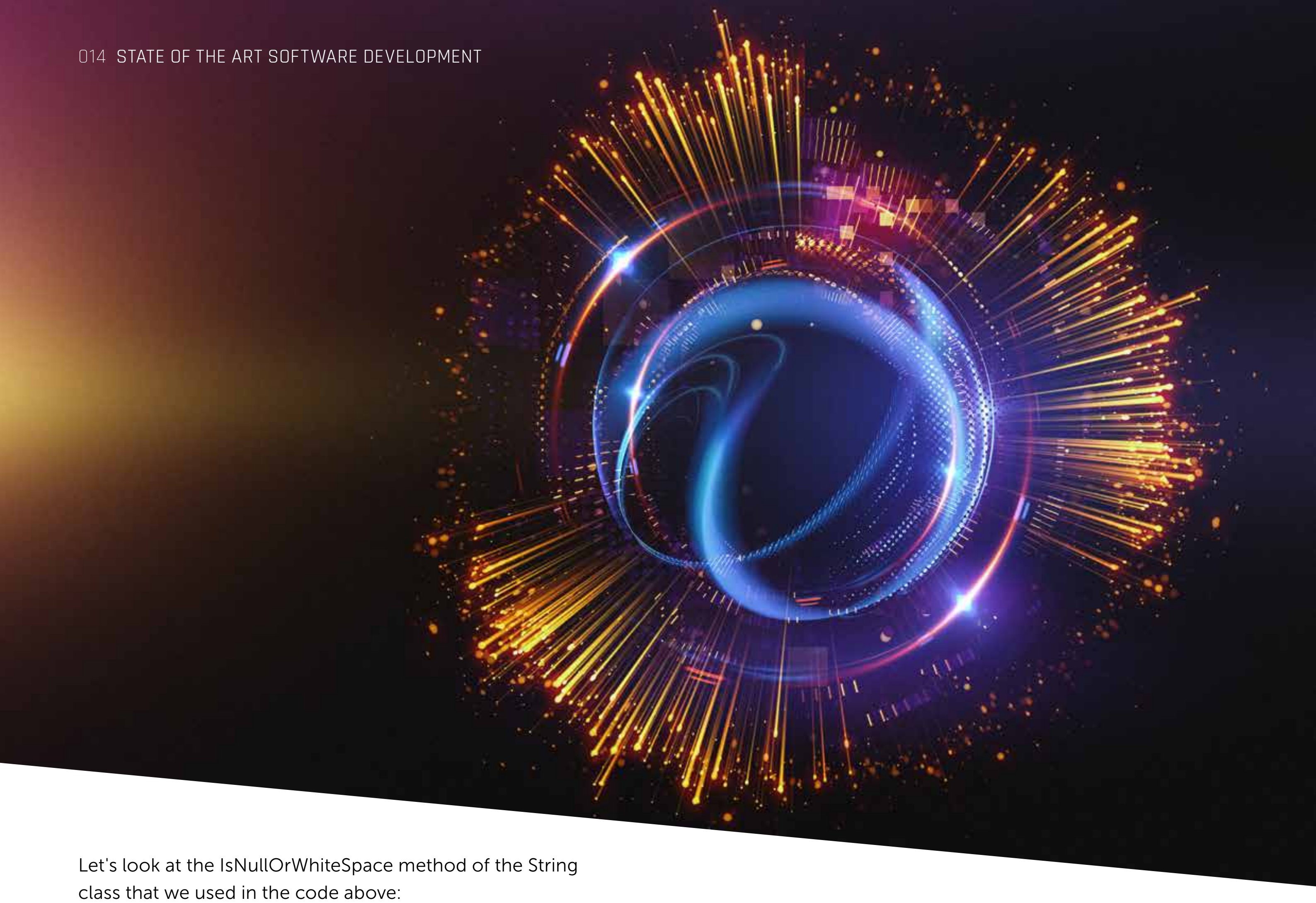
```
if (!String.IsNullOrWhiteSpace(text))
{
  if (text.ToLower() == "hello")
  {
    //No warning?
  }
}
```

No warning. And if we would hover over the text variable after the first if, the tooltip would say the text variable is not null here. How does the compiler know that? We don't do an explicit null check, but still the analyzer infers the variable's null state after calling the IsNullOrWhiteSpace method.

The answer is:

Attributes

There are a handful of attributes that we can use to help the compiler figuring out the null state of variables.



Let's look at the IsNullOrWhiteSpace method of the String class that we used in the code above:

```
public static bool IsNullOrWhiteSpace
([NotNullWhen (false)] string? value)
```

The NotNullWhen attribute can be used to tell the compiler that the given argument, which is nullable, will be not null when the method returns the specified bool value (false in this case).
It is because of this attribute that, in our previous example, the compiler was able to determine that text is not null inside that if-block as we would only enter that if-block when the IsNullOrWhiteSpace method returns false.

All those attributes that help the compiler infer the null state of variables can be divided in different categories.

Conditional post-condition
Attributes in this category let the compiler infer the null state of a variable based on the return value of that method.
Just like the NotNullWhen attribute we just discussed.

Another example of such an attribute in this category is the MaybeNullWhen attribute. This one is typically used for Try-methods, like the IDictionary's TryGetValue method for example:

```
if (dictonary.TryGetValue("key", out string? result))
{
  //result is not null
  Console.Writeline(result.Length);
}
else
{
  //result can be null
}
```

Although we're passing-in a nullable string value as out parameter, inside the if-block, the compiler doesn't give us any warning about the result value potentially being null. That is because the definition of the TryGetValue looks like this:

```
bool TryGetValue (TKey key, [MaybeNullWhen (false)] out
TValue value);
```

Because of the MayBeNullWhen attribute, the compiler knows that the out value can only be null when the method returns false. Hence, in the above example, we don't get a warning when accessing the Length property on the result variable when the TryGetValue method has returned true.

Another attribute in this category is the NotNullWhenNotNull attributeAn example of this attribute can be found in the Path's GetFileName method:

```
[return: NotNullIfNotNull ("path")]
public static string? GetFileName (string? path)
```

This attribute tells the compiler that the return value will not be null if the provided parameter is not null.

Let's see this in action:

```
if (path is not null)
{
  var filename = Path.GetFileName(path);
  Concole.WriteLine(filename.ToUpper());
}
```

Because of the null-check on the path variable prior to passing it to the GetFileName method, the compiler infers that the resulting filename variable is not null, thanks to the NotNullIfNotNull attribute.

Note
These attributes are used on classes and methods in the BCL starting from .NET 3.0 and .NET Standard 2.1. Can you use NRTs in .NET Standard versions prior to 2.1? Yes! But methods and classes won't have these attributes on them to help the compiler to infer null state, so you will have to do additional null-checks yourself if you want to get rid of all warnings.

The previous examples showed examples of attributes on baked-in methods and types in the BCL. We can, of course, add these attributes to our own methods as well. Interestingly, when applying these attributes, you will get warnings when the implementation of the method doesn't match with the attribute defined. Take a look at following snippet, which is a good example of using the NotNullIfNotNull attribute:

```
[return: NotNullIfNotNull("username")]
static Uri? GetProfilePicture(string? username)
{
  if (username is not null)
     return new Uri($"http://mysite.be/users/{username}.png");
  return null;
}
```

If I would have made a typo, and typed 'is null', instead of 'is not null', I'd get a warning:

```
if (username is null)
   return new Uri($"http://mysite.be/users/{username}.png");
return null;
```

The warning says: "Return value must be non-null because parameter 'username' is non-null." With this typo, the implementation of this method wouldn't match what the NotNullOfNotNull attribute describes.

Postcondition

This category of attributes informs the compiler about the null state of a value after the method completes successfully.

The NotNull attribute is typically used on nullable method parameters in scenarios where we want to inform the compiler that the value of the parameter is not null when the method exits successfully. For example, we could rewrite the GetProfilePicture method we had before, so that it throws an exception when the given username parameter is null:

```
static Uri? GetProfilePicture([NotNull]string? username)
{
  if (username is null)
     throw new ArgumenException(nameof(username));

  return new Uri($"http://mysite.be/users/{username}.png");
}
```

Because of the NotNull attribute, the compiler will know that the username variable will not be null when the method completes successfully.

```
void GetProfilePicture(UserProfile profile)
{
  try
  {
     var picture = GetProfilePicture(profile.Username);
     Console.WriteLine(profile.Username.ToUpper());
  }
  catch
  {
  Console.WriteLine(profile.Username.ToUpper());
  }
}
```

The listing above shows that in the catch-block the compiler warns us about the fact that the Username can be null. We should do a null-check prior to accessing it. In the try-block, on the other hand, after successfully calling the GetProfilePicture method, the compiler doesn't warn us because, at that point, we know the value of Username is not going to be null.

As you might expect, the MaybeNull attribute can be declared on a non-nullable parameter that is passed to a method by reference which might be null when the method returns. Also, prior to C#9, the MaybeNull attribute was used a lot in combination with generics. In C#8 it isn't possible for an unconstraint generic type to be marked as nullable. Because of that, the only option to mark that an unconstraint generic return value could be null was by using the MayBeNull attribute.

In my personal experience, I don't use the MaybeNull attribute often. Instead of adding the attribute, I can just make the return type of a method nullable, which I can now do for every type since C#9 supports nullable unconstraint generics.

Precondition

Besides Conditional Post-Condition and Postcondition, there are attributes that form the Precondition category. With these attributes we can assign a value that doesn't match the nullable annotation of that variable.

Imagine a UserProfile class that has a Bio property. We want the bio property to be non-nullable, but the backing field, on the other hand, might be nullable (this could, for example, be the value that is persisted in the database). When the backing field is null, the Bio property returns a default value. Because the Bio property is not nullable, we cannot assign null to it without getting a warning. But on the other hand, we might want to be able to 'reset' the user's bio, so we actually want to be able to assign a null value to the Bio property.

This can be easily achieved by adding an AllowNull attribute:

```
string? bio;

[AllowNull]
public string Bio
{
  get => bio ?? "nothing";
  set => bio = value;
}
```

This allows us to assign a null value to the Bio property, whilst being assured we'll never get back a null value from it:

```
profile.Bio = null;

Console.WriteLine(profile.Bio.ToUpper());
```

The DisallowNull attribute is the same but different. Its purpose is to disallow assigning a null value to a nullable property. For example, a Product class can have a Rating property which is nullable, i.e. a product may not have a rating yet, but once it has a rating, it cannot be removed, it can only be updated.

```
Rating? rating;

[DisallowNull]
public Rating? Rating
{
  get => rating;
  set => rating = value;
}
```

With the DisallowNull attribute, we can have a property that can be null, but from code we are not allowed to assign a null value to it:

```
if (product.Rating is null)
{
  Console.WriteLine("no rating yet");
}
else
{
  Console.WriteLine(product.Rating.TotalStars);
}

product.Rating = null;
```

Unreachable code

Another pair of interesting attributes are the DoesNotReturn and DoesNotReturnIf attributes. Their purpose is to inform the compiler about unreachable code. They are particularity useful on Exception helper methods. Take a look at the following example:

```
public class Product
{
  Rating? rating;

  [DisallowNull]
  public Rating? Rating
  {
  get => rating;
  set => rating = value;
  }
  public bool RequiresCategory { get; set; }
  public Category? Category { get; set; }
}
```

```
public void Validate(Product product)
{
  if (product.RequiresCatagory)
  {
    if (product.Category is null)
    {
      ThrowUnexpectedValueExeption(nameof(product.Category));
    }
    var categoryId = product.Category.Id;
  }
}

[DoesNotReturn]
public void ThrowUnexpectedValueException
(string propertyName)
    => throw new ArgumentException(
       $"The value of {propertyName} is not correct!");
```

See how, under the if-block, we don't get a warning when accessing the Category property. That's because we do a null check and the ThrowUnexpectedValueException method is marked with a DoesNotReturn attribute. Hence, the analyzer can perfectly determine there is no other way we can reach the code below the if-block other than when the Category property is not null.

The DoesNotReturnIf attribute works the same way, but instead of the method never returning, we can let a bool parameter indicate if the method will return or not. If the value of attribute's argument matches to value of the associated parameter, it means the method will not return. If we want to use the DoesNotReturnIf attribute, we can refactor our code and use it like this:

```
public void Validate(Product product)
{
  if (product.RequiresCatagory)
  {
    ThrowUnexpectedValueExeptionWhen(product.Category is null,
        nameof(product.Category));
        var categoryId = product.Category.Id;
  }
}

public void ThrowUnexpectedValueExceptionWhen(
  [DoesNotReturnIf(true)] bool isInvalid, string propertyName)
    => throw new ArgumentException(
       $"The value of {propertyName} is not correct!");
```

Both attributes achieve the same thing and have the same effect, but they are used in a slightly different way and in different scenarios.

Helper methods

The last two attributes we need to discuss are MemberNotNull and MemberNotNullWhen. The first one is typically used on methods that are called from a constructor, to let the compiler know which members the method sets. When not assigning a non-nullable member from the constructor, the compiler will give you a warning. Unfortunately, when doing the assignment of a member in a method that gets called from the constructor, the compiler does not track this. By adding the MemberNotNull attribute to a method, we can tell the compiler which fields and properties are being assigned by the method. That way, the compiler knows that after calling this method, the defined properties and/or fields are set. If the developer forgets to assign one of the listed members inside the method, the compiler gives a warning.

This is very helpful in helper methods that set a particular state on the object.

```
public string Id { get; set; }
public Status Status{ get; set; }
public string? Username { get; set; }

public UserProfile()
{
  Init();
}

[MemberNotNull(nameof(Id), nameof(Status))]
private void Init()
{
  Id = String.Empty;
  Status = Status.New;
}
```

With the MemberNotNullWhen attribute we can tell the compiler that when a particular Boolean property or method returns the specified value, the value of another property or field will not be null. This allows us to create helper methods that give back a meaningful bool value about a property's null state, and after calling the method having the compiler infer the null state of said property in subsequent code.

```
[MemberNotNullWhen(true, nameof(Rating))]
private bool HasRating
  => Rating is not null;

private void PrintRating()
{
  if (CheckHasRating())
  {
    Console.Write(Rating.TotalStars);
  }
  else
  {
    Console.Write(Rating.TotalStars);
  }
}
```

See how we get a warning on the Rating property when the CheckHasRating method would return false, while we don't get one when the method returns true. That's all thanks to the MemberNotNullWhen attribute.

This attribute can also be applied to a property of type bool:

```
[MemberNotNullWhen(true, nameof(Rating))]
private bool CheckHasRating()
  => Rating is not null;
```

Opting-out

In some situations, you might want to (temporarily) op-out of using NRTs. By adding the '#nullable disable' directive, we can disable the nullable annotations context and have our code working like 'in the old days' i.e. reference types are nullable and are null by default. That is until the end of the file or until the compiler comes across a '#nullable enable' or '#nullable restore' directive. This can be very handy when copying external code from a blog post or from StackOverflow. Often code found online doesn't embrace NRTs and as a result, using that code in your codebase would potentially give a lot of warnings regarding nullability. In order to bypass these warnings, you might want to surround that code with the earlier mentioned directives. In an ideal world, after you've validated that code, you would want to refactor it, deal with NRTs and remove the directives again.

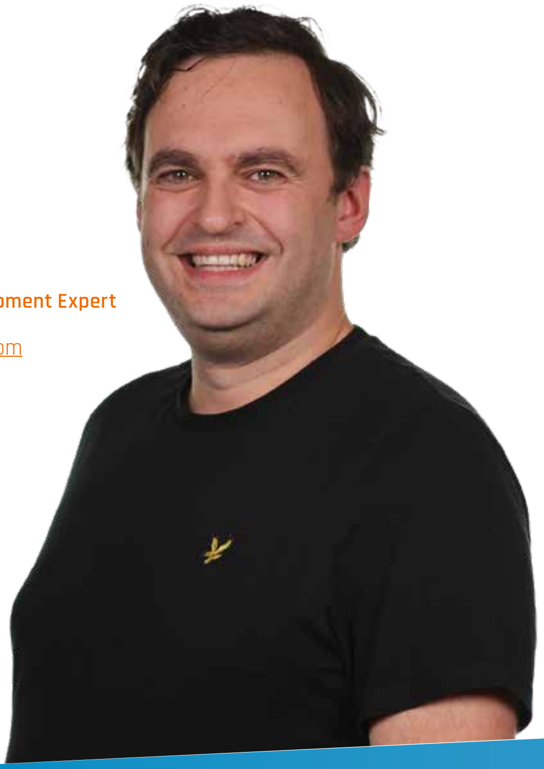
Just getting started...

I hope , after reading this article, you have a good understanding on how to deal with NRTs. It's a lot to digest, with new concepts to take in, and it requires you as a developer to be attentive about nullability and be more expressive and explicit. In all honesty, though, I've far from covered everything in this article. I barely scratched the surface. Things like deserialization of objects, integration with Entity Framework, making sure your objects are correct by construction, etc, require some special attention! But it's important to start with the core concepts.

Also be aware that working with NRTs, doesn't guarantee having no null reference exception anymore. Another developer might use the null-forgiving operator (!) to assign null to a non-nullable member, or pass null where the argument type is non-nullable. At least the developer knows at that moment they're doing something that will most likely fault the system. And when an exception occurs, it's their own fault. It's more subtle when deserializing an object that hasn't got a value for a non-nullable member. That's why it is recommended you still do null-checks in some cases. Especially on API endpoints to see whether the posted value is correct in terms of non-nullability.

Personally, I think NRTs is one of the greatest features introduced in the last couple of years in .NET. Despite it having a pretty high threshold to fully embrace, it defiantly makes your code a lot better in many ways, in my opinion.

Please do reach out if you have any more questions about NRTs! </>



Pieter Nijs
Mobile Development Expert

pnijs@xpirit.com

Microsoft Most Valuable Professional

Introduction to Blazor

Creating a beautiful and functional website is something we can easily do nowadays because of one of the many existing frameworks. A question to ask is, why Blazor? Is it yet another framework to do the same? What are the benefits? When should you use it, and as important, when should you not use it? In this article, we will answer these questions and show all the different flavors Blazor has to offer.

Author Mark Foppen

What is Blazor?

Blazor is the response from Microsoft to enter the competition with other Single Page Application (SPA) frameworks such as React, Vue.js, or Angular.
The definition of Blazor according to Microsoft:
"Blazor lets you build interactive web UIs using C# instead of JavaScript. Blazor apps are composed of reusable web UI components implemented using C#, HTML, and CSS. Both client and server code are written in C#, allowing you to share code and libraries."[1]

The scope of Blazor is very wide.
You can use Blazor for building a new frontend as a single page application (SPA) or server rendered application, a mobile application or to modernize your WPF or Windows Forms application.

Blazor is built on top of ASP.NET and is therefore not an entirely new framework. It uses C# .NET and because it is .NET, it opens many doors. Developers can use NuGet libraries to speed up development. NuGet is the equivalent for NPM that is used in many JavaScript-based frameworks. NuGet allows you to (re)use shared libraries and other business logic already written.
This can increase your development speed or make the transition from a classic ASP.net frontend to a Blazor project much smoother. Blazor is also part of the open-source .NET platform with a strong community and active contributors.

In addition to sharing code, you can also share components. All the user interface-related code is written using the Razor syntax. This allows the development of small components that are re-used throughout your application the same way you share C# code through libraries.

In the statement from Microsoft, it appears they are trying to replace JavaScript. This is not true. In Blazor, you can still use JavaScript and all the libraries available through NPM. If you have an amazing JavaScript library, you can and should use it to speed up your development. Through Blazor JavaScript Interop, you can have full control over a JavaScript library.

One of the many questions we get about Blazor is if it is ready for production purposes. Yes, Blazor is production ready. In fact, it has been production ready since 2018. There was only Blazor Server which used a lot of components that were already production ready. Blazor server consists of a combination of technologies like SignalR, Razor syntax, and .NET Core ASP.NET. From the start, Blazor was very mature. This is not the case for all the different hosting models of Blazor such as WebAssembly or MAUI, which we will dive into in more detail later in this article.

What is Blazor not?

Blazor is not a React killer. It is a new framework based on ASP.net that has been around for over 15 years. It targets different audiences to use it, for example, where React uses Typescript, Blazor uses C#. The larger initial page load for webassembly or the required WebSocket for Blazor server that React or Vue.js does not have. Blazor has its own strengths which will be explained later in this article. Use Blazor when it fits your requirements, but also make sure to use React, Vue.js, and other frameworks when appropriate.

Four hosting models of Blazor

At the time of writing, Blazor has four different hosting models:

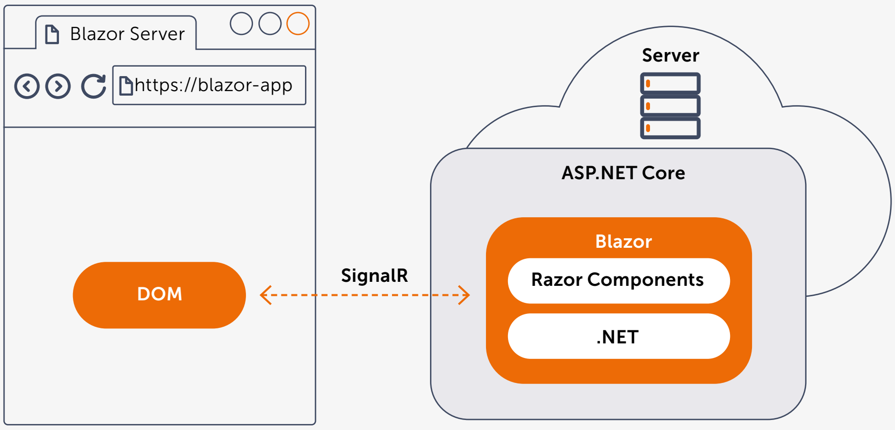
- Server
- WebAssembly
- Hybrid with .NET MAUI
- Custom Elements (React/Angular components)

More information on hosting models can be found here: https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models

[1] Blazor documentation https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

Blazor Server

This hosting model is all about executing a Web App on a server and serving that content as fast as possible. Your first page visit downloads a small set of files and opens a WebSocket with SignalR to the backend. From then on, all communication to the server is sent over a SignalR connection. User interaction results in a trigger to the backend, which returns only a fragment of the page to be re-rendered. This re-render is a change in the Document Object Model (DOM) in the current viewed page. For the user of the application this will update the user interface.



Because the application is being hosted on an ASP.NET webserver, you can use any .NET API available.
"Why all the other options? This sounds like the silver bullet I was looking for!"

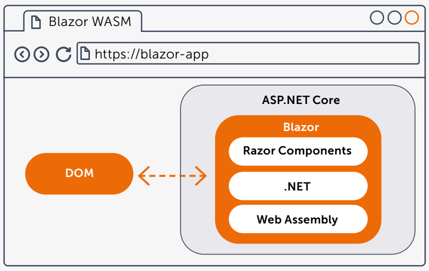
Within Blazor Server a few so-called "render modes" can be chosen as well. The render mode tells the server how pages should be rendered and sent to the browser. The render modes Blazor server supports are Server, Server Prerendered and Static.

When using the render mode Server, only a small html file with the Blazor server scripts and some metadata is sent to the client. It can, however, lead to SEO (search engine optimization) issues such as lower ranking because search engine crawlers will not execute JavaScript and therefore not initiate Blazor. It is up to the requirements of the project to determine if this is acceptable. For most LOB (line of business) applications this is not an issue.

If the server has to render the first page load instead of the local browser, then Server Prerendered can be used. This will render the entire page on the server, just like an asp.net (MVC) website, before sending it back to the client. For any search engine crawler this is the same as any static page. This is also a way to solve the pre stated SEO issue. The caveat to this is your page will load twice. One time for the initial load and again after the initialization of Blazor and setting up the SignalR connection. In this case, consider adding data caching so the user will not notice.

Blazor Web Assembly

Blazor Web Assembly allows you to move from everything on the server to everything in the browser. Its underlying technology is Web Assembly (as described in Xpirit Magazine #12 "What's what with WebAssembly?"). Of course, there needs to be a webserver somewhere that hosts the files and serves them to the browser. This can be done, for example, with an Azure storage account and static website enabled and served via a CDN (Content Delivery Network).



By doing so, you gain some new features a PWA (progressive web application) has, for example, the offline capability and the ability to be installed as app.

Since entire website is running from within the browser, the browser is responsible for rendering the website. This enables you to also view the website when you are offline by caching your data. Keep in mind though that your device is responsible for everything and because of that the user experience is dependent on the performance of that device. This goes hand in hand with loading your website, which takes longer, since it first needs to download all the files (typically 2MB with hello world) before it can start Blazor and render the website. Be careful with adding NuGet Packages, because this can have an enormous impact on your users, especially for those with a slow internet connection.

One of the key benefits of putting everything on a CDN is that you do not have to pay for the compute of your servers but only for the storage and egress of the files. If you have a lot of concurrent users, this can drastically reduce the bill. When hosted in a CDN you will also gain global distribution of your application.

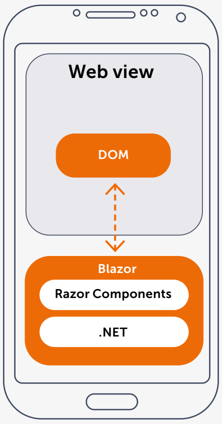
To be fair, most websites still need some sort of API to be able to function. Currently there are a lot of options to choose from. Your API could be a .NET minimal API hosted on an Azure service plan, Logic app, Azure function or any other http-based API.

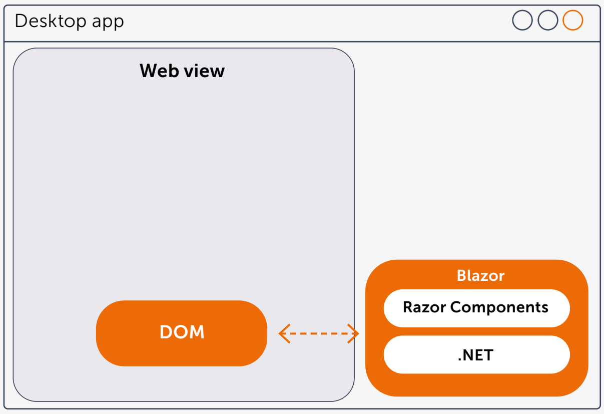
If the application is getting bigger by adding a lot of third-party libraries, the load times with Web Assembly will increase. To counter this effect there is an option to do prerendering, just like with Blazor Server. In this case a server is needed, but it will only be responsible for the initial load of the site. This will cause the page to load twice. You saw this with the Server hosting model as well. Keep in mind that client-side caching is needed to prevent long loading times on the second load.

We have seen the two most extremes. Either run your code 100% on the Server, or 100% on the Client. Let us continue and find out what the other options are.

Blazor Hybrid with .NET MAUI

When building applications with Blazor, you have the option to add your components and pages to a separate library (Razor Component Library). This library can be used to create an application with MAUI (.NET Multi-platform App). MAUI is the successor of Xamarin Forms and a cross-platform framework for creating native mobile and desktop applications with C#.





This means if you are building a solid responsive UI, you can reuse all those components and even pages in the mobile app. The mobile app works like Blazor Web Assembly but Internally it has a WebView where it renders the UI. Because it is rendered on the device, you are not loading an external website in the WebView like many other frameworks do.

An added benefit for this hybrid hosting model is the capability to access device native features like location, notifications, connectivity status, etc. You can even make the decision to combine Blazor with Native MAUI mobile development through Xaml.

Blazor Custom Elements (Experimental)

While this all sounds great, I can hear you say: "I'm not starting from scratch, so now what?" If you have an existing React or Angular application, there is a hosting model to embed Blazor components. React is specifically mentioned, but everything mentioned is also available for Angular.

At first this didn't sound right. Mixing multiple frameworks that do the same thing might lead to increased complexity without delivering any real value. Therefore, this hosting model should be seen as an exit strategy. Choose to migrate the application to Blazor instead of running it side by side.

With Blazor Custom Elements you have the option to host Blazor components on a server. Currently it can only host components and not pages. Within your existing React application you can add a component by using the libraries provided by Microsoft. These libraries form a bridge between Blazor and React. This allows for migrating the existing application component by component without needing to do everything in one big bang. The bridge between Blazor and React is two way and will preserve state. This hosting model is still in an experimental state at this moment.

When to use Blazor?

Blazor is currently in general availability (GA) which means it can be used for production workloads. Because of the different hosting models, it is particularly important to get the requirements first. If you build a Facebook-like product and expect hundreds of millions of users, then maybe Blazor is not your framework. Except for those extreme use cases, Blazor can be used for any project. You can easily switch between hosting models if you set it up the right way with a shared library for pages and components.

The other factor to consider is the type of developers available. C# developers with html knowledge can create working solutions amazingly fast. Developers coming from ASP.net with razor pages will have a major advantage since they already know C# and the razor syntax is almost the same as Blazor. But for non C# developers it can be a steep learning curve and Blazor might not be a good fit.

What is next for Blazor?

Now we know what Blazor is, what hosting models there are, and when you should and should not use it. Where do we see the future of Blazor? Blazor is currently a robust and production-ready framework that can adapt to many scenarios. The development speed is fast in our experience, and you can create business value right from the start. The most important part is to see if a particular Blazor hosting model matches your requirements. There is no one size fits all.

There are some shortcomings in its current state that will be addressed in future versions such as[2]:

- Running multiple Blazor apps in one SPA
- Pre-rendering performance
- Multi-threading for Web Assembly
- MAUI's hot Reload, performance, and authentication features.

These and many other features will be implemented in the next releases of .NET. We think it will make Blazor better match your requirements and enable teams to create solutions blazing fast. </>

Mark Foppen
Developer

mfoppen@xpirit.com

[2] Blazor Roadmap https://github.com/dotnet/aspnetcore/issues/39504

Real world mocking! Http Service testing in C# using Wiremock.Net

Writing tests is hard by itself, often it is forgotten that there is a need for different kinds of tests. There are unit tests that cover a specific functionality and these tests are scoped clearly. There are system tests that cover a bunch of functionalities but these tests replace the external systems with fake ones. There are also Integration tests - which are like system tests - but this time external systems are involved in the test process.

Authors Bas van der Sande and Kristof Riebels

Maintaining Integration tests is the hardest part when it comes to maintaining tests, there are a lot of dependencies that are not necessarily under your control. Furthermore, it is not easy to incorporate the integration tests into the CI-pipeline.

Each type of test has its own difficulties. In this article the tests that are going to be discussed are the system and integration tests. WireMock.Net is going to play an important role in converting the hard to maintain integration tests into controllable system tests and vice versa.

What is WireMock.Net?

WireMock.Net is a GitHub community project[1] and contains the C# implementation of mock4net which mimics the functionality from the JAVA based WireMock.org.

The idea behind WireMock.Net is to mimic the behaviour of a real-life HTTP API. HTTP requests, that are made from the code that is tested, are captured and sent to a WireMock.Net HTTP server (which is part of the testing framework) and as a result an HTTP response is returned that can be verified against an expected behaviour.

Which features are offered by WireMock.Net?

One of the most interesting features of WireMock.Net is that it can be used in Test projects. It can record and playback captured messages. In integration tests WireMock.Net can be setup to act as a proxy in order to capture and/or forward the HTTP requests. WireMock.Net can also be configured to give the matching response when it sees similar requests. This means that integration tests can be turned into system tests and vice versa. Of course, assertions can be written for those incoming requests.

In case the requests are too dynamic, request matching is the technique to be used to generalize incoming requests. Requests can be matched by URL, path, request method, request header, cookies and/or request body.

WireMock.Net can also be used when manual testing is needed but when the dependencies are not ready or are unavailable on the testing environment. It can be run as a standalone tool.

[1] https://github.com/WireMock-Net/WireMock.Net

Hello world!

With the understanding of what WireMock.Net is all about, let's see how it can be set up using XUnit.

The example below describes a simple test method that sets up a wiremock.net server. It defines a request and a response. When a Get-request with path "/foo" is sent to the WireMock-server, then a response will be created with status code OK and with content "bar".

```
public class GivenAWireMockServer
{
  [Fact]
  [Trait("Category","SystemTests")]
  public async Task WhenSendingAGetRequestTo_foo_
  ReceiveAResponse_bar()
  {

    //Arrange
    var wireMockServer = WireMock.Server.WireMockServer.Start();
     wireMockServer.Given(Request.Create()
        .UsingGet()
        .WithPath("/hello")
    ).RespondWith(Response.Create()
         .WithStatusCode(HttpStatusCode.OK)
         .WithBody("world"));

    var httpClient = wireMockServer.CreateClient();

    //Act
    var barResponse = await httpClient.GetAsync("hello");
    var body = await barResponse.Content.ReadAsStringAsync();

    //Assert
    Assert.Equal(HttpStatusCode.OK, barResponse.StatusCode);
    Assert.Equal("world", body);
 }
}
```

Of course the scenario above is to show the simplicity of setting up a test. Let`s see how it can be used as an integration test.

Test scenario

The following code represents a WeatherForecastController with a Get method. Browsing to http://localhost:3011/weatherforecast results in one of the following messages: "Too hot", "Cozy", "Cold" or "Too cold".

The method consumes a weather forecasting service https://api.open-meteo.com/v1/forecast. In order to consume that service the WeatherForecastController is dependent on the IOpenMeteoClient that is depend on the IHttpClientFactory.

The httpClientFactory creates an HttpClient that will get the forecast from the weather forecasting service. Based on the result, the method in the WeatherForecastController returns one of the messages defined above. The OpenMeteoClient is actually a proxy that helps to hide the fact that an HttpClientFactory is used.

Define the first integration test

Before an integration test can be defined, it is needed to know what messages and responses go back and forth over the line. Fiddler can be used to intercept the traffic from the WeatherForecastController to the open-meteo api.
The actual request and the response looks like the following:

```
request
GET https://api.open-meteo.com/v1/forecast?latitude=
51.09&longitude=4.06&daily=temperature_2m_max,temperature_
2m_min&current_weather=true&timezone=Europe%2FBerlin&
start_date=2022-07-31&end_date=2022-07-31 HTTP/1.1
Host: api.open-meteo.com
traceparent: 00-1e25de5aeaa91dba70b47f8679ea7dc9-
d74d7ed281f40bdb-00

response
HTTP/1.1 200 OK
Date: Sun, 31 Jul 2022 15:13:41 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

{"latitude":53.1,"longitude":5.06,"generationtime_ms":
0.38301944732666016,"utc_offset_seconds":7200,"elevation":
3.0,"current_weather":{'temperature":22.5,"windspeed":22.8,
"winddirection":249.0,"weathercode":3.0,”time":
"2022-07-31T17:00"},"daily_units":{"time":"iso8601",
"temperature_2m_max":"°C","temperature_2m_min":"°C"},
"daily":{"time":["2022-07-31"],"temperature_2m_max":[23.6],
"temperature_2m_min":[17.0]}}
```

When using WireMock.Net to do the integration test, a mocked IhttpClientFactory is set up to return the HttpClient of the embedded WireMock.Net Server. The WireMock.Net Server runs in the same process but it is reachable from the "outside" as well.

A basic integration test of the Get method on the WeatherForecastController from the test scenario, can look like this:

```
[Fact]
[Trait("Category","IntegrationTests")]
public async Task WhenRequestingCurrentWeatherInformation_
DateShouldBeUtcToday()
{
   //Arrange
   var application = new WebApplicationFactory<Program>()
         .WithWebHostBuilder(builder =>
         {
           builder.ConfigureServices(
              services => services.AddConfiguredServices());
         });

   //Act
   var httpClient = application.CreateClient();

   //Assert
   var response = await httpClient.GetAsync("/WeatherForecast”);
     Assert.Equal(HttpStatusCode.OK, response.StatusCode);
     var weather = await response.Content.ReadFromJsonAsync
     <WeatherForecast>();

 ...
}
```

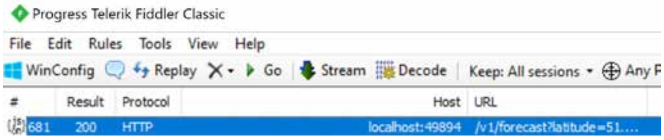
This integration test tests, if a filled WeatherForecast response was returned from the api and if the date of the weather forecast matches with the local system date.

Integration test: using the Request Matcher

The code below shows setting up the request matcher and defining the corresponding response. The matching is setup to be fairly generic but can be setup as strictly as desired.

```
public async Task WhenRequestingCurrentWeatherInformation_
DateShouldBeUtcToday()
{
  //Arrange
  var openMeteoWireMockServer = WireMock.Server.WireMock-
  Server.Start();
    openMeteoWireMockServer.Given(Request.Create()
      .UsingGet()
      .WithPath(path => path.Contains("forecast"))
  ).RespondWith(Response.Create()
      .WithStatusCode(HttpStatusCode.OK)
          .WithBody( ..."current_weather\":{\"temperature
           \":22.5,\"windspeed\":22.8,\"winddirection\":249.0,
           \"weathercode\":3.0,\"time\":\"2022-07-31T17:00" ...));

  var openMeteoHttpClient = openMeteoWireMockServer.
  CreateClient();

  var fakeHttpClientFactory = new Fake<IHttpClientFactory>( );
  fakeHttpClientFactory.CallsTo(httpClientFactory =>
  httpClientFactory.CreateClient("OpenMeteo"))
      .Returns(openMeteoHttpClient);

  var application = new WebApplicationFactory<Program>()
      .WithWebHostBuilder(builder =>
      {
          builder.ConfigureServices(
              services =>
              {
                  services.AddConfiguredServices();
                  services.AddScoped(provider =>
                  fakeHttpClientFactory.FakedObject);
              });
      });

  //Act
  var httpClient = application.CreateClient();

  //Assert
  var response = await httpClient.GetAsync("/WeatherForecast");
  Assert.Equal(HttpStatusCode.OK, response.StatusCode);
  var weather = await response.Content.ReadFromJsonAsync
  <WeatherForecast>();
 ...
}
```

Open Fiddler, execute the test and notice that there is a call from the WeatherForcastController to the open-meteo api. Instead of accessing the original host at the Url *"https://api.open-meteo.com/v1/forecast?lattitude=51...",* the host being called is the WireMock.Net server, running at localhost listening to port 49894. The Url passed in contains the part "forecast", and that will result in a Http status code 200, with the Http body that was defined in the test.


Progress Telerik Fiddler Classic
File Edit Rules Tools View Help
WinConfig Replay Go Stream Decode Keep: All sessions Any P

#	Result	Protocol	Host	URL
681	200	HTTP	localhost:49894	/v1/forecast?latitude=51....

Integration test: working with recorded messages

A new test is set up in which recorded messages are being used. In this scenario WireMock.Net will play these messages back if the matching request comes in. Fiddler is used as proxy server to monitor what is going on.

In order to prepare the test to work with recorded messages, the response messages need to be captured initially. The code below shows how to record the messages. The recorded messages are stored in the local debug folder.

```
//Arrange
var openMeteoWireMockServer = WireMock.Server.WireMockSer-
ver.Start(
new WireMockServerSettings()
{
    ProxyAndRecordSettings = new ProxyAndRecordSettings()
    {
        Url = "https://api.open-meteo.com",
        SaveMapping = true,
        SaveMappingToFile = true,
        WebProxySettings = new WebProxySettings()
        {
            Address = "127.0.0.1:8888"
        },
        ExcludedHeaders = new string[]{"Host", "traceparent"
}
     },
         StartAdminInterface = true,
         FileSystemHandler = new LocalFileSystemHandler(".")
});

var openMeteoHttpClient = openMeteoWireMockServer.
CreateClient();
```

Open Fiddler and execute the test. Notice that there is an actual call forwarded to the open-meteo api.

The mappings are recorded in the project's debug folder as shown below.

Once the initial recording is done, the playback mechanism can be used as shown in the example below. For more advanced use cases, the mapping model can be opened, split or organized. In the example below WireMock.Net will be handling the mapping model the way it was recorded.

```
//Arrange
var openMeteoWireMockServer = WireMock.Server.
WireMockServer.Start(
new WireMockServerSettings()
{
    ReadStaticMappings = true
});

//Act
var openMeteoHttpClient = openMeteoWireMockServer.
CreateClient();
 ...


//Assert
 ...
```

Using WireMock.Net in a CI pipeline

WireMock.Net can be used for automated testing in GitHub workflow actions as well. In the pipeline example below a simple GitHub workflow action is setup that will restore, build and test the code. The following yaml is an example.

```
name: .NET
  on:
    push:
      branches: [ "main" ]
    pull_request:
      branches: [ "main" ]
  jobs:
    build:
      runs-on: ubuntu-latest
      steps:
      - uses: actions/checkout@v3
      - name: Setup .NET
        uses: actions/setup-dotnet@v2
        with:
          dotnet-version: 6.0.x
      - name: Restore dependencies
        run: dotnet restore ./src/WiremockSamples.sln
      - name: Build
        run: dotnet build ./src/WiremockSamples.sln
--no-restore
      - name: Test
        run: dotnet test ./src/WiremockSamples.sln --no-
build --verbosity normal --filter "Category=SystemTests"
```

The tests have the attribute Trait with key "Category" and value "IntegrationTests" or "SystemTests". When running all tests, the pipeline will fail because the IntegrationTests are actually trying to contact the actual open-meteo api. This api can not be reached from the the CI server. In order to overcome this, a filter is applied to ensure that only "SystemTests" are executed. Let's see what happens if the pipeline has been executed. The pipeline results show that the system test ran successfully.

Summary

This article scratched the surface when it comes to testing with WinMock.Net. There is much more to discover with WireMock.Net, but the above will help to setup integration and system tests in any .NET project that is using Http based services.

The WireMock.Net Wiki[2] pages at GitHub are self-explanatory. The repository contains loads of samples in which the most common scenarios are covered. The ease of use to test Http based services with real requests and responses makes it worthwhile to revaluate existing unit tests. Instead of mocking entire functions just the internal Http requests and responses can be mocked, giving an additional level of control in the software testing process. The sources in this article are shared on GitHub as well[3]. </>

[2] https://github.com/WireMock-Net/WireMock.Net/wiki

[3] https://github.com/kriebb/Wiremock.net

Kristof Riebbels
Software Developer /
Coach / DevOps

kriebbels@xpirit.com

Bas van de Sande
Azure Coding Architect /
Consultant | Integrator /
Trainer

bvandesande@xpirit.com

Scaling application security with codified security knowledge

It is an exciting time to be working on application security. With more information, personal information, being processed by applications the stakes of getting security right increases each day with each new release. We have come a long way with the availability of amazing static analysis tooling that scan for well-known issues the minute you push your code. With security being the responsibility of everyone, automation is key to supporting everyone with mitigating security issues before they reach production.

Author Remco Vermeulen (GitHub)

However, there are many challenges with automating security. Today we are going to look at an opportunity to take a step forward in one of those challenges, scaling security knowledge.

Security knowledge is discovered when application security practitioners audit applications, so let's start with a security audit.

Application security audit

Security audits come in various shapes and forms from fully automated to manual audits performed by security specialists. We are going to look at an example of the latter, the one that in my opinion generates the most useful security knowledge, but is also the one that does not scale well.

First, you start with mapping the attack surface, the entry points, the way others interact with the application. Then, determine what can be influenced by looking at what is accepted as input, how it flows through the application, and where it is being used. An SQL query is constructed using string concatenation without the proper contextual output encoding. Tracing back the flow reveals no sign of input validation, but the endpoint requires authentication.
Still, a serious security vulnerability.

The flow just described is common in manually auditing applications for security issues. The other part is looking for patterns learned from previous audits or security research published by other application security practitioners. Manual security audits are typically done once or twice a year per application. The knowledge obtained during an audit is mostly lost in notes and the findings are written down in a report that hopefully ends up in the hands of those that can mitigate the discovered issues, the developers.

Automatic vs manual

Performing a security audit once or twice a year is obviously not enough in a time where development teams push a new release to production once per week, once per day, or even multiple times per day. This is already well known and the shift left movement is attempting to address this gap between releases and security audits. Manual security audits, however, demonstrate that there remains a different security gap i.e. issues not being mitigated by developers.

This security gap that I encountered during the many manual security audits that I performed was interesting.
While trying to understand why this happens, I encountered the three following reasons:

1. The developers didn't look at or weren't aware of the results found by the static analysis tooling integrated into their CI/CD pipelines because the result page was not part of their development workflow. It required a visit to a separate dashboard.

2. Developers were provided with a tremendous task of sifting through a lengthy list of findings, many not actually an issue and known as a false positive. At one point, developers simply started to ignore the alerts.
3. The analysis was missing significant findings, known as false negatives, because the analysis didn't understand the domain of the application nor its threat model.

So, what can we do to reduce the gap between audits and automated security analysis? Today we are going to have a look at reason 3.

Scaling security knowledge

The key is still automation. DevOps practices provide a unique opportunity for security to join the effort in reporting security issues as soon as possible. Manual security audits don't scale. Even if your organization has a team of security engineers, it is still tough when you have 100,000 repositories. Ignoring many aspects that impact the scaling of application security, rolling out a static analysis tool across your organization is non-trivial. We are going to look at an approach to scale security practitioners tasked with auditing applications for security issues.

The way we can scale is by codifying their security knowledge.

Codifying security knowledge

Static analysis tools have been around for ages. However, a new generation of static analysis tooling is stepping it up a notch by being easily programmable. Two of which have managed to establish a community of security practitioners that are using it while making their efforts available to the rest of the world. The first one is Semgrep[1], a lightweight static analysis solution for finding variants of security bugs in software. The other, and the one discussed in this article, is CodeQLf[2].

What makes CodeQL unique is it turns source code into data that can be queried using a query language named QL (which stands for query language). QL is a language that looks like SQL, but its semantics is based on a declarative logic language called Datalog[3]. This makes QL a logic language and all the operations are logic operations. So, what does this all mean and how does this benefit us?

QL allows you to codify code patterns in a way that is very succinct and easy to read. The reason it can be succinct is due to its declarative nature. If you have experience with writing SQL queries then you are already familiar with this. It means you state what you want to find instead of how. The following examples will demonstrate this. QL is a generic purpose language, but with CodeQL there are libraries available that implement many program analysis algorithms that enable you to analyze programs.

Let's start with an example, a "hello world" in QL.

```
select "Hello World"
```

Short and to the point, but not very insightful. Let's have a look at something more useful. Assuming a Java program for which we have used CodeQL to build a database of facts we can, for example, ask the following question.

```
import java

from IfStmt ifStmt
where ifStmt.getThen().getNumStmt() = 0
select ifStmt, " ..."
```

Take a minute to determine if you can fill in the "...", a message to the user, by reading the query.

If you would translate the query to English, it would state something like - given all the if statements in the source code represented by the class IfStmt[4], let the variable ifStmt represent those that have zero statements in their "then" branch. In other words, an if statement with an empty "then" block. The message "Redundant 'if' statement." would be an appropriate alert message instead of the "...".

Nice, but weren't we interested in security knowledge? Remember the audit flow in the beginning of the article where we started from an entry point and followed it to a security sensitive operation? In program analysis there is an analysis called data flow analysis that can follow how data is used in a program. By giving it a start location in the program we can determine if that data reaches another location in the program. The former we call a source and the latter we call a sink.

Let's have a look at what a simplified query would look like.

```
import java
import semmle.code.java.dataflow.FlowSources

class SqlInjectionConfig extends TaintTracking::Configuration {
  SqlInjectionConfig() {
    this = "SqlInjectionConfig"
  }

  override predicate isSource(DataFlow::Node node) {
    node instanceof RemoteFlowSource
  }

  override predicate isSink(DataFlow::Node node) {
    exists(MethodCall call | call.getTarget().getName() =
    "executeQuery" | call.getAnArgument() = node.asExpr())
  }
}

from SqlInjectionConfig config, DataFlow::Node source,
DataFlow::Node sink
where config.hasFlow(source, sink)
select sink, "Possible SQL injection because this query
relies on $@.", source.getNode(), "user-supplied data"
```

[1] https://github.com/returntocorp/semgrep
[2] https://codeql.github.com/
[3] https://en.wikipedia.org/wiki/Datalog
[4] https://codeql.github.com/codeql-standard-libraries/java/semmle/code/java/Statement.qll/type.Statement$IfStmt.html

There is much more going on, but if you focus on the from … where … select part, you can see that it looks for a data flow between a source and a sink. The definition of the source and the sink is provided by a configuration. The source is defined by a predicate isSource that only holds if a data flow node is part of the set of values represented by the class RemoteFlowSource[5]. This is a class provided by the standard library of CodeQL for the JAVA language that represents all the elements in a program considered a source of user-supplied data. In other words, data an attacker can influence and which should be scrutinized. An example is a parameter in a HTTP request such as the parameter email in the following Spring REST endpoint method.

```
@GetMapping("/user")
@ResponseBody
public User getUserByEmail(@RequestParam String email) {
  …
}
```

While many details remain unexplained, I hope this shows that a complex analysis, tracking data through a program, can be expressed in a succinct manner that is also readable and understandable. This is a desirable property of a system used to capture knowledge.

What is neat is that you do not have to know the details of the class RemoteFlowSource and that any extensions (i.e., new HTTP frameworks) will automatically be available to your query. The SQL injection query[6] injection query provided with CodeQL does the same for the sinks. Any new SQL library with methods that are susceptible to injection can be added and no change to the query is required to find issues in applications that use those new libraries in an insecure manner.

With this a security practitioner can write a query to find an issue that they already found manually. How does this help scale application security?

Variant analysis

A strategy applied by security practitioners before they start auditing software is to look for known prior security issues. This not only provides them with useful information on the target, but it is common that a fix is incomplete, or the issue follows a pattern that occurs elsewhere in the application. Possibly in other applications as well. It is therefore interesting to look at variants of a security issue.

Looking for variants is a difficult process if done manually, but when the pattern is codified it becomes much easier to look for the same pattern in the same application or across all your applications. It also helps with preventing regressions if the same mistake is made again or if a mitigation is inadvertently reverted.

Being able to look for variants is super useful, but what really helps reduce the security gap is that the knowledge being codified is domain specific. The queries written for your application allow static analysis tooling to find issues that might only occur in your application. For example, an incorrect use of a proprietary framework. These kinds of issues are never found out of the box by any static analysis tooling!

How cool would it be if, at the end of security audit, you would not only get a report, but also a set of queries that you can run in your CI/CD pipeline to find the same issue and variants in all your applications. It can even help others if those queries are made available. Two notable examples are:

› A ZipSlip query to find a widespread security issue documented by Snyk[7] that allowed security teams to quickly assess if their projects are vulnerable.

› Multiple queries to hunt for Solarigate activity described in the blog post Microsoft open sources CodeQL queries used to hunt for Solarigate[8].

Conclusion

Codifying security knowledge by security practitioners is the next step in reducing the security gap we currently have with static analysis tooling. It will help with scaling application security by making it easier to apply the knowledge repeatedly across many codebases. This is a capability you will need to help developers and security engineers to respond to unknown application security threats.

You can start learning how to codify security knowledge today through fun CodeQL Capture The Flag[9] exercises made available by the GitHub Security Lab. </>

Remco Vermeulen
CodeQL Analysis Engineer
GitHub Expert Services

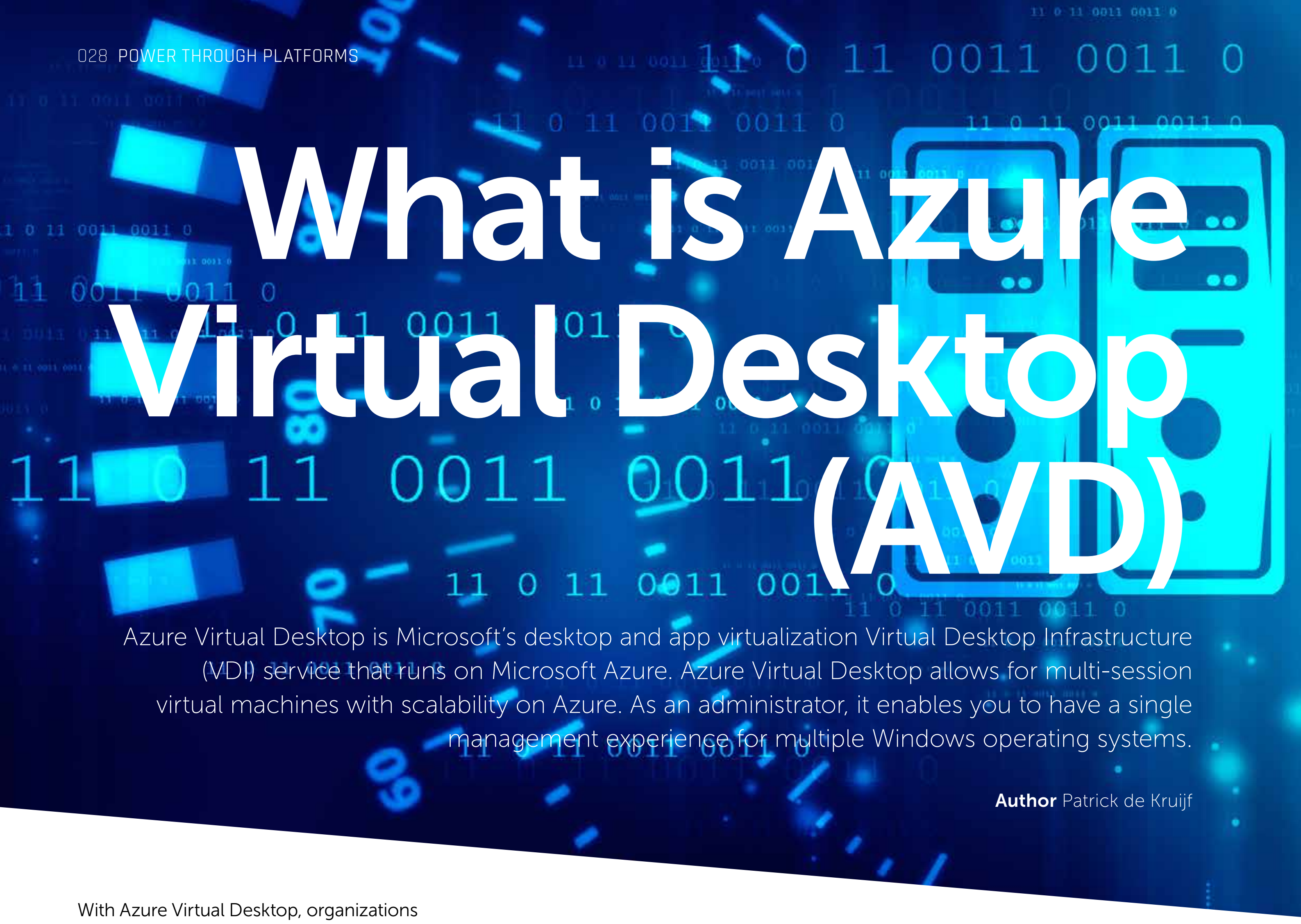
[5] https://codeql.github.com/codeql-standard-libraries/java/semmle/code/java/dataflow/FlowSources.qll/type.FlowSources$RemoteFlowSource.html

[6] https://github.com/github/codeql/blob/main/java/ql/src/Security/CWE/CWE-089/SqlTainted.ql

[7] https://security.snyk.io/research/zip-slip-vulnerability

[8] https://www.microsoft.com/security/blog/2021/02/25/microsoft-open-sources-codeql-queries-used-to-hunt-for-solorigate-activity/

[9] https://securitylab.github.com/ctf/



What is Azure Virtual Desktop (AVD)

Azure Virtual Desktop is Microsoft's desktop and app virtualization Virtual Desktop Infrastructure (VDI) service that runs on Microsoft Azure. Azure Virtual Desktop allows for multi-session virtual machines with scalability on Azure. As an administrator, it enables you to have a single management experience for multiple Windows operating systems.

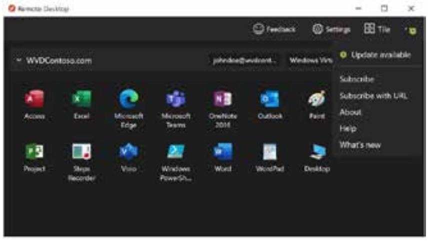
Author Patrick de Kruijf

With Azure Virtual Desktop, organizations can centrally provide users, both internally and externally, with a full desktop and applications hosted in the Microsoft Azure cloud. This allows organizations to leverage the scalability of the Microsoft Azure cloud for your desktops and desktop applications.

Some key features that Azure Virtual Desktop brings organizations:

- Gateway functionality included in the default Azure Virtual Desktop setup
  You do not have to worry about the connections to the Azure Virtual Desktop machines; this is all integrated into the service.
- You can use your own images from your shared image gallery
  Using your own images will accelerate your machine readiness. You can even share images across your services and even Microsoft Azure tenants.
- Option to create pooled (shared), personal (private) machines, and a mix of both
  The option to choose the machine types gives you the freedom to supply Azure Virtual Desktop machines based on your user requirement.
- Autoscaling to increase or decrease resources based on time of day, days of week, and on-demand
  Scaling itself gives you the flexibility to supply the right amount of machines that you need at any given moment. Adding automation will ensure that you do not have to worry about your capacity.

Azure Virtual Desktop can be used with either the Remote Desktop Client (available on multiple platforms) or the HTML5 web client.



Azure Virtual Desktop use cases

Azure Virtual Desktop is the cloud version of Remote Desktop Services (RDS), a Virtual Desktop Infrastructure service. A Virtual Desktop Infrastructure service is commonly used for a fully working business desktop with all the required applications for your business. This means that it can create a fully working workplace for your employees or contractors in the cloud without having to invest in physical hardware that is capable of running everything locally.

A well-known provider of Virtual Desktop Infrastructure services is Citrix (Virtual Apps & Desktop). Citrix is a more advanced solution that provides more advanced options for a Desktop-as-a-Service (DaaS) experience. Being more advanced also means that more setup and configuration effort is required to make it work.

Since Azure Virtual Desktop can also host and present an application to end-users and give them a seamless user experience (you experience the application as if it runs locally on your machine), this will mean that you, as a business, can distribute your application online with the scalability and flexibility of the Azure cloud and the control of your application. Although you could use this as an extension of your full desktop experience, this is also the basis for a niche use case.

What we mean with a niche use case for Azure Virtual Desktop is that your organization can provide centrally managed applications to your end-users, even though your application might not be fully cloud-native (yet). This use case could help organizations move to the Microsoft Azure cloud.

Azure Virtual Desktop and our customers

Some customers have an application that is not cloud-native (yet), some customers want control over the installation and configuration of their application, and some want to not burden their customers with application configuration or setup.

We have encountered these different reasons at our customers, and we have helped tackle them with Azure Virtual Desktop. The applications are still being improved to be cloud-native, but while the development and improvements are performed, we can move to the cloud quicker while adding scalability and flexibility in the Microsoft Azure cloud.

How Kongsberg Digital leverages Azure Virtual Desktop

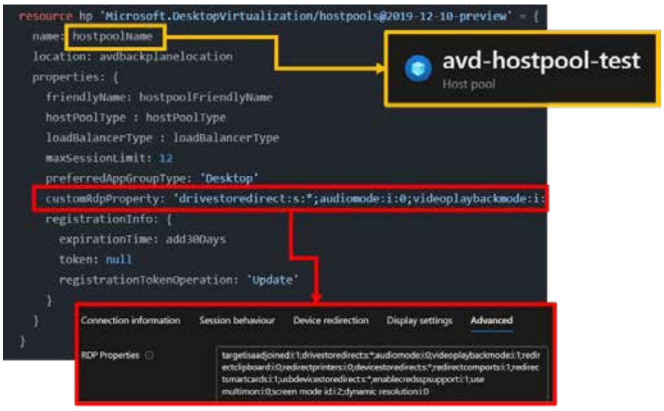
Among other industries, Kongsberg Digital is a key player in the maritime industry. Kongsberg Digital is constantly thinking of innovative solutions for the Maritime simulation industry and its instructors, students, organizations, and users.
They have innovative simulation solutions which are available both online and offline. Kongsberg Digital Maritime simulations are part of the K-SIM product line.

One part of the K-SIM product line is the K-SIM Instructor, where instructors and trainers can create simulation exercises for students to execute and perform. The K-SIM Instructor application is a Windows desktop application for instructors to use locally. The installation and use of the K-SIM Instructor application requires other K-SIM tools, models, and area charts. These requirements add complexity for the instructors. Azure Virtual Desktop helps Kongsberg Digital Maritime to take ownership of the installation and configuration part of the K-SIM Instructor application and the end users don't need to install anything on their machines.

The K-SIM Instructor application also has strict compatibility requirements with the simulator engine for the students to execute and perform the simulations in the K-SIM Connect cloud environment. Since Kongsberg Digital Maritime is in control over the complete installation and configuration of both the K-SIM Instructor application and the simulator engine versions, the compatibility requirements are centrally managed and guaranteed compatible.

Azure Virtual Desktop as a cloud accelerator

Azure Virtual Desktop is a quick and straightforward way to start your cloud journey by providing a solution for your traditional desktop applications. It is low maintenance, deployable using Infrastructure as Code, and has no or low requirements of client systems to start working. While using the Azure Virtual Desktop service, you will improve your organization's knowledge and experience what the Microsoft Azure cloud can offer your organization.



```
resource hp 'Microsoft.DesktopVirtualization/hostpools@2019-12-10-preview' = {
  name: hostpoolName
  location: avdbackplanelocation
  properties: {
    friendlyName: hostpoolFriendlyName
    hostPoolType : hostPoolType
    loadBalancerType : loadBalancerType
    maxSessionLimit: 12
    preferredAppGroupType: 'Desktop'
    customRdpProperty: 'drivestoredirect:s:*;audiomode:i:0;videoplaybackmode:i:
    registrationInfo: {
      expirationTime: add30Days
      token: null
      registrationTokenOperation: 'Update'
    }
  }
}
```

RDP Properties: targetisaadjoined:i:1;drivestoredirect:s:*;audiomode:i:0;videoplaybackmode:i:1;redirectclipboard:i:0;redirectprinters:i:0;devicestoredirect:s:*;redirectcomports:i:1;redirectsmartcards:i:1;usbdevicestoredirect:s:*;enablecredsspsupport:i:1;use multimon:i:0;screen mode id:i:2;dynamic resolution:i:0

The time-to-market is very fast while adding many other cloud benefits at the same time.

What services do you need for Azure Virtual Desktop

Azure Virtual Desktop is a service that handles the management layer for providing a Virtual Desktop Infrastructure service.
The Azure Virtual Desktop service consists of a few main areas:

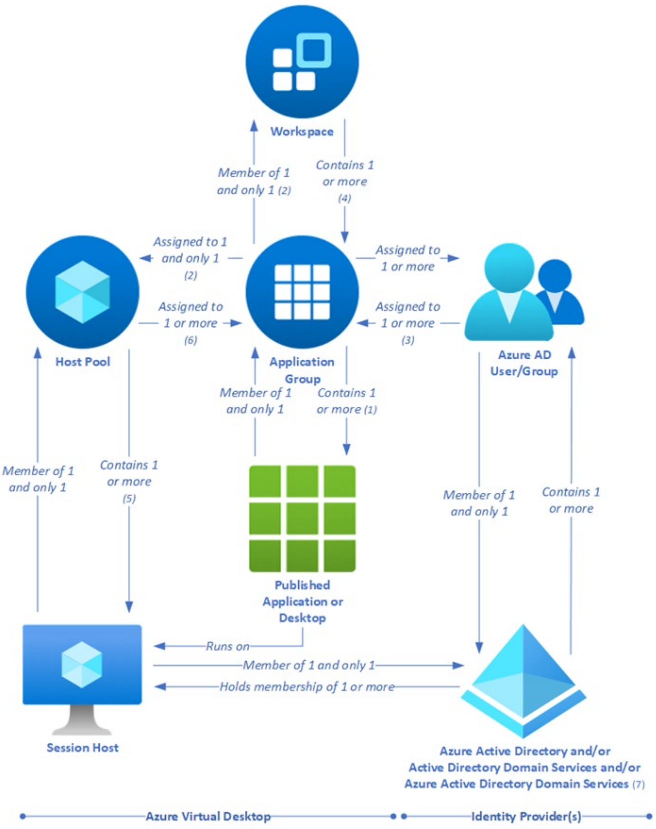


Figure 1. Azure Virtual Desktop Key Logical Components

- Workspaces
  Workspaces are a logical group of one or more application groups.

- Application Groups
  Within Application Groups, the applications and desktops are configured, including user and group assignment. The Application Groups are logical groups of installed software on the sessions hosts within the host pool(s)
- Host Pools
  The Host Pools are one or more pools of session hosts providing the computing power for the application groups.
- Scaling plans
  Scaling plans are used for ramping hosts up or down based on session usage and peak- and off-peak hours.

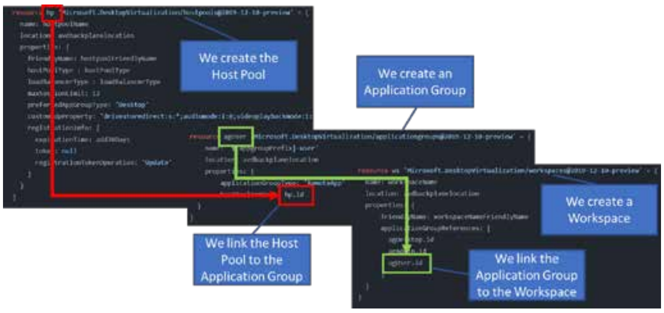
An Azure Virtual Desktop host pool requires either a Virtual Machine service or Virtual Machine Scale Set service to provide the computing power on which the applications are installed and running. These services require a Windows Image, which we build using the Azure Image Builder service. We store this Windows Image within a Shared Image Gallery, as a Virtual Machine Template.

The identity provider for Azure Virtual Desktop can be either Azure AD or a traditional Active Directory Domain Services domain. When using a pooled host pool, the user can log in to any of the hosts that are available. To make sure the user experiences their profile the same on all the hosts, we are using FSLogix profile containers, which are stored on an Azure file share.

FSLogix enhances and enables user profiles in Windows remote computing environments and is able to supply profile containers, office containers, and application masking.

Can I try it myself?

Yes, you can! We have set up a GitHub Repository for you to deploy Azure Virtual Desktop with Bicep.



```
resource hp 'Microsoft.DesktopVirtualization/hostpools@2019-12-10-preview' = {
  name: hostpoolName
  location: avdbackplanelocation
  properties: {
    friendlyName: hostpoolFriendlyName
    hostPoolType : hostPoolType
    loadBalancerType : loadBalancerType
    maxSessionLimit: 12
    preferredAppGroupType: 'Desktop'
    customRdpProperty: 'drivestoredirect:s:*;audiomode:i:0;videoplaybackmode:i:[illegible]
    registrationInfo: {
      expirationTime: [illegible]
      token: null
      registrationTokenOperation: 'Update'
    }
  }
}
```

```
resource agrser 'Microsoft.DesktopVirtualization/applicationgroups@2019-12-10-preview' = {
  name: '[illegible]groupPrefix}-user'
  location: avdbackplanelocation
  properties: {
    applicationGroupType: 'RemoteApp'
    [illegible]: hp.id
  }
}
```

```
resource ws 'Microsoft.DesktopVirtualization/workspaces@2019-12-10-preview' = {
  name: workspaceName
  location: avdbackplanelocation
  properties: {
    friendlyName: workspaceNameFriendlyName
    applicationGroupReferences: [
      agdesktop.id
      [illegible].id
      agrser.id
    ]
  }
}
```

We create the Host Pool
We create an Application Group
We create a Workspace
We link the Host Pool to the Application Group
We link the Application Group to the Workspace

Our Azure Virtual Desktop GitHub repository[1] contains all the bicep files you need to run an Azure Virtual Desktop in the cloud with a default app. </>

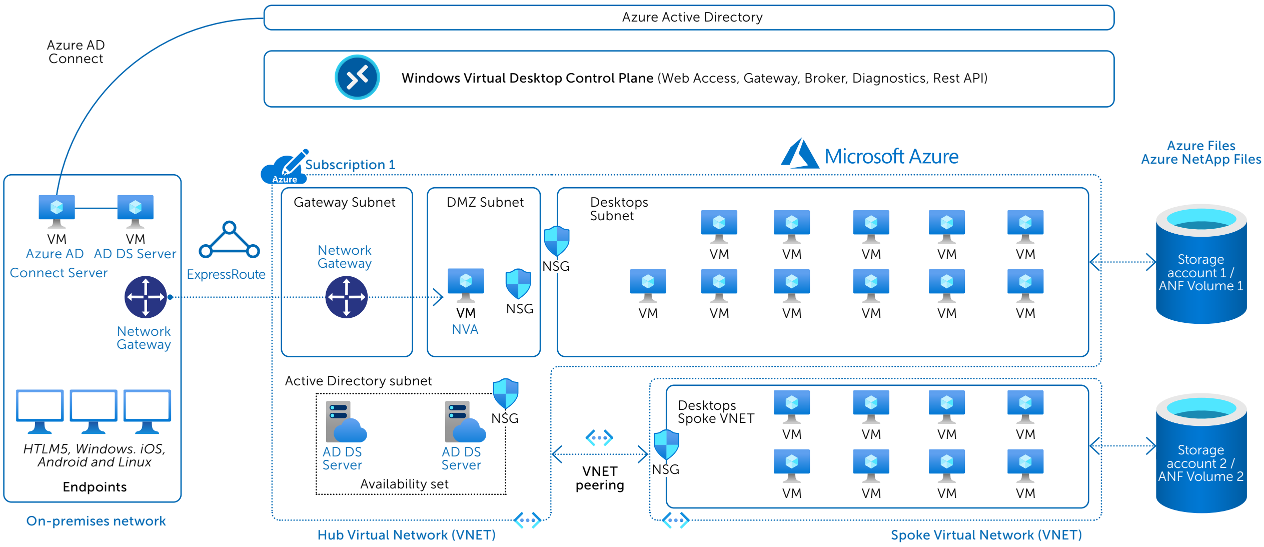


Figure 2. Example Azure Virtual Desktop Architecture

Patrick de Kruijf
Consultant

pdekruijf@xpirit.com

[1] https://github.com/XpiritBV/azure-virtual-desktop

Creating 3D experiences for Azure Digital Twins

A digital twin is a connected, digital representation of a physical environment. To create these living systems, Microsoft created Azure Digital Twins. This product has been around for a while, but it has always lacked built-in visualisation options for end users. This article explains how to create a 3D visualisation on top of your digital twin without needing developer skills and why you would want to.

Author Matthijs van der Veer

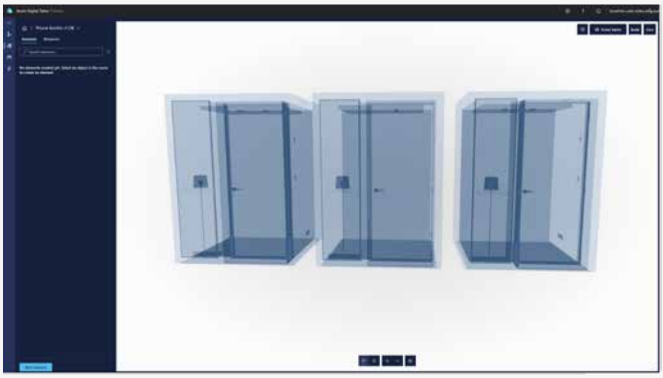
Digital twins and 3D seem to go hand in hand. However, a digital twin could be visualised in many ways. A dashboard or a web page can be just as effective, and you can always give access to your data through an API. Some digital twins don't even need any visualisation. Perhaps their data is used in other systems for further processing.

However, 3D visualisations of digital twins are everywhere to the point that the two have become synonymous for some people. In a way, this makes sense. When you build a digital twin, you dissect the physical environment into smaller logical parts and capture them into a logical model. When creating a visual overview of this twin, 3D can help bring these parts together in a recognisable fashion. Maybe that is why Microsoft decided to strengthen the Azure Digital Twins offering with the 3D Scenes Studio. This new preview allows users to create 3D experiences for their digital twin data. It offers the opportunity for non-developers to provide insights into the data that is gathered in the twin.

Let's use a connected office to demonstrate the features of 3D Scenes Studio. Our office has three phone booths for which occupancy and air quality are measured. To display the status of these phone booths, it is essential to know which booth is occupied and if the air quality allows a productive meeting or phone call. For this purpose, we will show the entire booth as occupied or unoccupied and show an alert when the air quality is poor.

Making a Scene

Everything starts with the Scene, which is a 3D model which contains all the objects. If you already have 3D models that you want to use, you will need to combine them into a *Graphics Language Transmission Format* (or .glTF) file. Various products can do this. For the example below, Blender was used.

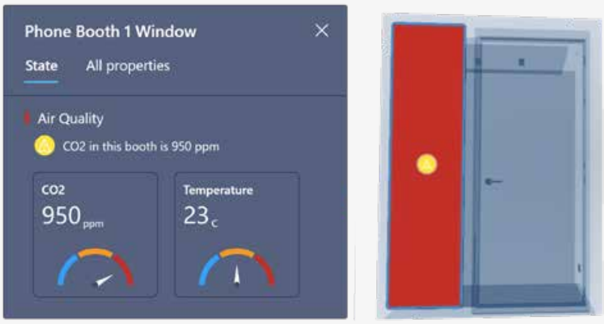


Picture 1. A 3D model of a phone booth, loaded in Azure Digital Twins

The first thing to do when loading the Scene is to assign Elements to the 3D model. Each Element can be linked to a twin in the Azure Digital Twins graph, giving it access to the twin's data. A twin can be assigned to a mesh, which is a collection of polygons, defining a subset of your 3D model.

In the case of our phone booths, the 3D model is split into two meshes: one for the booth itself and another for the glass pane next to the door. This allows for assigning different twins to a booth instead of the booth being one massive 3d model. Assigning an element to a mesh is a manual process. Click on the item that needs to be assigned, find the twin to associate and give it a name. This is fine for a couple of objects but is very tedious to do for a few hundred booths, as it currently cannot be automated.

After defining all the Elements, it is time to create Behaviours. A Behaviour can change the colour of an Element or attach Widgets and Alerts to it. In this example, the booth will turn red when occupied. The air quality can be monitored in a widget, but if the CO2 sensor reports over 900 parts per million, an alert is also shown.



Phone Booth 1 Window

State | All properties

Air Quality

CO2 in this booth is 950 ppm

CO2 950 ppm

Temperature 23 c

Picture 2. A booth with two widgets and one alert.

After the behaviours have been created, they can be assigned to the various elements. The whole process, from creating the Scene, to assigning behaviours, can be done in a matter of minutes. This is the strong suit of 3D Scenes Studio. When viewing the scene, you can freely navigate the 3D model where data is updated constantly.

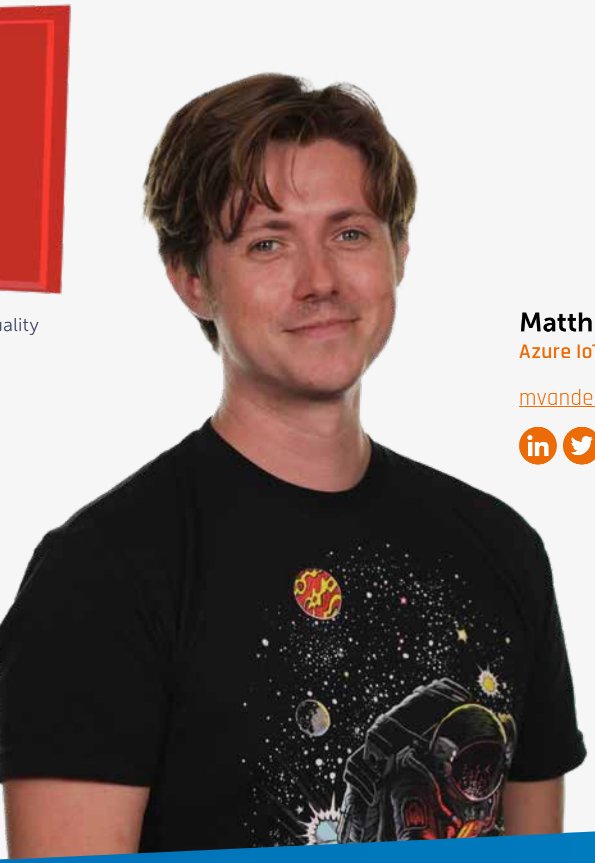
Picture 3. Two booths are occupied. One is free but with bad air quality

Why (not) use 3D Scenes Builder

When creating a scene and assigning Elements and Behaviours, a non-technical user can provide a rich 3D experience for other users. The result is a 3D environment that is updated in real-time. But when digging a little deeper into how the data is updated, the whole application is built on polling Azure Digital Twins. Every ten seconds, the data of the twins being displayed is retrieved. This will lead to an increase in API calls, which is also how Azure Digital Twins is being billed; you pay for what you use. When designing a (near) real-time application on Azure Digital Twins, a better practice would be to use the property change events and feed them to your app through SignalR. If you're already using the change events, the price increase would be minimal. But in the current preview state, polling for all the twins every ten seconds for every browser window looking at the scene can lead to increased API calls.

Another challenge with 3D Scenes Builder is that one 3D model equals one Scene. If you create a Scene for your factory and decide to change the configuration, you need to start over again with a new 3D model. The model cannot be edited in 3D Scenes Builder; therefore, you will always need someone with 3D experience to maintain the 3D models.

It is hard to point to any particular feature as being the best part of 3D Scenes Builder. For us, the best part is that finally there is a way to share the twin data with users that need it. For a long time, the default visualisation was limited to Azure Digital Twins Explorer. Having the 3D Scenes Builder adds to the maturity of Azure Digital Twins. We expect that we will see a lot of splendid examples of projects and prototypes making use of 3D. Because in the end, while a digital twin is not synonymous with 3D, it undoubtedly contributes to an impressive experience. </>



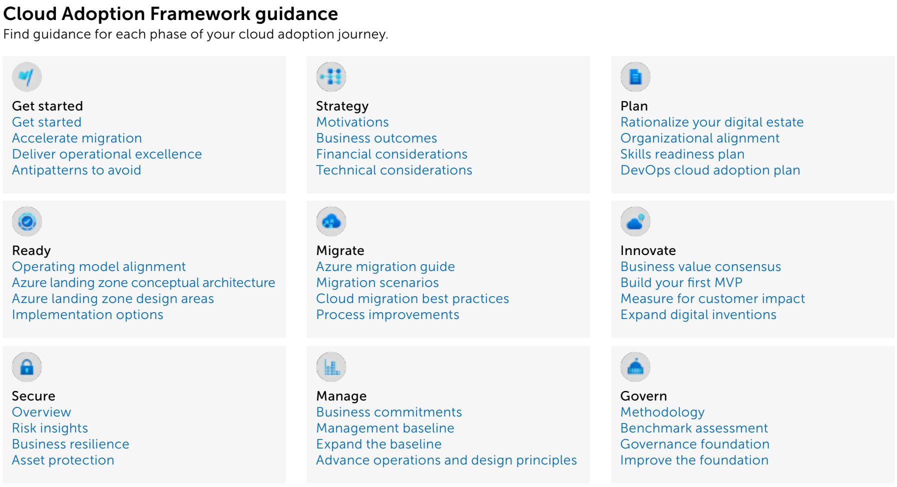
Matthijs van der Veer
Azure IoT Specialist
mvanderveer@xpirit.com

How to move to the cloud according to the Microsoft Cloud Adoption Framework

The first question is whether moving to the cloud is the right thing for you. After all, each organization is unique. What are your business objectives? What's the required investment versus the return? Does a cloud-based business model fit in with your organization, or do you need to change the model or the organization? What´s the technology and expertise that´s required? Is it something you can do yourself, or do you need expert assistance?

Authosr Patrick de Kruijf, Erwin Staal and Marc Bruins

Xpirit has supported dozens of organizations, ranging in size from global corporations to SMEs, in successfully transitioning to the cloud using the proven Microsoft platform. A valuable part of this platform is the Microsoft Cloud Adoption framework, which assists you in investigating your motivation and clarifying your strategy. The following image shows the key areas of your investigation.



Cloud Adoption Framework guidance
Find guidance for each phase of your cloud adoption journey.

Get started
Get started
Accelerate migration
Deliver operational excellence
Antipatterns to avoid

Strategy
Motivations
Business outcomes
Financial considerations
Technical considerations

Plan
Rationalize your digital estate
Organizational alignment
Skills readiness plan
DevOps cloud adoption plan

Ready
Operating model alignment
Azure landing zone conceptual architecture
Azure landing zone design areas
Implementation options

Migrate
Azure migration guide
Migration scenarios
Cloud migration best practices
Process improvements

Innovate
Business value consensus
Build your first MVP
Measure for customer impact
Expand digital inventions

Secure
Overview
Risk insights
Business resilience
Asset protection

Manage
Business commitments
Management baseline
Expand the baseline
Advance operations and design principles

Govern
Methodology
Benchmark assessment
Governance foundation
Improve the foundation

If you are considering a migration to Azure, using the Microsoft Cloud Adaption Framework (CAF) is one of the paved roads you could follow. The CAF helps you to define and implement a cloud strategy. It contains documentation, tools, and best practices. It also describes a cloud adaption lifecycle that has eight phases. One of them is the 'Ready' phase, and this is the phase in which the cloud environment is prepared for the planned implementation. One way to do this is to implement a Landing Zone.

Getting started and strategy

The Microsoft Cloud Adoption framework helps you focus on your business objectives. Why would you move your business to the cloud? Is it because of cost savings, or do you want to enable a new business model for your products and services? For instance, do you want to upscale to match market demand?

If moving to the cloud is the right decision to achieve your business objectives, the next step involves investigating the steps you need to undertake. This means taking stock of your digital estate, including a list of all your technical servers. You must decide whether you need to move, rearchitect, rehost, rebuild, refactor, or maybe retire them, i.e., the five Rs of rationalization.

From the organizational perspective, you must ensure that your skill set matches what you want to achieve or that you can grow to match the required skillset. Most likely, the organization of your business also needs to change to embrace the cloud and move to the cloud truly. These changes require a broad commitment to ensure your success. Moreover, to truly embrace the cloud, companies need to adopt a DevOps way of working, which leads to rethinking your organization structure. At Xpirit, we help companies get started migrating to and working with the cloud. At the same time, we guide them to implement a DevOps approach from a technical perspective as well as from an organizational perspective.

Creating the right foundation

When ready to move to the cloud, a key success factor is a solid and secure foundation for your applications. As a company, you will still need to consider cross-cutting concerns such as security, networking, compliance, costs, etc. However, your teams may be responsible for this and should be enabled to do so effectively and efficiently. Here the concept of a landing zone comes in, which is the concrete implementation of the foundation that will help your teams to be effective.

What is an Azure landing zone?

"An Azure landing zone is the output of a multi-subscription Azure environment that accounts for scale, security governance, networking, and identity. An Azure landing zone enables application migration, modernization, and innovation at an enterprise scale in Azure. This approach considers all platform resources required to support the customer's application portfolio and doesn't differentiate between infrastructure or platform as a service."

Landing zones can be easily customized to fit your organization's requirements seamlessly and thus be the enabler for your migration.

Using landing zones when migrating to the cloud

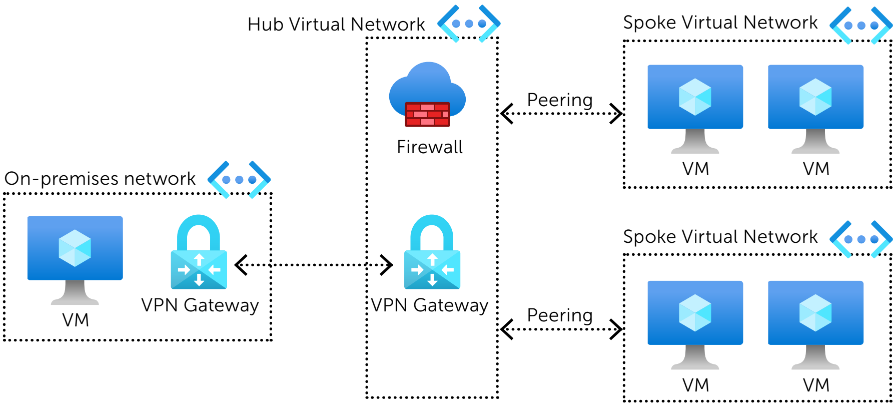
Imagine a company called FreeBirds. They have run their software in a data center in the Netherlands for over ten years. Their business is growing rapidly in terms of the number of customers, but it's also been a long time since all their customers came only from the Netherlands. FreeBirds decided to move their applications to the Azure cloud. One of the reasons for this decision is to reduce costs by fully leveraging the elasticity of the cloud. Another reason is to be able to run their applications across the globe and thereby increasing the speed of the application as well as its resiliency and scalability. Building a Landing Zone on Azure using a Hub and Spoke architecture is ideal for this scenario.

Ready-made foundation

When explaining the concepts of the Azure Landing Zone, it is helpful to do this in the analogy of building a home. If you want to build a new home, you could do everything yourself: dig the foundations, lay the bricks, and do all the plumbing. The same goes for the Azure cloud. You could start to build the infrastructure manually yourself. As with the house, you might find this very time-consuming, and there would be the risk of making many mistakes. It would be much easier to use ready-made foundations and a blueprint that shows you exactly how to do things and implements best practices. You could still customize the structure to personal needs, but the building would be architecturally sound, safe, and faster to build. An Azure Landing Zone is precisely like that. It will cover network architecture, security, identity, and governance, allowing DevOps teams to start building right away on a perfectly laid-out foundation.

Architecture

Landing Zones are often implemented using a Hub and Spoke architecture. In this type of architecture, you have a central hub. The network in the corner acts as a central point of connectivity to on-premises resources for many spoke virtual networks, as shown in the image below. You often find other resources in the hub shared among spokes, for example, an Azure Firewall or a Log Analytics workspace for central log management.



Hub and spoke architecture

The middle part of the image above shows the hub virtual network. It contains the resources needed to provide connectivity to the on-premises network on the left. That connection is often established using a VPN Gateway or an Azure Express Route. The connection in the hub network can then be used by multiple spokes, as shown on the right. Each spoke virtual network holds one workload, and the spokes allow you to isolate your workloads from those of other teams. For example, you can use spokes to run Virtual Machines (VM), Azure Web Apps, or databases.

In the scenario of FreeBirds, this would be a perfect fit. A central team, often called a platform team or something similar, can build and manage this Hub with all the central services. Each DevOps team creates and delivers applications end-to-end and becomes one or more spokes. This central team

can ensure a safe adaption of the cloud, for example, by forcing all traffic to the internet to go through a firewall. They can also use Azure Policies to set guardrails for the DevOps teams, ensuring that they cannot deploy, for example, a database in an insecure way or forcing a daily backup. It is essential to ensure that these significant teams do not become bottlenecks for other DevOps teams. This means that the platform team must fully deliver their services in a self-service and automated manner as much as possible.

Valuable scenario

A Landing Zone is not only useful in a scenario like that of FreeBirds, i.e., a company moving away from its on-premises data center. It will also fit in hybrid scenarios and fully cloud-native environments that do not contain any on-premises connectivity. All the benefits remain, such as easy onboarding on the cloud, security, identity, and governance. The Hub is also often used to provide central services to all teams that would otherwise be too expensive or complex. An excellent example of such an expensive and complicated offering is API Management, a service to expose APIs to the outside world. When run in a production-ready way, it easily costs over 3000 euro's a month. This is too much for each DevOps team to run independently, and thus it is often managed centrally. A service like Kubernetes, AKS on Azure, is another service that is often managed centrally. Besides being expensive, it's also a complex service to operate. Bringing that to a central team allows the other DevOps teams to focus on their business value instead of using their infrastructure.

Xpirit Managed Landing Zone

The Landing Zones of the Microsoft Cloud Adoption Framework provide a solid foundation with a complete set of generic functionality for moving your services, applications, and data into the cloud within weeks. And if you choose an Xpirit Managed Landing Zone, we take care of ownership and maintenance tasks, allowing you to focus on your core business. Moreover, we will ensure your system is always up to date by implementing new features as soon as they become available. What sets us apart is our intimate service provision with direct contact with the right expert, combined with our extensive experience and expertise in the domain of the Microsoft platform.

At Xpirit, we offer out-of-the-box landing zones that derive their value from our seven years of experience in building them. We have identified multiple flavors of these landing zones based on the domain that customers work in. In addition, we have automated and optimized them, allowing them to be deployed and up and running within two weeks.

For example, our Business Apps/ Independent Software Vendor (ISV) landing zone helps ISVs create the scale required to meet market demand. This landing zone allows them to make a fast and reliable transition to a world-wide scale. Typically these companies have one or more apps that need to be enabled worldwide or to harness the scalability and reliability principles of the Azure Cloud.

Of course, we can review your landing zone or create a custom zone that suits your organization.

Hub Spoke Model
The Xpirit Landing Zones use the Hub and Spoke architecture. Individual applications run in spokes, each with a domain-specific workload, while the hub provides generic services such as connectivity to the cloud and proven security features. Naturally, the Xpirit model pays extra attention to security with features that comply with ISO 27001, Azure Security Benchmarks, and CIS.

Comprehensive services with an ultra-short migration timeframe
We use a swift combination of the re-host and re-architecture cloud rationalizations – part of the Cloud Adoption Framework – to move logical components from a data center to the Hub and Spoke model. A primary application can be up and running in a spoke within three weeks, which means that a fast startup is possible using the generic features of the high-performance hub architecture. And if you have any specific needs requiring more specialized functionality or support for your applications, we can still support you. After the re-host and re-architecture phase is completed, we can look at the other R's of cloud rationalization to improve the application. In addition to managing the hub, we can also order a spoke application and help you improve your application.

Intimate service provision through a single point of contact
Traditional service provision is structured in terms of first and second-degree support lines. Usually, you won't get to speak to an engineer directly when you create your ticket but will receive automated responses. This means you often must wait to talk to a service team member who may not be completely aware of your specific situation. Our team wanted to innovate this approach and create a much more intimate form of service provision by making our Landing Zone engineers directly available to support our customers.

The Xpirit Managed Landing Zones team consists of knowledgeable Landing Zone experts with years of experience in enabling and supporting cloud migrations and consultancy, coding for specialized architectures, and advanced interfaces. You communicate directly with the right engineer, who can always rely on the know-how of our entire Xpirit team, should this be required. Moreover, our service is available 24/7: we have the most qualified expert ready to help you during business hours and direct operation support on standby at night. As a result of the single point of contact and the short communication lines, you reap the benefit of faster troubleshooting.

Guaranteed response times
Following a short but thorough intake process, you receive a transparent service-level agreement, including guaranteed response and recovery times for incidents. We align our service levels to the severity of a possible incident, varying from incidents that would mean the discontinuation of your business to the lesser impact that slows down service to less urgent issues. The organization of our service team allows us to meet the agreed response times on a 24/7 basis.

To continuously improve our support system, we use a net promotor system to regularly assess our clients' experiences and monthly meetings with our clients and their service managers.

Sustainable problem solving
Thorough engineering will improve your experience with the platform. A defining feature of our service is our policy of preventing recurring incidents and avoiding temporary workarounds and patches. We perform a post-mortem for every incident by thoroughly investigating the root cause of a problem and finding a sustainable solution. This problem-solving approach and properly fixing anomalies instead of quick patching will save time.

Our extensive set of automated tools with various validations and alerts often allows us to predict and be ahead of any problems before they occur. For instance, we use Azure Monitor to monitor the used components within your Azure tenant continuously, allowing us to take preventative measures before any workload or congestion problems occur. Moreover, thanks to Xpirit's close collaboration with Microsoft, we have direct and short contact lines with their team of cloud engineers. </>

Marc Bruins
Architecture / Azure / mobile development

mbruins@xpirit.com

Patrick de Kruijf
Consultant

pdekruijf@xpirit.com

Erwin Staal
Azure Architect

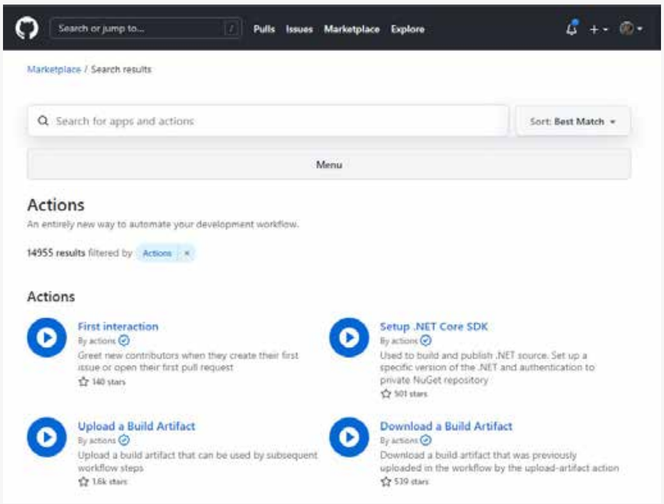
estaal@xpirit.com

GitHub Actions has security issues

I am fascinated with the security aspects of using GitHub Actions for my own workloads since I have started using them. My first conference session on this topic was at NDC London in January, 2021[1] and I have been advocating on these learnings ever since. That is why I also decided to run my usual security checks on the entire marketplace, starting with forking the actions so I can enable Dependabot on the forked repositories.

Author Rob Bos

The Marketplace shows us almost 15 thousand actions that are available for us to use! That means there is lots of community engagement for creating these actions for us, but also lots of potential for malicious actors to create actions that can be used to compromise our systems. Do be aware that in this post I'll only be taking actions into account that have been published to the marketplace. Since any public repo with an `action.yml` file in the root directory can be used inside of a workflow, there are many more actions that are available to us that are not part of this research.



Actions

An entirely new way to automate your development workflow.

14955 results filtered by Actions ×

Actions

First interaction
By actions
Greet new contributors when they create their first issue or open their first pull request
☆ 140 stars

Setup .NET Core SDK
By actions
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository
☆ 501 stars

Upload a Build Artifact
By actions
Upload a build artifact that can be used by subsequent workflow steps
☆ 1.6k stars

Download a Build Artifact
By actions
Download a build artifact that was previously uploaded in the workflow by the upload-artifact action
☆ 539 stars

Analysis of the actions from the GitHub Actions Marketplace

I created a new repo[2] to run these checks using GitHub Actions by scheduling a workflow that runs every hour and checks the dataset for new actions that have not been forked to my validation organization yet.

Some caveats up front:

- I could only load the information for 10.5 thousand actions. All the others have issues that makes it that I cannot find them anywhere. These are not included in the dataset for this analysis.
- Some have been archived by their maintainer, but still show up in the Marketplace. These are of course older and have more security issues in them. The actions are included in this analysis. I'm planning to remove these when the Marketplace doesn't show them anymore.
- There are some actions where I could not parse the definition file (if used), often because of duplicate keys in their definition file. I've reached out to some of the maintainers to get those fixed, but also want to improve my method of loading these kinds of files. Currently the library I use for this does not support duplicate keys and throws unrecoverable errors when it finds them.

I've reported this information back to GitHub and they are planning to improve the freshness of the data in the Marketplace. Still, this is a good two thirds of the actions that are available in the marketplace, so this is a representable dataset to look at.

[1] https://devopsjournal.io/blog/2021/01/28/GitHub-Actions-NDC-London
[2] https://github.com/rajbos/actions-marketplace-checks

Examples of actions that show up in the marketplace but will give an error when you want to load the detail information for them include:

> c-documentation-generator[3]
> cross-commit+[4]

Additionally, all this analysis is done on the default branch for the repository. I myself have one action, for example, that uses a Dockerfile in the main branch, but I am working on converting it to Node in another branch. This number should be small enough to have no significant impact on the overall analysis.

Security alerts for dependencies of the Actions

I have forked over the action repos to my own organization and enabled Dependabot on them to get a sense of the vulnerable dependencies they have in use. Some caveats to this analysis are:

> Not every dependency will end up in the action itself, so a high alert from Dependabot will point to a 'possibly' vulnerable action. Since this is not something you can track automatically and see if this would be the case, we cannot be sure that the action itself is vulnerable.
> This only works for Node based actions, which is 4.7k, so almost 50% of the analysed actions. Dependabot does not support Docker at the moment.
> I'm only loading the vulnerable alerts back from Dependabot that have a severity of `High` or `Critical`.

I'm planning to add something like a Trivy container security scan to the setup so that we get some insights from this as well.

Security scan results

Of the 10488 scanned actions, 3130 of them have at least 1 high or critical alert! This is a way higher than I even expected and very scary! And this is only for Composite or Node based actions! If your dependencies already are not up to date and thus have security issues in them, how can we expect your action to be secure? That calculates to 30% of the actions that have one or more high or critical alert on their Dependencies.

To be complete: I have not filtered down the alerts to a specific ecosystem. Since GitHub Actions is one of the ecosystems Dependabot alerts on, there is a chance these alerts come from a dependency on a vulnerable action for example, which would be unfair (since these will not end up in the action I am checking). Since there are only 3 actions in the GitHub Advisories Database[5], I expect this to be of zero significance, but still: it is worth mentioning.

Diving into the security results

I've also logged the repos with more 10 (high + critical) alerts to a separate report file and that file contains more than 600 actions!

The highest number of high alerts in one single action, is 58. Since that repo happens to be Archived, it should not be in the actions marketplace at all, as well as the fact that this should not be used at all. Luckily it is only used by a small number of workflows. I'd rather see that the runner would at least add warnings to the logs for calling actions that are archived.

The highest number of critical alerts in one singe action is 16. This repo is also only used by less than 10 other repos, so it is not a big impact. Since there is no API for finding the dependents that Dependabot finds, I cannot easily find out how many workflows are impacted by this.

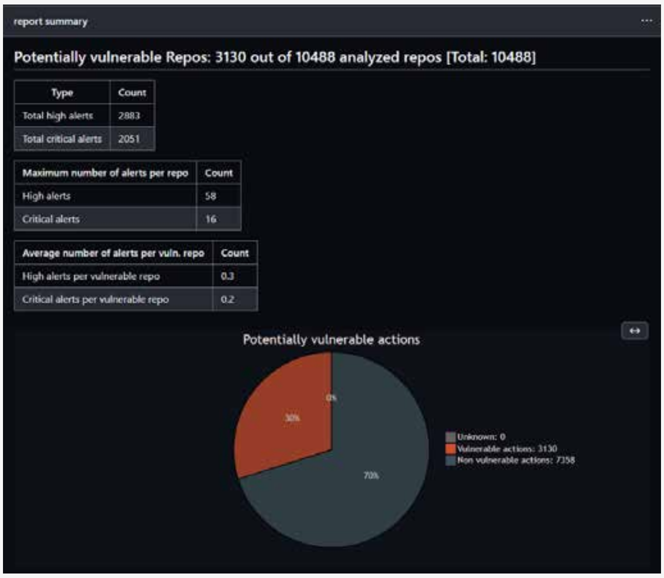
I've checked some of the repos with many of alerts and found one example that has 14 high severity alerts and 2 critical alerts. This action is used by 34 different public repos (so in private repo usage could even be more!). One of these dependents is a repo with 425 stars and another has 6015 stars. That last one is producing a serverless CMS that will be delivered as 48 different packages into the NPM ecosystem. One of those packages sees more than a 1000 downloads a week! This is a lot of impact for a single action that could be prevented by enabling Dependabot. Of course, more analysis is needed for this case to see if the alerts are actually relevant for the action. This depends on what the action does and how it uses the dependencies.

Source http://gunshowcomic.com/648

[3] https://github.com/marketplace?type=actions&query=c-documentation-generator+
[4] https://github.com/marketplace?type=actions&query=cross-commit+
[5] https://github.com/advisories?query=type%3Areviewed+ecosystem%3Aactions

Overview

In short, this is a top level overview of the security results:



report summary

Potentially vulnerable Repos: 3130 out of 10488 analyzed repos [Total: 10488]

Type	Count
Total high alerts	2883
Total critical alerts	2051

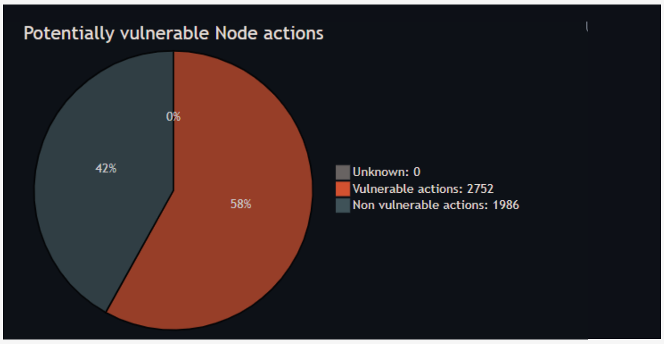
Maximum number of alerts per repo	Count
High alerts	58
Critical alerts	16

Average number of alerts per vuln. repo	Count
High alerts per vulnerable repo	0.3
Critical alerts per vulnerable repo	0.2

So for all action repos I could scan, 30% have at least 1 vulnerability alert with a severity of high or critical.

Node based actions

Filtering this down to only the Node action types, this becomes a lot scarier:



That is 58% of the Node actions that have at least 1 vulnerability alert with a severity of high or critical! And all demos and docs still indicate you can just use the actions as is and only hint at the security implications of that!

Want to learn how to improve your security stance for using actions? Check out this guide I made: GitHub Actions Maturity Levels[6].

Conclusion

There is a lot of improvement that can be made to actions ecosystem. I would like to see GitHub take a more active role in this by, for example:

- Enforce certain best practices before you can publish an action to the marketplace.
- Clean up the marketplace when an action's repo gets archived (work for this is underway).
- Add a security score to the marketplace, so that users can see how secure an action is, run at least these type of scans on the action repo and report it back to the end user.
- Add a check that validates you also pushed a new release of the action to prevent maintainers to add Dependabot and keep their (vulnerable) dependencies up to date, but not actually release a new version of the action.
- Add API's to not only the marketplace, but also Dependabot. This information should be publicly available, but currently I had to scrape this of the webpages.

Of course, as maintainers of actions we are also in this together! It's our responsibility to make sure our actions are secure and that we keep them up to date. I hope this post will help you understand why to do that! </>

Rob Bos
Consultant & Trainer

rbos@xpirit.com

Moving the Business Needle

Nowadays, every business is a technology business, no matter what product or service it provides. In the current day and age, no company can make, deliver, or market its product efficiently without technology. Whether it is banks, insurance companies, logistics companies, or retailers, technology is critical to their success. Many companies embrace this fact. They understand when they adopt new technology and implement it successfully, they gain a stronger foothold on the market and stay relevant. Companies that wait for a second or third wave stay at the back of the pack and will have a very hard time becoming a leader in their market or even staying afloat in markets with strong competition.

Author Dennis Thie

To be a technology company, you need to act and behave like a technology company. At Xpirit, we look at your business as a collection of capabilities that are required to allow you to run, create and deliver your products and services. We distinguish digital capabilities and leadership capabilities, which are both required to digitally succeed as a business. A digital capability we regard as a combination of people, processes, and technology to deliver value to internal and external customers. The leadership capability is about merging the skills and perspectives of your business and IT leaders with the goal to help them drive change together. For that, they need to have a digital vision, engage the organization at scale, and govern the change.

We call this combined set of capabilities and behaviors a company should have an Engineering Culture. Everything we do as Xpirit adds to this vision. This Engineering Culture can be seen from many different perspectives that we categorize into a number of distinct pillars that together will help you become successful and can be used to drive change.

One of the pillars of this Engineering Culture is *Moving the business needle*. For us, moving the business needle is all about achieving greater success for your company by helping you get the most value out of technology. Our way of working should have enough of an effect so that people notice a change that adds value to the business. Making a noticeable impact. Moving your business needle to become a truly digital enterprise.

> "Digital is not a thing, but is simply a word that describes our world today."

Start with *why?* (And *for whom?*)

A company can only exist by the grace of its relevance. To stay relevant, you will need to keep reinventing yourself. This applies to people as well as companies. We probable all know stories of companies that have been felled, and the stories of those that have proven able to reinvent themselves.

It's clear that, without relevance, it takes only time to become obsolete and for companies to eventually go out of business.

To be able to stay relevant, you must start by asking yourself the question: what is our *why?* Why are we in business? And, more important, *for whom?* Who are our customers? You'll need to find out who and what decides whether you're still relevant. Because let's face it, it's not you who has the say in that. You'll find out that your customers aren't part of a single group. In fact, your customers will be both internal parts of your organization, customers you directly serve and customers you indirectly serve, in the form of your customers' customers. You're not in business for a single of those groups and have the others merely follow. To be perceived as relevant, it is all about delivering continuous value to those that you serve, let's call them stakeholders. And to impact them, things don't stop at the gates of departments or even your company. You'll need to think in delivering value end-to-end to truly impact your customers.

Organize around value streams continuously delivering value

That is where value streams come into play. A value stream is focused on how value is delivered to stakeholders and is not about business processes or how things are done. Value streams use an outside-in view, from the perspective of the customer rather than an internal value chain or process perspective. Using this outside-in view, you're challenged to think as the stakeholder affected. A value stream can be cross mapped to enable business capabilities that describe what an organization must do to deliver value to that stakeholder.

"You build it, you run it!"

Within a value stream, multiple disciplines work together creating a continuous flow of value. Instead of working in silos, value streams contain cross-functional teams that can work autonomously. They have adopted the *'You build it, you run it'* approach, taking responsibility and control over development and operations. This shortens the feedback loop, eliminates barriers, and lessens delays, ultimately allowing faster learning, higher quality, increased productivity, and shorter time-to-market.

Know where you differentiate, versus what is (or should be) a commodity

After knowing your why and for whom you are in business, it is also important to understand where you differentiate. Because to bring the most value, you need to know where to focus your efforts. Things where you don't or simply could not differentiate should be commoditized, meaning you adhere to industry standards in processes and technologies. You must make choices and focus your precious efforts and resources, as it's not worth – nor feasible - spending those towards things that deliver a relatively low value. Having this focus is crucial for success, especially in times like these where talent is scarce, technologies change with the speed of light and you are in a constant race with your competition, be it known or unknown.

Don't make the mistake of thinking your organization is (too) special, be honest to yourself here. Sure, you will have processes that are in some shape or form unique to your business. But, is that because they are truly differentiating you, or are they because they've grown that way historically and you are holding on to them *believing* they differentiate you? Remember, most in fact are – or should be – commodity. Don't try and reinvent what everybody else is doing, but instead adopt standards whether it be processes, technologies, or software. This leaves a way to focus your precious money, energy, and time on what really differentiates you and can therefore deliver the highest value and make the biggest impact.

Embrace a culture of change... or brace for impact!

Understand that the heart of becoming a digital enterprise is not just technology, but human capacity. And with becoming a digital enterprise, therefore, comes cultural change. Instead of going about change in a project-by-project manner, create a culture of change. Continuous investments toward creating a culture of change will allow your organization to embrace it, make change happen, and sustain change.

When changing, it will be natural for there to be criticism and resistance. Communication is key here, communicate clearly, early, and often. Share your vision and the reasons for change. Listen to feedback and create a feedback loop. Bring people on board and develop a team mindset with people representing the whole value chain. This will form a powerful coalition and be the guiding team. These aspects serve as strong governance when embracing a culture of change.

"It does not only help to have a vision and a clear plan with objectives, it is imperative for success."

Develop a vision and a plan, but mind the concrete

At this point, you will have realized that becoming a successful digital enterprise does not begin with technology, nor that it stops at technology; rather, it's all about having the right strategy and mindset. Yes, technology is an essential tool in achieving your desired goals, but it's only a tool.

Where do you ultimately want to be, what's the dot on your horizon? The answer to that question is your definition of success. And remember, success is relative. To achieve success, set realistic goals on the journey towards that definition that fit your organizations' capabilities. Nothing is more demotivating than working endlessly towards unrealistic goals. That doesn't mean you should neglect the bigger picture and your ultimate goal. Or, as we like to call it, your BHAG: Big Hairy Audacious Goal. But define steps feasible steps towards that goal. Writing down how you plan to move the business needle helps with focus and commitment. It becomes your roadmap, and you will be able to tangibly measure your progress. And don't set your goal in concrete, this allows you to sharpen your goal along the way.

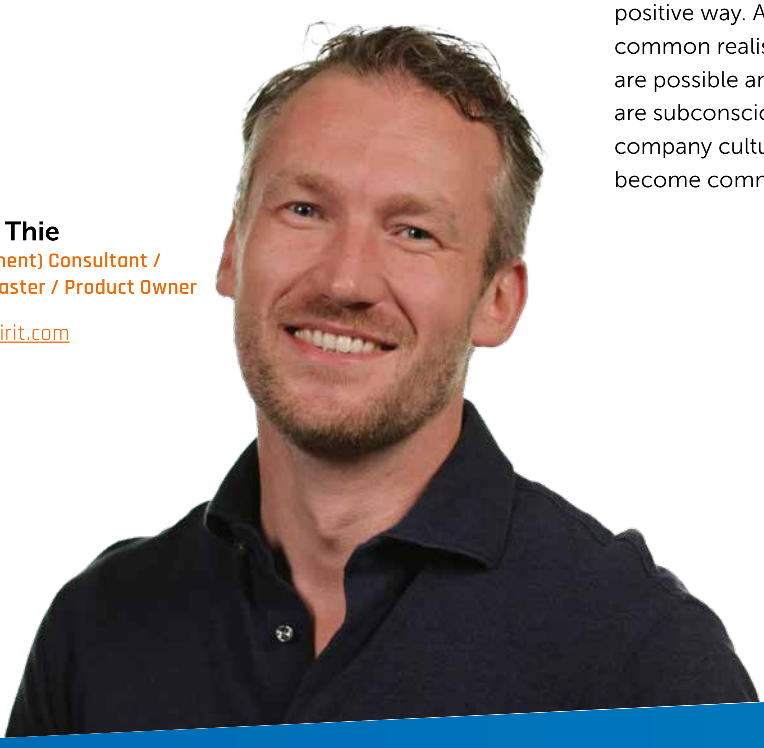
Collectively look at the same gauges and needles

To make a big impact towards your goals, you need to have everyone involved in the organization collectively look at the same gauges and move the same needles. This is something you will only achieve based on a mutual understanding of what is important (your gauges) and how it is measured (your needles). Having your gauges and needles in place allows you to measure the impact of changes and track the results over time. It enables you to employ statistical techniques to discover what aspect of your efforts is having the greatest impact. With this, you can take learnings and steer your plan to focus efforts on what delivers the most value. It creates a feedback loop that allows continuous improvement and that increases effectiveness.

Start small, but be strategic and intentional

You might have heard the phrase "It's not about the destination, but the journey". When you embark on becoming a digital enterprise, you'll for sure not know the exact end goal and how to get there, be it simply because you can't predict everything. If you think you do, look at some recent disruptive events, like Covid, and think again. Hell, it might even be all journey and there might not be a 'final' destination, and that's fine. So, whenever you start the journey, not knowing when or how you'll reach your destination, make the journey itself worthwhile. Start small, and work in steps towards overseeable goals. Create quick wins and be sure to celebrate and share achievements. Experiment and allow yourself failures, or as we like to call them: opportunities to learn.

"If it hurts, do it more often."



Dennis Thie
(Management) Consultant /
Quartermaster / Product Owner

dthie@xpirit.com

Leadership: Put your money where your mouth is

As a leader, show your commitment to achieving results, by being part of it, being seen, and getting your hands dirty. Don't just talk the talk, walk the walk. Put trust in the people around you and give them the power and mandate to make choices they think are best in line with their shared goals and ambition. Empower people to take ownership and reward those who go above and beyond. Nurture a culture that allows people to fail often and fail fast, without shame or punishment, in order to learn and improve.

Share a clear vision and communicate early, clear, and often. Be an approachable leader for your team and always keep your eyes and ears open, absorbing feedback from the organization and eliminating barriers that impede transformation. Address concerns and show that you are there to make difficult choices in the process when plans require adjustment. Don't stick to plans blindly and don't fall for the sunk cost fallacy, a human tendency to follow through on an endeavor just because of your investment in time, effort and/or money into it, whether or not it will outweigh the future benefits.

Build on the change and make it stick

Making all that change stick is not something that happens overnight, it takes thoughtful planning and a true shift in mentality. To make change stick, is to create a culture of change instead of handling change in a project-to-project manner. Only a culture of change will allow your organization to make change happen and truly sustain it. If you mind the beforementioned organizational and leadership behaviors, you are sure to be in the right direction.

"Remember, moving the needle can be infectious."

When you have the right foundation set and you are progressing well in your journey of transformation, moving the needle in a business environment can be infectious, in a positive way. As a team and in numbers, working towards a common realistic goal, greater and larger accomplishments are possible and skills like leadership and priority management are subconsciously born. Make that mindset a part of company culture and growth and success are likely to become commonplace. </>

Xpirit USA – Expanding our worldwide Authority mission

On July 1st, 2022, we announced that Xebia would expand our Microsoft-focused consulting into the US by starting Xpirit USA. Over the past 8 years, Xpirit has established itself as an authority in the world of Microsoft by leading the industry in innovative ways to help our customers adopt Microsoft's cloud (Azure) while enabling teams to take advantage of their new capabilities to deliver software quickly and often. We build Engineering Cultures and truly help drive change within organizations. We work closely with Microsoft, in terms of selling together as well as working with the product teams that build the tools that our customers adopt.

Authors Marcel de Vries and Esteban Garcia

After a successful expansion into Belgium and Germany, we turned our sights to the US. This is the story of how this all began.

It all starts with a conversation...

Marcel and Esteban knew each other for over ten years through the Microsoft MVP and RD programs. Both had seen each other's company's grow in the DevOps and Azure space. Esteban's company was acquired by Cognizant in 2020 and now, in early 2021, the Microsoft Business Group at Cognizant was looking to expand. They turned their attention to the Netherlands, and specifically, towards Xpirit. Esteban knew about the great reputation that Xpirit had and about the amazing team that fueled the company, so he reached out to Marcel for a conversation about the opportunity.

That phone call was the start of everything. At that moment, Xpirit was already part of the Xebia journey with Waterland and the simple answer to whether Xpirit was for sale was NO. But the conversation did not stop there – Marcel is great at turning things upside down, and instead he asked, "What if you give me a call when you want to build a new company?". And that is exactly what happened in November of the same year.

Xpirit had just recently announced that they had expanded their business into Germany, and that the expansion would be led by Michael Kaufmann as the CEO. Through the MVP and RD program, Esteban also knows Michael very well, and it made him think, "what if we could do the same in the US?".

Things got into motion with a new call, but with a different conversation. What if we would start Xpirit in the USA, just as we just did in Germany? Use our combined learnings and continue our Microsoft Authority mission in the US?

Esteban already had an elaborate plan to build a new company, leveraging his previous entrepreneurial experience to put together an amazing team. Robert Bremmer and Esteban had been talking about it for a long time, and the time was now. Rob would join him as the COO, bringing a wealth of operational excellence to the table. Together, they could build the company at an accelerated pace, much faster than we had done until now in the Netherlands, Belgium, and Germany. These are lofty goals, and for that, you need a strong core team. And of course, we must be able to maintain our culture, follow our Authority mission, and be guided by our Values. Once that is in place, our business grows from there.

We had seen this strategy work before, and we knew it would be a solid plan. So next, we had a chat with Daan, Andrew, Stefan, and Anand to see if we could make this plan work. We jointly agreed that building a new company in the USA would be the best approach for starting a great Microsoft Business. Long story short, together we built a business case, brought that to Waterland and we got an agreement to get started.

In April 2022, Andrew and Marcel flew to Orlando, Florida, to meet the future Xpirit USA team. They spent two days getting to know each other, dreaming up the future of Xpirit USA, creating a concrete and elaborate business plan, and making a plan that would take them to kick off. This resulted in both Rob and Esteban saying YES to the plan and us working together to make it happen.

We decided the start date would be July 1st, with Esteban and Rob leading the way in our US expansion. While preparing to get started, word got out and the team already started recruiting our first hire, an awesome recruiter! We met Elizabeth, who worked at Wintellect, a company with a similar DNA as Xebia and Xpirit. Wintellect had recently been acquired by Atmosera, and their leadership had a talent for making people run away to other companies. Elizabeth was looking for new opportunities and was actively interviewing at companies with big names in the tech world, and after hearing our stories and checking with her network of trusted peers, she decided that a future with us exceeded everything that other companies had to offer and joined us in June. So, before we even got officially stared, we had our first employee!

We know that the Xebia success formula includes a great salesperson in the team. Esteban knew exactly who to call – Natalie Reinford, a salesperson who has focused her career in Azure and knows the Microsoft and GitHub ecosystem better than most people. In fact, Esteban hired her in his prior entrepreneurial journey, and now it was time to get the team together. She was working at Wintellect and she jumped at the opportunity to join the Xpirit USA team and be part of this epic journey. Now everything was in place, and it was time to look for consultants so we could start delivering value to our customers.

Once we made the big announcement on July 1st, word spread quickly through our networks, with our LinkedIn posts receiving over one thousand responses from our connections. This resulted in a referral from a Microsoft Regional Director well-known by Esteban and Marcel. And just like that, we had our first applicant through the Xpirit website! Stuart Celarier applied and took the Xpirit Hiring assessment, which resulted in an offer. Stuart joined Xpirit USA on September 12th this year and became our first of many consultants. We also received applications from people at Microsoft and GitHub, who are in the process of getting hired. One of them, David Sanchez, joined us from Microsoft on September 21st. As we write this article, we have five other consultants in the process of interviewing or taking the Assessment.

The US is a big country and we've taken the show on the road. The team traveled to Kansas City and had five presenters in front of over 1,500 attendees at KCDC (Kansas City Developers Conference). If this is not enough, we also spent a week at the Microsoft campus in Redmond, speaking with leadership at Microsoft and GitHub. We made great connections and plans to partner closely globally. Most recently, we were asked to be part of GitHub Universe, deliver workshops there, and be part of the exclusive GitHub Partner Advisory Board. GitHub also keeps looking to us to train Microsoft itself, GitHub Partners, and sellers and develop a new certification Bootcamp to be rolled out worldwide. The Xpirit USA team also traveled to Atlanta to work with Vipul Baijal's Xebia USA team and look for ways to expand our offerings within our existing customer base. In October, Esteban has been asked to join Microsoft Ignite's firs in-person conference in Seattle, as a speaker.

There is a lot of excitement and buzz happening as we get started and we are on track to end this calendar year with at 15 employees, who will work together with our teams in Germany, Belgium and the Netherlands to support our customers globally.

It has been an amazing ride thus far and we are just getting started. We have all the pieces in place to build a real epic company in the US, and we are confident we will be able to make a real statement in the market with our simple but also complex mission: Being an authority in the Microsoft space. </>

Marcel de Vries
Chief Executive Officer
Xpirit Group

mdevries@xpirit.com

Microsoft Regional Director
MVP Microsoft Most Valuable Professional

Esteban Garcia
Chief Executive Officer USA

egarcia@xpirit.com

Microsoft Regional Director
MVP Microsoft Most Valuable Professional

Bring Observability into practice with Azure Managed Grafana

At Xpirit we live by the motto "You Build it, You Run It!". As true DevOps minded people our interest does not stop when a new release is deployed. This is only the first step towards the ultimate goal: deliver a solution with the best end-user experience possible. Running a solution in a reliable, secure and resilient way requires knowing what is going on *after* go-live. We want to measure application performance in the broadest sense of the word.

Authors Rik Groenewoud and Casper Dijkstra

With the shift from monolithic systems towards distributed systems, from on-premise towards the cloud, from running virtual machines to serverless computing, it has become indispensable to monitor properly in order to get a grip on the complete system.

At the same time, proper monitoring has become an ever increasing complex matter. Back in the old days performance issues could be predicted, such as full or broken HDDs, memory issues or CPU overload. Nowadays a different approach is required. Things will break down, only it cannot be foreseen when or where this will happen. This calls for a new approach regarding how the running state of our cloud solutions can be measured. Static dashboards or reactive alerting are not sufficient anymore. This is where Observability comes in.

Observability

The term "Observability" originates from control theory and can be explained as a measure of how well internal states of a system can be inferred from knowledge of its external outputs.[1] In the recently released (June 2022) O'Reilly handbook called "Observability Engineering", this formal definition is applied on software systems. The authors come up with a more practical approach:

"... our definition of "observability" for software systems is a measure of how well you can understand and explain any state your system can get into no matter how novel or bizarre."

In other words, when an issue occurs all tools and data are directly at your disposal to debug and mitigate the issue at hand. In the current age of complex production systems, traditional monitoring and alerting practices no longer suffice. One of the main disadvantages is that collecting and monitoring metrics, such as CPU or Memory usage of a VM, is fundamentally reactive. Furthermore, monitoring these kinds of metrics is based on previous known possibilities of where the system could fail (known-knowns).

But what to do if something unexpected happens; for instance, something not seen before and therefore not recorded in the existing monitoring? No alert will trigger and the engineer that eventually will have to fix the issue has to go and search for the needle in the distributed serverless haystack.
As the writers of Observability Engineering put it:
"In modern cloud native systems, the hardest thing about debugging is no longer understanding how code runs but finding where in your system code with the problem even lives."[2]

How to prepare for the "unknown unknowns"? This is the question Observability aims to answer. Instead of adding more and more metrics and alerts as new things break over time, observability turns around this approach by acknowledging that it cannot predict what will break next in the system.

[1] Majors C., Fong-Jones L., Miranda G (2022). Observability Engineering. Achieving Production Excellence, 4
[2] Majors C., Fong-Jones L., Miranda G (2022). Observability Engineering. Achieving Production Excellence, 13

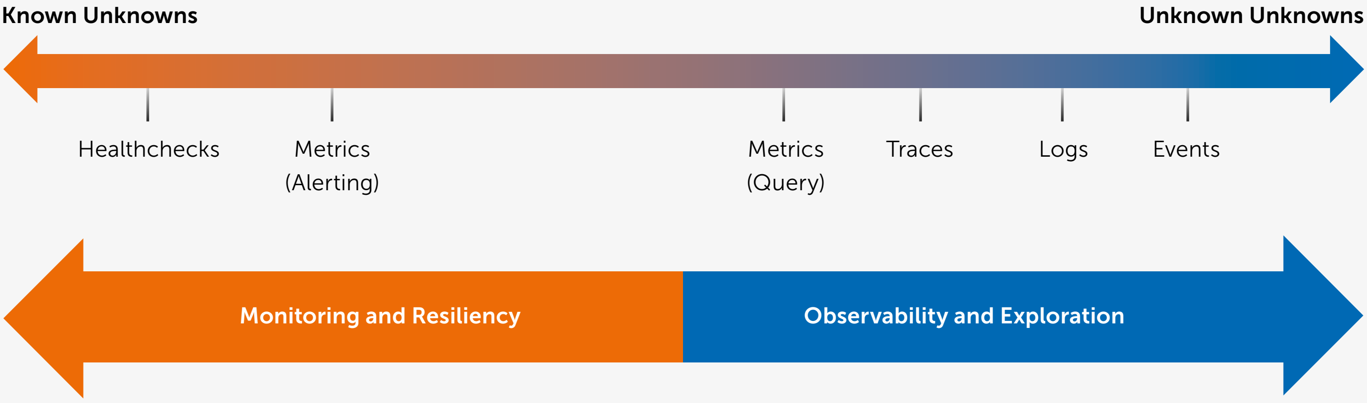


Figure 1. While monitoring and its tools aim to cover the Known Unknowns, Observability covers the realm of the Unknown Unknowns.

From this acknowledgement follows a new ultimate goal: collect as much relevant contextual data as possible. Collecting structured logs and events and making it possible to slice and dice through the data becomes pivotal. A curious pro-active attitude enables us to make previously unknown states of the system insightful.

New tools should be created that help us to preserve: **"... as much of the context around any given request as possible, so that you can reconstruct the environment and circumstances that triggered the bug that led to a novel failure mode."**[3]

Microsoft Azure and Observability

With the above theory in mind, Microsoft seems to embrace Observability. They also aim to provide centralized logs, metrics and traces. Azure Monitor promises to *"... deliver[s] a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments."*[4]

Azure Monitor aims to be the *"single pane of glass"* when it comes to observing performance of resources. From here one can dive into automatically aggregated logging, application insights, service health, or monitor a Kubernetes cluster.

With Azure Monitor and solutions like Application Insights at our disposal, Azure provides a very solid monitoring and logging solution. Just as the observability theory advises, it brings together a lot of important data and, more importantly, transforms this data into *structured events* that can be queried and zoomed into on a per request level.

So why the need for an additional tool on top of this? For many workloads (such as running App Services) one could argue that Azure provides a fruitful tool to your application in depth. Still, there are scenarios in which there are real benefits to adding Grafana to the toolset. As we will show in the rest of this article, it can be a powerful solution because it can do things not possible in Azure. But first: Why should one even bother to visualize data into a graphical representation called a dashboard?

Why use visualization?

One of the more obvious advantages is that one gives access to otherwise unseen events and status by making data visible, one gives access to otherwise unseen events and status. An important benefit of simplifying data into a graphical representation is that non-technical people get an idea of how the system is performing. In other words, the data is *democratized* and made available for the whole company.

A noteworthy nuance here is that some dashboards may be irrelevant for certain users. These dashboards are particularly useful when scoped to certain teams, which Grafana allows for. For instance, SLO dashboards can be made accessible company-wide while dashboards that help the debugging process, e.g. Kubernetes dashboards or technical error collections, can be made available for just the DevOps teams.

For developers or IT engineers who have the task to trouble-shoot an incident, a well-designed dashboard can be a great starting point for investigation. In case of a complex technical problem an elaborate dashboard can act as the Swiss army knife pointing the right direction, or at least verify that several metrics are within the realm of expectation. It can also tell *which factors can be excluded from the possible factors incurring the problem*.

The uitility of a dashboard stands or falls with the graphical display of the dashboard. Which metrics are shown? Which alerts are implemented? Naturally it depends on the workload where to focus, but in general it is wise to focus the measuring and alerting on events that have end-user impact. To be able to determine where this impact lies, it is important the business stakeholders - who know the value of their product is – are asked, what kind of problems or outages would be truly painful. Based on these kinds of conversations the Service Level Objectives (SLOs) can be constructed. From there the SLOs can be visualized and share through the company.

(To learn more on SLOs and how to use error budgets, read the article in Xpirit Magazine #11: "The reliability paradox: Why less can be more" by Geert van der Cruijsen and Casper Dijkstra.)

[3] Majors C., Fong-Jones L., Miranda G (2022). Observability Engineering. Achieving Production Excellence, 13
[4] See: Azure Monitor overview - Azure Monitor | Microsoft Docs https://learn.microsoft.com/en-us/azure/azure-monitor/overview

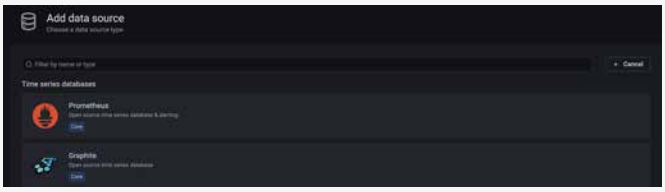
Azure Managed Grafana

Usability: combine multiple data sources and metrics

What makes Azure Managed Grafana an interesting addition to the toolbox? Let's start with usability. People may argue that Azure provides all the required tooling, however, Grafana is a specialized dashboarding tool with many visualization options, graph types and a user-friendly interface. By creating togglable data panes and metrics, correlations between different data can be investigated easily, e.g. see that performance degraded directly after a new deployment or that the application became unresponsive as soon as the number of unhealthy pods spiked. With Grafana, professional dashboards can be set up with insightful graphs. Instead of using the static tiles of an Azure dashboard or workbook, Grafana allows for free customization of the dimensions of the graphs with more options when it comes to the look and feel of the graph.
As said before, dashboards have a specific audience. That is why it is good to know that Azure Managed Grafana supports RBAC on the Azure and Grafana level. On the Azure side, Admin, Viewer and Editor permissions can be set on individual users and AD groups. Global permissions of who can modify dashboards and who has read-only permissions can be set as well.

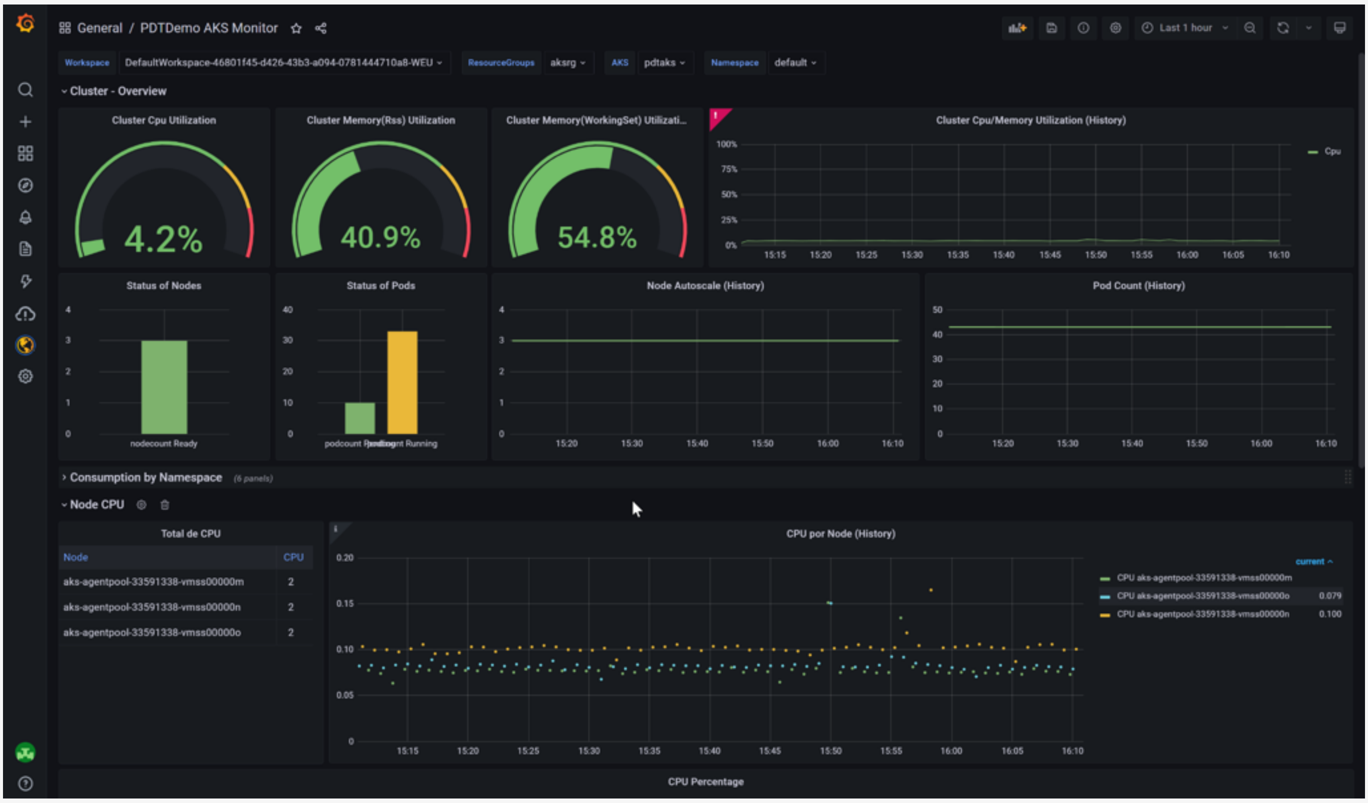
A powerful asset of using Grafana is the ability to combine multiple data sources. Think of scenarios in which data from multiple Azure subscriptions from different regions, different tenants *etc.* needs to be combined. This can all be done easily. The Azure Monitor data source authenticates using an Azure AD App Registration.

On https://grafana.com/grafana/plugins you can find more than 100 data sources that can be added to the dashboards. Using the Microsoft stack, we are particularly interested in the Azure, Azure DevOps and Prometheus plugins. Customers using Jira for ticket management or GitHub for development should know that those integrations are also supported.



Add data source
Choose a data source type
Filter by name or type
Cancel
Time series databases
Prometheus
Open source time series database & alerting
Core
Graphite
Open source time series database
Core

Take this one step further and go outside of Azure and it becomes even more interesting. Envision a hybrid infrastructure in which all kinds of data sources can be brought together, such as on-premise SQL Server, AWS or other external observability tools e.g. DataDog or Jaeger.



Total de CPU

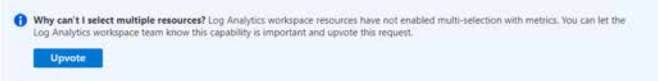
Node	CPU
aks-agentpool-33591338-vmss00000m	2
aks-agentpool-33591338-vmss00000n	2
aks-agentpool-33591338-vmss00000o	2

Figure 2. An example of a visualization of AKS cluster performance in Grafana. The dashboard consists of multiple so-called rows such as 'Cluster - Overview' and 'Consumption by Namespace'. The rows can be maximized and minimized to see just the data needed. All panels can be transformed freely and there are a lot of visualization options available. At the top variables are used to select a specific resource group or cluster. These variables are powerful because this one dashboard now can be used for all clusters in a particular infrastructure. This dashboard is available via: https://grafana.com/grafana/dashboards/12817

On the Grafana side, access to individual dashboards can be fine-grained; documentation can be found at https://grafana.com/docs/grafana/latest/administration/roles-and-permissions/access-control/.

In short, a lot of the relevant data can be brought towards one hub system. In this way the combined data can be visualized and it becomes a powerful first-stop to get a comprehensive overview of the performance of the complete infrastructure.

When trying to visualize multiple Azure metrics on one graph via the Azure Portal, the following message may have popped up more than once.



Why can't I select multiple resources? Log Analytics workspace resources have not enabled multi-selection with metrics. You can let the Log Analytics workspace team know this capability is important and upvote this request.

In the Azure Metrics graphs only one resource can be selected at a time. This contradicts good observability practice.
Often one would like to compare several resources to see how they correlate or differ in a certain time frame. This can give valuable insights into the root cause of issues.

This use case is something that is totally possible in Grafana. Multiple queries of different metrics can be added in one graph, table or other visualization of preference. An example could be an overview of response times of several app services or maybe to compare the amount of healthy pods in several AKS clusters. This can be done easily in Grafana, as can be seen in Figure 3 and Figure 4. where multiple metrics are shown on the same time-scale.

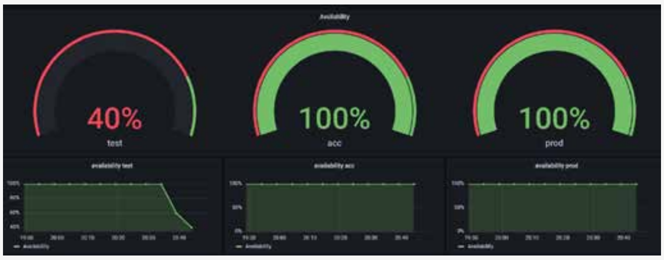
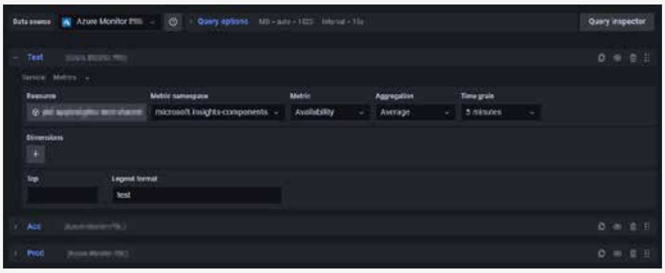


Figure 3. An example where availability statuses from multiple environments, in multiple subscriptions are combined in one panel.



Resource | Metric namespace: microsoft.insights-components | Metric: Availability | Aggregation: Average | Time grain: 5 minutes

Dimensions

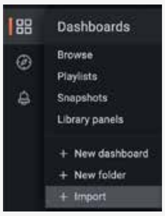
Top | Legend format: test

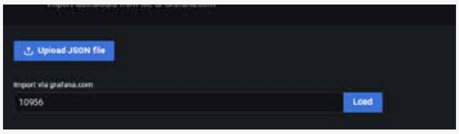
Figure 4. The 3 queries (Test, Acc and Prod) from the panel in figure 3. Add/remove queries and rearrange their order, pick any available resource and metric per query.

A Grafana dashboard is more than just a static visualization. It allows you to explore data and jump to Azure when further investigation is needed. Examples are links to more elaborate logging or to Azure DevOps pipelines for deployments that incurred diminished performance to initiate a rollback.
This makes a Grafana dashboard truly a starting point when issues arise. Quick insight can be gained into key indicators. When something strange is noted, one can explore the particular query, see what is going on and jump directly to the particular Azure resource via a link provided in Grafana.

Community

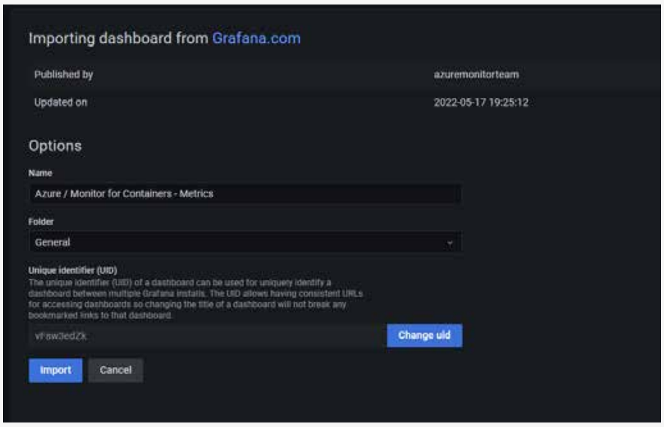
Since Grafana is a specialized cross-cloud monitoring tool, it has gained a lot of traction and contributions from the community. This can be seen in the great amount of available monitoring templates. An abundance of optimized dashboards can be found at https://grafana.com/grafana/dashboards, where each dashboard is given a unique dashboard identifier. This ID can be specified in the import functionality of Grafana and allows you to easily add a Kubernetes and/or cost management dashboard to your dashboard stack.





Import via grafana.com: 10956

Step 1: Go to Dashboards/ Import Step 2: Fill in the ID of a dashboard



Importing dashboard from Grafana.com

Published by: azuremonitorteam
Updated on: 2022-05-17 19:25:12

Options

Name: Azure / Monitor for Containers - Metrics

Folder: General

Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

vFsw3edZk

Step 3: Check the details of the dashboard and Import.

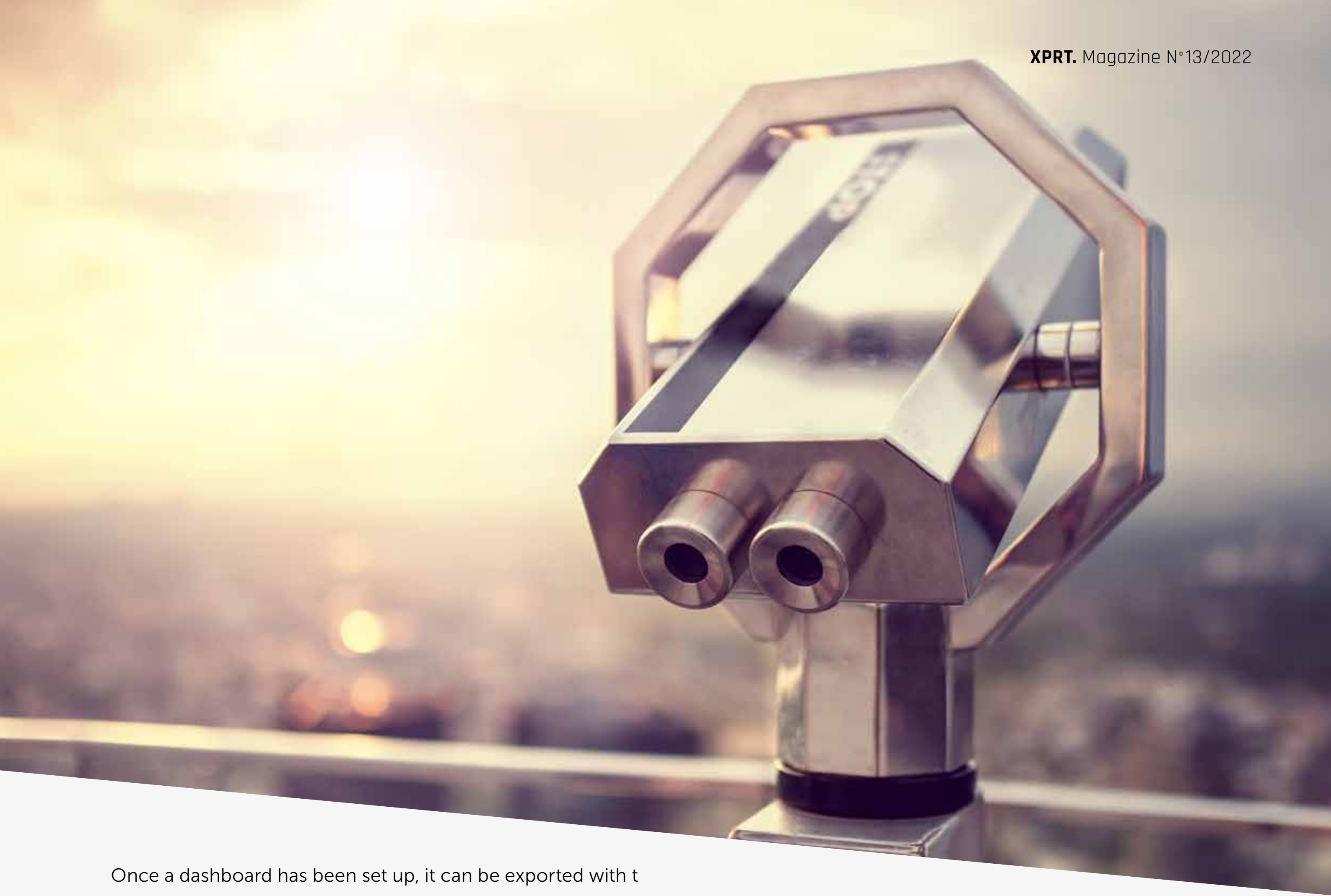
These template dashboards are great examples and provides a running start when someone is new to Grafana. They give a good example of what is possible for the specific data sources and can be tweaked to personal preference after import.

The lively and expanding Grafana community works as an enabler for adoption by more and more companies worldwide. We strongly think that Grafana is the visualization weapon of choice in, among others, the Kubernetes realm. See for example this article: How To Setup Prometheus (Operator) and Grafana Monitoring On Kubernetes (getbetterdevops.io)

Dashboards-As-Code

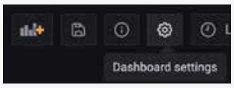
One of the important aspects of working DevOps is to automate everything. This includes dashboarding.
Grafana dashboards can be created from a JSON template using the Grafana API.

By treating your dashboards as code and including them in the DevOps work process you get all the advantages of automatic deployment, git versioning, collaboration et cetera. Let's have a more hands-on look of how this could work.

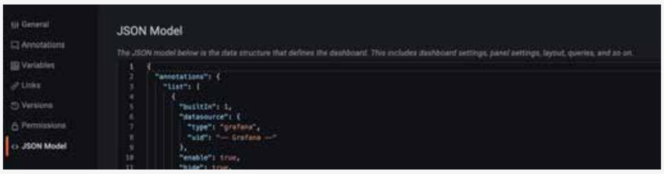


Once a dashboard has been set up, it can be exported with t he following steps:

1. Open the dashboard
2. Go to Dashboard settings



3. Open JSON Model



JSON Model

The JSON model below is the data structure that defines the dashboard. This includes dashboard settings, panel settings, layout, queries, and so on.

```
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": {
          "type": "grafana",
          "uid": "-- Grafana --"
        },
        "enable": true,
        "hide": true,
```

4. The Dashboard-As-Code can be copied.

Following these steps we can set up modular dashboards that integrate with miscellaneous services, such as:

> Azure DevOps
> App Services
> Azure Kubernetes Services
> Azure Monitor

After setting up an insightful dashboard, the environment specific parameters are parametrized, such as the subscription ID, resource group and managed identity used to obtain the data. By doing this, the dashboard can be reused for multiple scenarios, environments, or customers. By repeating this for different dashboards, a pipeline can be set up that combines the deployment of the different dashboards. This is accomplished as follows. First create a pipeline template to which the different dashboard panels can be fed.

```
parameters:
  - name: appService
    type: object
    default: ''
  - name: kubernetes
    type: object
    default: ''
  - name: costManagement
    type: object
    default: ''
  - name: namespace
    type: object
    default: ''
```

Subsequently, the Grafana pipeline template can be called from the environment-specific pipeline where parameters (e.g. a list of App Services and corresponding resource group) are passed to this pipeline. The pipeline then checks if parameters are set for that specific component (using the `condition: neq(variables.appService, ")syntax)`, sets the variables in the appService.grafana file and adds it to the total Grafana file that will be deployed. A portable dashboard is thereby obtained which only has to be devised once. Moreover, when changes are made in a customer dashboard, this can be put back in the infra-as-code such that it can be rolled out for other customers too. This way, dashboards can be incorporated in the DevOps workflow.

The utility thereof is not limited to managed services providers; larger companies with multiple DevOps teams can strongly benefit from such an approach too! These practices help to standardize the look and feel of dashboards and to create a uniform dashboard experience. Incorporating the dashboard-as-code technology allows companies to create well-designed dashboards and share them across teams for reusability and efficiency.

Costs

At the publication date of this article, Azure Managed Grafana has an hourly rate of €0,088 per hour per instance. Active users (users who accessed Grafana in a given month) are less than € 6,00 per month. So with +/- € 75,00 a month (and a 30-day free trial) the initial investment to start with this offer is fairly low. The Azure costs should not be a barrier to exploring this tool.

This has three big advantages.

1. Firstly, this prevents Egress
2. Data costs from Azure to the external tool;
3. As Egress Data is invoiced per GB this can cause unhappy surprises when such a tool is implemented.

Secondly no additional agents need to be installed on the Azure resources that could have an impact on performance and maintenance. Lastly there is almost no delay in data availability between Azure and Grafana. As soon as the data becomes visible in the Azure Portal, it is available as well in Grafana.

Conclusion

Azure managed Grafana allows us to bring dashboards to the next level and reuse dashboards that have been well-designed. The ability to concentrate and retrieve useful information at a centralized place greatly helps in the debugging process in case of technical errors.

Moreover, one can easily include information that is otherwise scattered across cloud services. Billing information is collected in Azure Cost Management, SLOs are defined in terms of KQL queries, and Kubernetes performance is monitored under the Azure Kubernetes Services monitoring tabs. Now all information is leveraged in a single place, where multiple dashboards, all with their own audience, can be defined.

By bringing all these data sources together and combining them in a smart and useful way in Azure Managed Grafana, observability becomes something real and will add value to the existing tools that Azure offers. In the current age of highly complex distributed systems it is no longer about only trying to prevent issues from occurring, but to make sure engineers have the right tools to literally observe what is going on and to locate the issue as quickly as possible.

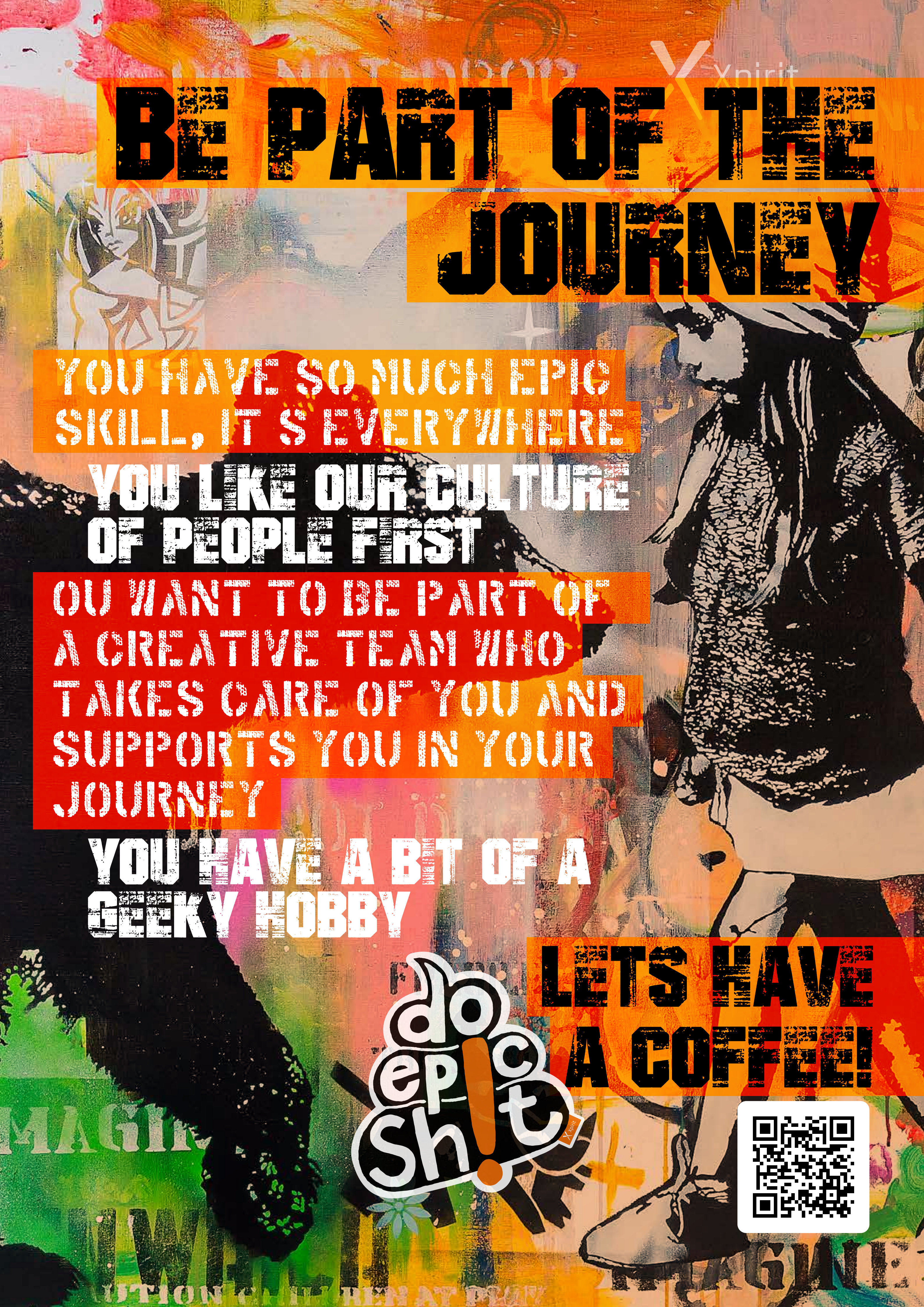
If you're interested in Azure Managed Grafana and would like us to help you with designing fit-for-purpose dashboards or would like to get a workshop on how to do this yourself, don't hesitate to reach out to us! </>

Rik Groenewoud
Consultant Managed Services

rgroenewoud@xpirit.com

Casper Dijkstra
Cloud Engineer

cdijkstra@xpirit.com



Xpirit

BE PART OF THE JOURNEY

YOU HAVE SO MUCH EPIC SKILL, IT S EVERYWHERE

YOU LIKE OUR CULTURE OF PEOPLE FIRST

OU WANT TO BE PART OF A CREATIVE TEAM WHO TAKES CARE OF YOU AND SUPPORTS YOU IN YOUR JOURNEY

YOU HAVE A BIT OF A GEEKY HOBBY

LETS HAVE A COFFEE!

do epic sh!t

www.xpirit.com

Together we drive change.

If you prefer the digital
version of this magazine,
please scan the qr-code.