XPRT.

Exploring new horizons: Identity Evolution

Five reasons why digital transformations fail

Is ChatGPT a Better Software Engineer than Me?

Identity Access Management in Microsoft 365

Zero Trust - "Never trust, always verify"

Preventing Identity Crisis in Azure

Xebia | Xpirit

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)

View all training options online.

Xebia|Xpirit

Colophon

Microsoft
Solutions Partner

GitHub Verified Partner

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

In this issue of **XPRT.** magazine our experts explore new horizons.

Vertigo

As Bono chants in the U2 song Vertigo: Unos, dos, tres, catorce! This is now also true for this brand new magazine, XPRT Magazine 14! We have come a long way since our first magazine in 2015. We went through various major and minor rebranding. And this is also true in this edition. You will see some major visual and structural changes to our magazine. This not only reflects our push for continuous improvement, but also reflects our own transformation at Xpirit. Although we have always been a proud part of Xebia, we recently transitioned from Xpirit to Xebia | Xpirit. This move represents a significant evolution in our organization's capabilities and scope, allowing us to offer a broader range of services to our clients and take on even more complex and challenging projects.

René van Osnabrugge

But no worries, our commitment to thought leadership and continuous improvement stays the same. We remain passionate about exploring new ideas, challenging ourselves and our clients, and finding innovative solutions to the most pressing challenges facing businesses today. So, as I hold this brand new XPRT magazine in my hands, I can't help but feel a sense of pride and excitement.

We did it again! As I read through these articles, I'm struck by the depth of knowledge and expertise our engineers and thought leaders possess. And I'm reminded once again of how fortunate we are to work in an organization that values thought leadership, innovation, and continuous improvement.

Twice a year, this magazine showcases the expertise and insights of our engineers and thought leaders, providing you, our readers, with practical advice and new ways of thinking about complex challenges. In this edition, we again gathered a broad range of articles, tackling everything from digital transformations to identity access management, from load testing to zero trust security models.

In this edition, we explore the latest tools and techniques for enhancing your software development practices. We look at how to add load testing to your CI/CD workflows by using Azure Load Test and GitHub Actions, examine why digital transformations often fail, and provide insights into how to upgrade your app to the future by migrating from WPF or Winforms to Blazor.

We also delve into the world of identity access management. We explore some of the key topics related to identity management on the Azure platform, including how to mock OpenID Connect Providers for testing, how to use Azure AD Access Packages to manage access permissions, and how to implement OAuth2 device flow-proxy using free Azure components. We also delve into the topic of zero trust security, examining how this approach can help organizations protect their data and systems from increasingly sophisticated threats. Finally, we explore ways to prevent identity crises in Azure, providing practical guidance on how to manage identity-related risks and ensure the integrity and security of your Azure-based applications and services.

And it would not be a XPRT magazine if we did not cover some cutting-edge software development topics, including EF Extensions and mutation testing in C#. We even take a playful look at whether ChatGPT is a better developer than you, showcasing the latest capabilities in AI and machine learning.

By organizing these topics under our 8 pillars of an engineering culture, we're making it clear that there is a lot of ground to be covered if you want to act like a software company. Much like the song Vertigo, which is known for its high energy and driving rhythm, XPRT magazine is designed to inspire and energize you to drive this transformation forward.

We hope this edition of XPRT magazine inspires you to explore new ideas, challenge yourself and your team, and take your Engineering Culture to the next level.
</>

Five reasons why digital transformations fail

In 2011, Marc Andreessen claimed in the Wall Street Journal that "Software is eating the world", predicting that the rise of software would digitally transform every industry and every sector of the world as we know it. Ten years later, there is nothing more to say but: he was right. Software has completely transformed our lives, and with that, every company, and every organization on the planet. It has fundamentally changed the entertainment industry with Netflix and Spotify. It has transformed travel and hospitality with AirBnB and Booking. It has transformed our cities with scooters, public transportation, and parking, that you can book with your phone in seconds. Every company is becoming a software company because the business models transform from selling products to creating experiences.

Author Michael Kaufmann

This means, that companies must change. Not only their business model – but also the way they work, the way they are organized, and the way they leverage IT resources. This transformation has many names. If we talk about the change of the business model, we talk about <u>digital transformation</u>. If we talk about the way of working, we talk about <u>DevOps transformations</u>. And, if we talk about transformation of IT, we talk about <u>Cloud</u> <u>transformation</u>. But they are all part of the same story. You can only succeed, when you master all three of them. The hard part is not digital, DevOps, or Cloud – the hard part is the transformation.

Common pitfalls – Why transformations fail
Many transformations fail – and it takes many companies more than one attempt to get it right. But in most failed attempts we can see the same patterns. If you want to succeed in your digital transformation, make sure to avoid these five common pitfalls:

1. Assuming your company or industry is special
Many customers that I meet believe that they are special, but they are not. And I'm sorry to say, it's probable that neither is your company or industry. At least, not when it comes to digital transformation. Could your product kill people if it has a defect? So could cars, airplanes, trucks, medical devices, and so on. And the same is true for all of the parts that are produced for these products.

They are nothing special. Do you have to you comply with certain standards? Do you create military products? Are you publicly traded? Do you work for governments? Whatever you think makes your company special, chances are there are many companies that face the same challenges that you do. The same rules apply to them as to you when it comes to your digital transformation.

If you look at the studies for DevOps transformations, you'll find they apply to all companies: from small start-ups to big enterprises and from cutting-edge internet companies to highly regulated industries, such as finance, healthcare, and government.[1]

And this is actually a good thing. This means a lot of the problems you're probably facing during your own transformation have already been solved by others. You can learn from their failures and don't have to experience them yourself.

2. Having no sense of urgency
The biggest blocker to change is complacency. If people in your business are complacent, they will tend to resist change and keep on doing business as usual.

You must establish a true sense of urgency for people to address critical things now. Urgency in this case does not mean pressure from management that creates anxiety. True urgency should drive people to change with a deep determination to win – not with anxiety about losing.[2]

Without a sense of true urgency, people will resist change and are more likely to keep their old behaviors. Note that a sense of urgency might arise for completely different reasons at distinct levels of your organization. Management might feel pressure from the market and the lack of agility to react with frequent releases. Engineers might feel the pressure of technical debt and the problem of attracting and retaining talent because of old processes and tools. It is important to align these stories to a common root cause using a clear vision. If you manage to align the different senses of urgency into a single force that drives in the same direction, you can ensure that the different forces will not neutralize themselves.

3. Having no clear vision
It is easy to replace tools, processes, and roles, but it is hard to change behavior, culture, and stories. Without a clear vision, the transformation will not yield the desired results.

If I hear customers say we are not Microsoft or Google or we are not a cutting-edge internet company, it tells me they are missing a clear vision. If your vision clearly states you want to become the digital leader in your industry or change from a product company to a service company, people will not dare to say things that contradict it.

A good vision to drive change is a clear and compelling statement of where all your transformation leads.[3]

I believe it is worth noting that transformations are not always driven by upper management. I know many companies where the transformation is driven by individual departments or even teams. Nevertheless, the same rules apply – you need a clear vision for the members in your team or department and to establish a sense of urgency to ensure the transformation is successful.

4. Let obstacles block your progress
When you start a transformation, many obstacles will block your transformation. Good examples that I often experience are certain regulations in certain industries. Many regulations, such as ISO26262 or GxP, propose the V-Model for software engineering. The V-Model is based upon the waterfall model, so it contradicts basically everything we have learned in many years of research. If you insist on keeping the waterfall model, your transformation will most likely fail, due to your internal interpretations of the regulations. If you have a closer look at them, you'll realize they just insist on best practices. If your practices are superior to the recommended ones, you can justify that and still pass an audit.

Most obstacles you'll encounter are caused by your organization, for example, your organizational structure, tight job categories, processes, or trench warfare between the working council and management. Don't permit these obstacles to block your transformation.

5. Not getting help
Consultants have a bad reputation in many companies, mostly because of bad experiences.
Often their appearance is associated with layoffs and other tough decisions. But if you want to learn a new sport, you don't just buy the equipment and watch some videos on YouTube. You join a club that provides a professional trainer or find yourself a coach that will guide you. Sports are not just about knowledge and tools – they are about building skills. And without an experienced coach, it is hard or impossible to succeed in certain sports.

[1] Forsgren N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations (1st ed.) [E-book]. IT Revolution Press. Page 22
[2] John P. Kotter (2008), A Sense of Urgency, Harvard Business Review Press
[3] John P. Kotter (2012), Leading Change, Harvard Business Review Press

The same is true for building new skills and capabilities in your business. There is no shame in getting help from someone more experienced who can guide you through the change. The odds are high that help will be cheap based on what you save in time and effort, never mind the costs of failure.

HOW TO SUCCEED WITH YOUR DIGITAL TRANSFORMATION

Start with Why
For a transformation to succeed, you need a clear vision and a sense of urgency. The vision should be precise, compelling, short, and should inspire people to follow it. To communicate the vision, you can follow the Golden Circle[4] and communicate from the inside to the outside (see Figure 1).
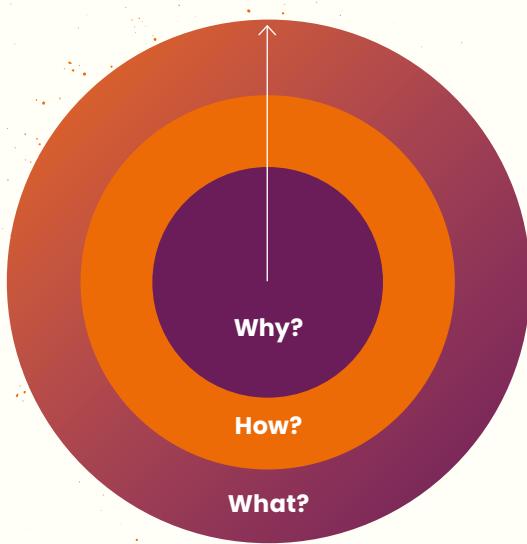

Figure 1: Communicating a vision should start with Why?

Let us see the layers of the circle in more detail:
- **Why?:** The reason why your company will undertake the transformation. This gives it a purpose and establishes a sense of urgency. Why should anyone care?
- **How?:** How are you going to succeed in the transformation process?
- **What?:** The actual thing that you want to transform. What are you doing or making?

A purpose-driven mission
Don't underestimate the power of vision! If you are a manufacturer of combustion engine cars, transformation to electrical cars will not come easy. There will be resistance. People will be afraid to lose the power of their jobs.

To succeed, you need a clear vision and to communicate the Why? – like the Volkswagen Group in its goTOzero mission statement in 2019, which concentrated on four main fields of action: climate change, resources, air quality, and environmental compliance. By 2050, the entire Volkswagen Group wants to become balance sheet CO2-neutral. By 2025, the company plans to reduce the carbon footprint of its fleet by 30 percent over its entire life cycle compared to 2015[5]. This perfectly explains the Why?, establishes urgency, and fits into their overall updated vision to make this world a mobile, sustainable place with access to all the citizens. Equally, Mercedes-Benz stated in their Ambition 2039 statement from 2019 that they aim to have a carbon-neutral car fleet and production over the next 20 years.[6]

And it is the same when you transform a product company into a software or services company. Even if you only transform from a waterfall organization to a DevOps organization, people will be afraid of the change and there will be resistance if you cannot paint a picture of a desirable future and explain why you have to undertake the transformation.

Establish an engineering culture
Having a purpose-driven vision will help you to establish an engineering culture during your transformation: an inclusive and secure organizational culture that fosters talent and is driven by sharing and equality. My colleagues wrote about this in XPRT Magazine #12.[7]

This is a culture where people feel safe to speak up when they feel something is wrong, a culture where people feel safe to experiment and be creative without fear, and a culture where everyone feels welcome and safe – independent of heritage, gender, or religion.

The culture of an organization is a set of shared assumptions that guides behaviors within the organization[8]. That's why it is hard to change it. Creating PowerPoint slides with values and mission statements might affect the culture but maybe not in the way management intends to.

---

[4] Simon Sinek (2011), Start With Why – How Great Leaders Inspire Everyone to Take Action, Penguin, p.38
[5] Volkswagen (2019): Volkswagen with New Corporate Mission Statement Environment "goTOzero": https://www.volkswagenag.com/en/news/2019/07/goTOzero.html
[6] Mercedes-Benz Group Media (2019): "Ambition2039": Our path to sustainable mobility: https:// group-media.mercedes-benz.com/marsMediaSite/ko/en/43348842
[7] de Vries, M., & van Osnabrugge, R. (2022): Together we build an Engineering Culture. XPRT Magazine #12: https://xpirit.com/together-we-build-an-engineering-culture/
[8] Ravasi, D., & Schultz, M. (2006). Responding to organizational identity threats: Exploring the role of organizational culture. Academy of Management Journal.

As an engineer, you might ask yourself why the organization's culture matters to you. Isn't that a task for management? However, the culture is the result of the assumptions and the behaviors of every single person in the system – and that means every single person can change it. As an engineer, you should be aware of your culture and you should speak up if you see that something is wrong. Start doing the right things and telling the right stories.

Culture is best ingrained into corporate behavior using little quotes and principles that have a deeper meaning. They are easy to remember and encourage people to do the right things. Here are some examples you will often hear in companies with great engineering cultures:

- **Ask forgiveness, not permission:** Encourage people to do the right thing, even if it is against current rules or processes.
- **You build it, you run it:** Establish end-to-end responsibility and ownership for the things built.
- **Fail early, fail fast, fail often (or fail fast, roll forward):** Try to fail early and fast instead of making everything 100% bullet-proof.
- **Embrace failure:** Encourage people to experiment and take risks and ensure blameless learnings from failure. Take responsibility and don't blame others.
- **Collaborate, don't compete or work together not against:** Foster collaboration – across organizational boundaries and also with customers and partners.
- **Go fix:** Encourage people to take ownership and fix things instead of just complaining, but you have to ensure that innovation is not suppressed. Make sure people are also empowered to really fix the things they complain about.
- **Treat servers like cattle, not like pets:** Encourage people to automate everything.
- **If it hurts, do it more often:** Motivate people to practice things that are hard to build up the skills to accomplish. This phrase is often used in relation to releasing or testing applications.

These are just a few examples. More stories and sayings will arise when you transform your culture and become a digital company.

A great engineering culture is not just the responsibility of management. They have to let it happen and provide the vision but the best culture is then created by the people themselves during the transformation.

Data-driven transformation

If you want your transformation to succeed, it is critical to measure the right metrics and to prove the transformation really yields better results than the old system. Every metric or Key Performance Indicator (KPI) you measure will have an impact on the behavior. That's why it is important to measure metrics that matter[9] that allow you to optimize the right things first and to achieve small wins that will help you keep everyone motivated to continue with the transformation. Measuring the right data should always be the start for a transformation.

Optimizing something that is not a constraint is a waste of resources and can even have a negative impact. Objectives and Key Results (OKR)[10] is a flexible framework to define and track objectives and their outcomes. It is used by many digital companies, among them Google, Microsoft, Twitter, and Uber.OKR helps organizations to achieve a high alignment on strategic goals while keeping a maximum level of autonomy for teams and individuals. That's why it is a good framework for supporting your digital transformation.

A data- and AI-driven transformation is not a big-bang project that can be planned for a fixed finish date. Change is a long process – so your digital transformation is rather a way than a goal. Led by the digital vision – the WHY – the transformation can be separated into 3 phases (see Figure 2):

- **Digital Strategy:** Start by defining a Cloud, AI, and DevOps strategy. Gather company and market benchmarks.
- **People, Process & Culture:** Build your first cross-functional teams around customer needs. Enable them to deliver value end-2-end and adjust the accompanying processes.
- **Scale and optimize:** Create more teams around customer journeys and make sure to measure if the transformation is successful and yields the expected results.

Adoption and change management are an important pillar that accompany the entire change process.

---

[9] Michael Kaufmann (2022). Accelerate DevOps with GitHub: Enhance software delivery performance with GitHub Issues, Projects, Actions, and Advanced Security. Packt Publishing. P. 14

[10] Doerr, J. (2018). Measure What Matters: OKRs: The Simple Idea that Drives 10x Growth. Portfolio Penguin

Digital Vision
Fundamental reason, the Why.

Digital Strategy
Cloud strategy, DevOps strategy, company and market benchmark.

Metrics
Defining metrics. Gather data.

People, Process and Culture
Building Pilot teams around customer needs, defining accompanying processes, Governance.

Scale and optimize
Move teams over. Optimize outcomes.

Adoption, Change and Communication

Figure 2: Phases of a data-driven digital transformation

Summary

Change is always hard and that is the reason why many enterprise transformation fail – not only the digital transormation. But there are some typical pitfalls that can be avoided. If you think that one of these pitfalls apply to your company – hit refresh and start with a clear vision and a good strategy. And, there is no shame in getting help from someone that can guide and coach you. Get yourself a partner you trust that has experience with helping other clients succeed in their transformation.
</>

Michael Kaufmann

"Change is hard because people overestimate the value of what they have and underestimate the value of what they may gain by giving that up."
— **James Belasco and Ralph Stayer**

READ MORE ONLINE

Is ChatGPT a Better Software Engineer than Me?

When Alan Turing proposed the idea of a machine that could simulate any human intellect, he was met with raised eyebrows and skeptical chuckles. Fast-forward to today, and we find ourselves surrounded by a veritable army of digital masterminds that can do everything from ordering pizza to mastering the ancient game of Go. The question now is, will ChatGPT be the software engineer that steals our jobs or makes us better at them?

Authors Geert van der Cruijsen and Thijs Limmen

The March of AI Through History

The AI journey began with the notorious Turing Test, which postulated a machine that could imitate human conversation so accurately that it would be indistinguishable from the real thing. The quest for such a machine gained momentum during the Dartmouth research project in the 1950s, and today, we have five main types of AI: reasoning, natural language processing, knowledge representation, planning, and perception.

Up until now, most AI solutions excelled in one area of intelligence. But now we've arrived at the age of ChatGPT, the multifaceted wonder that combines reasoning, natural language processing, knowledge representation, planning, and perception.

ChatGPT: The Jack of All Trades

ChatGPT isn't just your run-of-the-mill AI; it's a Swiss Army knife of digital engineering. It can write application code, craft unit tests, and create pipelines to build and deploy it all. In fact, it even generates the infrastructure as code, leaving us mere mortals to wonder, "Is ChatGPT a better software engineer than me?"

Let's have a look at certain strengths and weaknesses of ChatGPT.

Geert van der Cruijsen

Thijs Limmen

Strengths:
- **Inspiration:** ChatGPT can generate new ideas, igniting creativity.
- **Empowering:** This AI boosts productivity, equipping developers to tackle complex tasks.
- **Versatility:** ChatGPT can wear many hats, from a programmer to a graphic designer.

Weaknesses:
- **Dreaming:** ChatGPT might generate unrealistic or infeasible ideas.
- **Slow results:** Sometimes, it takes a bit of time to receive useful outputs.
- **Confidence and reasoning:** ChatGPT can be confidently wrong and lacks human-like reasoning abilities.
- **Biases:** As with any AI, biases can seep in, affecting the quality of results.
- **Rudeness:** On occasion, ChatGPT might be unintentionally impolite.

Despite these weaknesses, the consensus is clear. It is a great tool to empower you as a developer. So, for now, the answer to the question "Is ChatGPT a better software engineer than me?" is that AI probably won't replace you. However, if you don't use AI tools, someone who does use AI will replace you.

Midjourney - /imagine running::3 overtaking::2 exponential increase --ar 16:9

We can position AI like pair programming with an "Experienced Expert Junior Developer". There is a lot of knowledge and information being generated for you, but you still need to connect the dots and know when it is right or wrong.

Ask ChatGPT, Skip Google: Streamlining Your Search for Answers
In today's fast-paced digital world, information is just a few clicks away, and search engines like Google have long been the go-to resource for answering questions and seeking advice. Unlike traditional search engines, ChatGPT is designed to understand context, allowing users to ask questions and receive relevant, nuanced, and human-like responses. By engaging with ChatGPT, users can bypass the time-consuming process of sifting through search engine results, while benefiting from personalized and comprehensive assistance. So, the next time you find yourself in need of quick, reliable information, consider turning to ChatGPT.

Next, let's explore how to become an AI powered developer.

Becoming an AI powered developer: the art of prompt building
To get the most out of ChatGPT, one must master the delicate art of prompt building. Just as an artist carefully selects their brushes, so must developers fine-tune their prompts to elicit the most useful and relevant responses from ChatGPT. An example of prompt building is as follows:

```
I want you to act as ... (job title, artist, role)
I will ... (give you x as input) and you will ... (what AI must do)
I want you to ... (more context, limits)
```
https://prompts.chat contains more than 500 prompts as examples.

Next, Let's explore some AI tools that can elevate your coding prowess and ensure you stay ahead of the curve.

A DEEPER DIVE INTO AI TOOLS FOR THE MODERN DEVELOPER

GitHub CoPilot X
GitHub CoPilot X is an innovative, AI-powered code assistant designed to enhance the software development process. By offering context-aware code suggestions, this sophisticated tool allows you to expedite your coding tasks while simultaneously reducing errors. CoPilot X goes a step further by incorporating a ChatGPT-like experience within Visual Studio Code and all IntelliJ IDEs, ensuring developers receive relevant and context-specific answers to their questions.

Acting as a knowledgeable partner, CoPilot X provides invaluable assistance and guidance throughout every phase of the coding journey. Its seamless integration with popular development environments ensures that you have access to expert advice and support whenever you need it. In essence, GitHub CoPilot X is the ultimate coding companion, helping you navigate the complexities of software development with ease and efficiency, serves as an informed companion, expertly guiding you through each stage of the coding process.

Bing Browser with Chat AI

Bing has integrated AI-powered Chat into its browser, allowing you to ask questions, get coding assistance, and even troubleshoot issues. This smart feature can be a game-changer when you're stuck on a coding problem or simply want to brainstorm ideas.

AI Commits

The "aicommits" command line tool is a handy utility that generates descriptive and informative commit messages for your code repositories using AI based on your staged changes. This tool ensures that your commit history is well-documented and easily understandable, making collaboration and code review a breeze.

By incorporating these AI tools into your workflow, you'll not only enhance your productivity but also ensure you stay ahead in the ever-evolving world of software development. Remember, AI probably won't replace you, but those who use AI tools effectively will have a competitive edge in the industry.

AI-Powered Image Generation Tools for Design Inspiration

Midjourney: An AI-driven image generation platform, Midjourney helps designers create stunning visuals by generating unique images based on descriptions or keywords. It can give you inspiration for your UI designs, create logos, UI components or even complete landing pages.

These AI-powered tools can significantly enhance your coding and design capabilities, helping you stay competitive in the world of software engineering.

/imagine Awesome application icon for a cyberpunk app, design, ux/ui, ux, ui --v 4 --q 2"

/imagine Beautiful application icon for a dating app, design, ux/ui, ux, ui --v 4 --q 2"

/imagine Beautiful application icon for a trading app, design, ux/ui, ux, ui --v 4 --q 2

Dall-E: Another image generation tool is developed by OpenAI. Dall-E generates images based on textual descriptions, providing ideas for icons, artwork, or other design components.

/imagine Beautiful landing for a travel website, design, ux/ui, ux, ui, behance, dribbble --ar 3:2 --v 4 --q 2

CoPilot for Office 365 products: A New Era of Productivity

Embracing AI's potential extends beyond software engineering. With CoPilot for Office 365 products like Word, Excel, and PowerPoint, users can now experience an unprecedented level of productivity. This intelligent assistant offers real-time suggestions for text formatting, data analysis, and presentation design. In Word, CoPilot can help draft content, generate outlines, or even proofread your document. In Excel, it can suggest formulas, create charts, or analyze data trends. And in PowerPoint, CoPilot can recommend slide layouts, color schemes, and even provide ideas for impactful visuals. With CoPilot's seamless integration into Office 365, you can now harness the power of AI to elevate your work across these essential productivity tools.

Conclusion

AI probably won't replace you. However, if you don't use AI tools, someone who does use AI will replace you. To stay competitive, adopt a mindset shift to asking ChatGPT for help, master prompt building, and equip yourself with AI-powered tools to become a more powerful productive developer.

Not convinced yet? This article was written by ChatGPT-4 with some extra help from Thijs & Geert. Want to know how? We've used the following prompt to generate 90% of this article:

```
Write a creative and humorous yet formal magazine
article about Microsoft Technology, with title
"Is ChatGPT a better software engineer than me?"
Discuss:

AI history:
• Turing Test
• Dartmouth research project
• 5 types of AI (reasoning, natural language
  processing, knowledge representation, planning,
  perception)

ChatGPT: a combination of all 5 AI types

ChatGPT can write application code, unit tests, but
also pipelines to build and deploy it. And even it
creates corresponding infrastructure as code.

Strengths of ChatGPT like Inspiration, Empowering,
Be any profession

Weaknesses of ChatGPT: Dreaming, Slow results,
Biases, Rude, Is sometimes confident wrong and it
doesn't have reasoning likes humans do.

Conclusion is that AI probably won't replace
developer, but developers will be replaced that
don't use AI tooling. Highlight this exact quote:
"AI probably won't replace you. However, if you
don't use AI tools, someone who does use AI, will
replace you."

Becoming a better developer using AI and the
mindset switch from Google to ChatGPT

Importance of prompt building for ChatGPT

AI tools to help with coding.
• Bing browser with Chat AI
• Github CoPilot X
• "aicommits" command line tool
• CoPilot for Office 365 products

Image generation tools for creating inspiration
for UI, Icon, artwork:
• Midjourney
• Dall-E

AI applications: blogging,
documentation, tutorials

Key takeaways
</>
```

READ MORE ONLINE

THE PROBLEM

Mutation Testing in C#

Let's face it, software development is hard. It's a highly creative task that fully takes place in "non-physical worlds" like our mind and inside IT devices. As physical human beings, we live in the real world, we experience the real world, we breathe and speak the real world. The direct consequence is that we learn from all the tiny things that might happen. We know to be cautious with a fresh cup of coffee, based on past experiences, as it might be quite hot.

Author Michael Contento

With software this is a bit different. Sure, we also gain experience over time. We learn to anticipate situations and re-use knowledge from the past, but we cannot easily transfer previous "real world knowledge" to our profession. This is a major difference to other jobs like carpentry or painting, where our human real-world judgement can be applied a bit easier. I mean, you don't have to be an experienced carpenter to verify if a chair does its job of carrying a human being.

Testing or verifying software on the other hand adds yet another complexity level to our construct in the non-physical world. If your primary code is already quite complex, how do we keep our unit tests simple? Refactoring our primary code becomes easy with a good set of unit tests, granted. But how can we refactor our unit tests? Are we sure that, after a refactoring, our tests yield the same level of confidence / security? Can we be sure that our tests always evolve with the primary code? Maybe, just by accident, a few small bugfixes in the past were made without a companion unit test. Who knows?

Measuring quality

So how do we evaluate the quality of our unit tests? Sure, simple gut feeling would be easy but also highly subjective and nothing we could add to our CI pipeline. Gathering some code coverage metrics while running our unit tests is, on the other hand, something we could easily add to our CI pipeline and would give us some objective numbers. But how do we interpret those numbers?

Coverage metrics only tell you what percentage of your code has been executed. Not what percentage of the business logic behind those lines of code have been evaluated!

And in combination with coverage metrics, you quickly hear or read some guidance like "70% coverage is enough, as 100% is not worth the effort". Why shouldn't we strive for 100%? Why do we have to be careful when interpreting those numbers?

Aren't there better metrics available? Maybe something with a high developer experience that focuses on actionable things instead of theoretical values? We developers like to improve things and not argue about numbers!

Mutation testing to the rescue

Usually, we use unit tests to evaluate our primary code, but with Mutation Testing we turn things upside down! We mutate our primary code to actively break or invert the existing behavior and test if our unit tests are able detect

---

this breaking change. If the unit tests pass, then we know that the original behavior was not properly covered by a test, and we need to rework / sharpen our tests in this regard.

This has the significant benefit of being very hands-on. Because the output of a Mutation Testing run is always "when I break this part of your primary code, no unit tests complain!". No abstract number to interpret. No softening "70% is good enough". Mutation Testing can either find places where you have gaps in your unit tests or not. It's as simple as that.

HOW DO WE UTILIZE THIS IN C# STRYKER.NET IS HERE TO HELP

To make things more concrete let's start with a short piece of code:

```csharp
public class Calculator
{
    public int Multiply(int a, int b)
    {
        return a * b;
    }
}
```

Yes, this is a very simple class and truly made up for this article. This piece of code is here just to convey the idea and usage of Stryker.NET[1] and Mutation Testing in general. Even in this scenario, we try to be good developers who care about quality. Therefor we also have a corresponding unit test that looks like this:

```csharp
[TestCase(1, 1, 1)]
public void Multiply_test(int a, int b, int c)
{
    var calc = new Calculator();

    var actual = calc.Multiply(a, b);

    Assert.AreEqual(c, actual);
}
```

Here we have a simple piece of code and a unit test that executes it. Our unit test is green, so everything is fine, right? If we would apply our code coverage metric from before, we would be at 100%! Great.

Let's see what Stryker.NET thinks about our project. For that we quickly need to install the `dotnet-stryker` command line tool via:

```
$ dotnet tool install -g dotnet-stryker
```

As you can see, Stryker.NET is a simple NuGet package that can be installed globally on your machine (like we just did) or project locally. Which way you prefer is, in the end, a matter of test and/or project convention. Once installed we can execute Stryker.NET against our code and see the results:

```
$ cd path/to/your/solution/folder
$ dotnet stryker
```

You didn't expect it to be that simple, did you? Stryker.NET tries its best to maintain a high-quality developer experience and will handle as much as possible. There are multiple command-line options available to change the default behavior, such as filtering mutations to a subset of your files, changing the output level, selecting the type of reports to generate, and much more. But for now, we can leave it at the defaults and open the HTML-based report, which is generated by default:

Arithmetic mutation Survived (7:16)

Here we can see that Stryker.NET mutated our original code by replacing the multiplication with a division and our unit tests were still green! Or in Stryker.NET words: the generated mutant was able to survive (no failing unit tests that caught him).

This is true, as our unit test only tested with a limited parameter set! We can do the mutation ourselves, totally invert the business logic and our test does not guard us. Improving our unit test is as simple as adding another parameter variant:

```
[TestCase(1, 1, 1)]
[TestCase(4, 2, 8)] ⟵
public void Multiply_test(int a, int b, int c)
{
    var calc = new Calculator();

    var actual = calc.Multiply(a, b);

    Assert.AreEqual(c, actual);
}
```

In the next round of `dotnet stryker` this mutant would no longer survive, and we actively improved the quality of our test!

Things that Styker.NET mutates
We saw that Stryker.NET was able to mutate our multiplication with a division and the question is now: What else can Stryker.NET mutate? Because in the end, the amount and diversity of those mutations define the spectrum and quality of the generated mutants.

The good news here: The number of available mutations in Stryker.NET is staggering and spans multiply categories:

Category	Original	Mutated
Arithmetic operators	+	-
Equality operators	!=	==
Logical operators	and	or
Boolean literals	true	false
Assignment statements	+=	-=
Initializers	new int[] {1,2}	new int[] {}
Unary operators	-var	+var
Update operators	var++	var--
LINQ methods	First()	Last()
String operators	"foo"	""
Bitwise operators	<<	>>
Math operators	Floor()	Ceiling()
Null-coalescing operators	a ?? b	b
Regex operators	abc{5,}	abc{4,}
Removal mutators	break	(simply removed)

As you can see, the list is huge! And I picked only one example out of every category. For a full list of all supported mutations, you should look at the documentation, which is very detailed. If you have any questions, the documentation always has you covered - not only for a list of all mutations.

Mutation score as KPI (Key Performance Indicator)

Stryker.NET will create mutants and count how many of them managed to escape or were caught by our tests. This information can be condensed down to a single score: The mutation score.

The calculation is simple, as we just divide the number of caught mutants by the total amount we created. Given we have created 120 mutants and only 5 of them survived, we get a mutation score of 92% (the higher the better).

This simple score is also visible in the various reporting formats that Stryker.NET can generate. In the default HTML report that we used earlier, we can use this as an uncomplicated guide to find classes that have more escaping mutants and thus less effective unit tests.

Conclusion

Mutation testing turns the world upside down and uses the primary code to evaluate the quality / completeness / robustness of our unit tests. It does so by spawning an army of mutants (logically inverted variants of our primary code), which must be caught by our existing unit tests. Every mutant that escapes (does not trigger a failing unit test) highlights a piece of logic within our primary code that does not have a verifying unit test.

In the end, this methodology is, as a software developer, very hands-on and creates actionable insights. If at some point Stryker.NET is no longer able to create mutants that survive our unit tests, chances are high that future-me can also not accidentally create mutants in the next refactoring. And this is what I really care about: Trustworthy unit tests. </>

Michael Contento

READ MORE ONLINE

Mock your OpenID Connect Provider

An article that teaches you how to test your endpoints using OpenID Connect. All without changing or mocking the authentication and authorization configurations in dotnet 6.

Author Kristof Riebbels

When creating new applications or updating existing applications, we have to take security into account. The protocol we will use in this article is OpenID Connect flow.

The OpenID Connect flow is a protocol used for authentication and authorization between different parties, such as a client application and a server providing identity services.

Having a realistic testing environment early on allows for more accurate testing of the application's behaviour. This can help identify issues related to the authentication and authorization process that may not be apparent in a test environment with different settings.

It will ensure that the application is secure and compliant with industry standards on a CI-pipeline.

A tale on how it can help you...

A customer required support for two identity providers to access our resource server, and our developer implemented the corresponding authentication schemes. Later, the requirement changed to support only one identity provider, and the relevant code was removed. Everything seemed to be functioning well until another team attempted to access the resource server, unaware that the second identity provider was no longer supported. In this situation, a HTTP status code 401 (Unauthorized) would be expected. However, due to incomplete code removal, a HTTP 500 (Internal Server Error) was encountered instead. By incorporating the testing methods outlined in this article, we gained insight into what happened and have fixed the situation.

Testing isolation

There are various categories of tests, each serving distinct purposes: Unit Testing, Integration Testing, System Testing, Acceptance Testing, Performance Testing, Load Testing, Stress Testing, Security Testing, Usability Testing, Regression Testing, Smoke Testing, Compatibility Testing.

Incorporating integration tests with mocked external dependencies in a Continuous Integration (CI) pipeline is essential. Those tests are also focused on the interactions between your application and the external systems.

A stable and predictable environment is created on your own machine and pipeline, as they remove the risk of external service outages or changes affecting the test results. You can debug and recreate problems more easily.

The key objective of testing is to ensure the delivery of quality on all different kind of aspects. However, by employing Test-Driven Development (TDD) and utilizing representative, non-trivial data, developers gain a clearer understanding of the business and the code's functionality.

To WireMock or not to WireMock, that is the question

To mock out an OpenID Connect provider, we need to simulate the behavior of the provider's endpoints that are involved: JSON Web Key Set (JWKS) endpoint /jwks and the OpenID Connect discovery endpoint /.well-known/openid-configuration. The description of these endpoints is found in another paragraph below.

For validating /weatherforecast endpoint, there should be an algorithm to generate valid and invalid JWT's.
Those generated tokens will be used in the request.
Finally, the resource server validates the request.

There are multiple strategies to mock endpoints:
• Using a library like WireMock.NET allows us to create stubs for HTTP requests and responses easily. By creating JSON documents that mimic the responses of a real OpenID Connect provider, we can define the mock endpoints' expected behavior. XPRT Magazine issue 13 discussed WireMock.NET and its setup. However, WireMock.Net does not help in providing valid and invalid JWTs; it specializes in mocking HTTP dependencies.
• Another solution involves manipulating the `ConfigurationManager` of the OpenID Connect settings and using a custom `HttpClientHandler`.

In both cases, these solutions intercept the HTTP requests to these endpoints and return pre-defined JSON documents as responses.

For this article, the `ConfigurationManager` is manipulated by providing a concise, straightforward solution. To maintain focus on creating tests, setting up the web application with policies, and preparing the necessary boilerplate without third-party libraries, we avoid adding complexity and extra dimensions.

Source code

This article includes code snippets and diagrams to provide readers with a clear overview. I encourage you to experiment with the code. You can find the source code at the following location: https://github.com/kriebb/MockOpenIdConnectIdpAspnetcore.

JWT tokens

The OpenID Connect protocols leans on the OAuth2 protocol. The authentication and authorization mechanism they provide makes use of JWT. So what is a JWT (JSON Web Token)?

A JWT Token is a compact and self-contained way of transmitting information between two parties in a secure manner.

A JWT token consists of three parts, separated by dots: a header, a payload, and a signature. The header contains information about the type of token and the algorithm used to sign it. The payload contains the information that needs to be transmitted, such as user ID or permissions. The signature is used to verify that the token has not been tampered with during transmission.

Here is an example, decoded by https://jwt.io

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4g
RG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF
2QT4fwpMeJf36POk6yJV_adQssw5c

```
header:
{
  "alg": "HS256",
  "typ": "JWT"
}
body:
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

The algorithm used to sign the JWT token is specified in the header. There are several algorithms that can be used, such as HS256, RS256 and others... RS256 stands for RSA-SHA256, which is an asymmetric encryption algorithm that uses a public key for encryption and a private key for decryption.

The public keys to validate JWT's are provided by the OpenID Connect Provider. The resource server should be able to access the public keys so the JWT's can be validated.

### Why is a Private Key of a certificate called private?
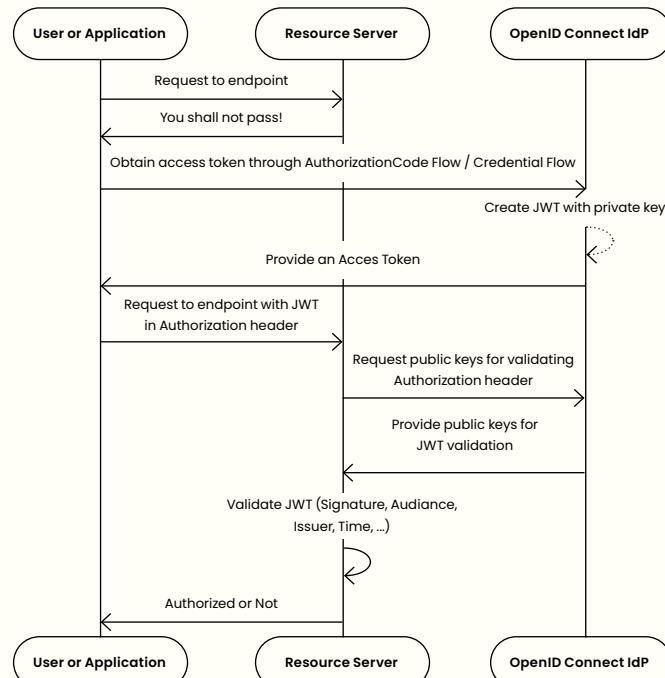
In this article, we will generate a self-signed certificate to help us generate valid and invalid JWT. It is crucial to remember that in production and testing environments, OpenID Connect IdP Providers' certificates should be kept confidential and not exposed. Additionally, there should be a mechanism in place to refresh the certificates periodically.

Attackers can impersonate the IdP (Identity Provider) by using the leaked certificate to sign tokens or establish secure connections. That can lead to manipulation of the communication between the parties involved and unauthorized access to sensitive resources or user data: user credentials, personal information.

With access to the private key, attackers can create forged tokens that appear valid to the client applications and resource servers. This can grant them unauthorized access to protected resources or enable them to perform actions on behalf of legitimate users.

About the OpenID Connect Flows

Let us discuss two standard flows: the Authorization Code Flow and the Client Credential flow.
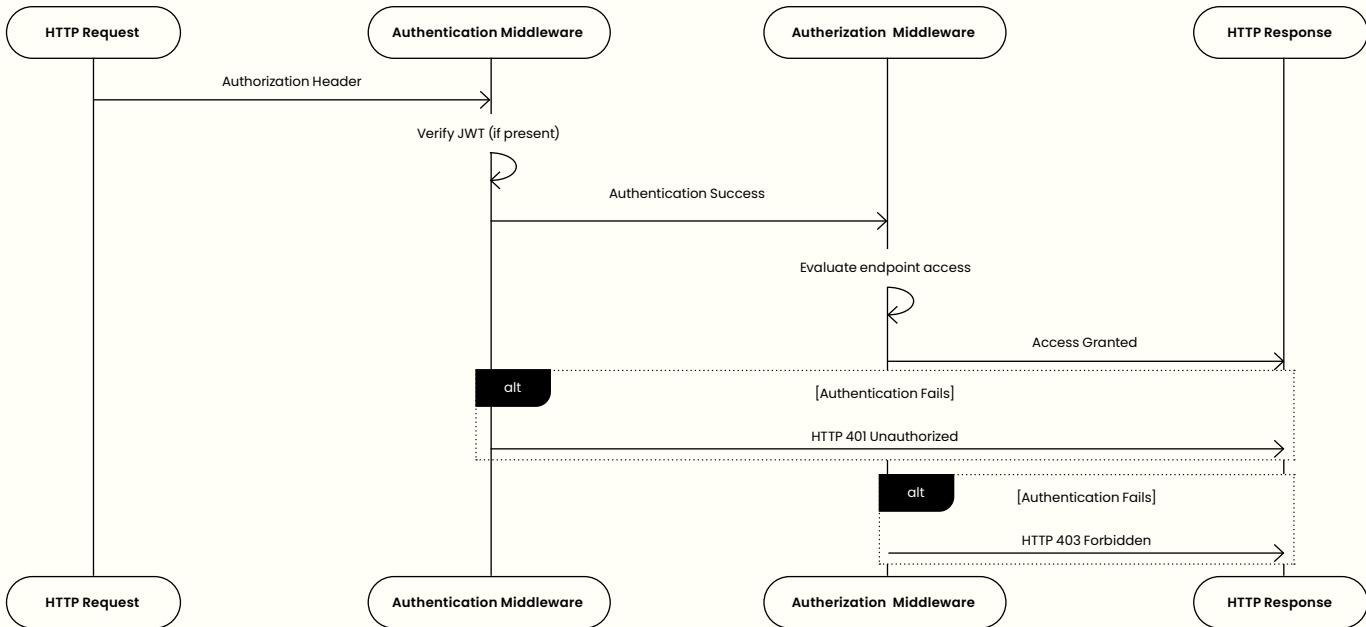
The Authorization Code Flow will require user input to obtain the access token. To get an access token between two services without user input you can use the Credential flow. Those steps do not involve the resource server you build. The resource server can ask the OpenID Connect for additional information for validating the Authorization header. The OpenID Connect Idp has created the JWT with its private key. To validate that JWT, the Idp provides access to the corresponding public key.

OpenID Configuration Discovery

OpenID Connect supports discovery of the needed endpoints used for all the needed steps. We are interested in two endpoints. One is for listing all the endpoint configurations and one is for validating the signature of the access token:

- The `/.well-known/openid-configuration` endpoint, which returns a JSON document containing metadata about the provider's configuration, such as the authorization and token endpoints, supported grant types, and public keys for validating tokens. One of the endpoints listed in this configuration is the location of the /jwks endpoint.
- The `/jwks` endpoint, which returns a JSON document containing the provider's public keys in JSON Web Key (JWK) format, which can be used to validate the signature of the access tokens.

By mocking out these endpoints, we can simulate the behavior of a real OpenID Connect provider without having to set up and configure one.

Authentication and Authorization

When an HTTP request arrives with an authorization header, the authentication process checks the validity of the header, typically by verifying a JSON Web Token (JWT) if present. Once the authentication succeeds, the authorization process begins to evaluate whether the request is allowed to access a specific endpoint. If the authentication check fails, the response has HTTP status code 401 (Unauthorized). If the authorization check fails, the server responds with an HTTP status code 403 (Forbidden).
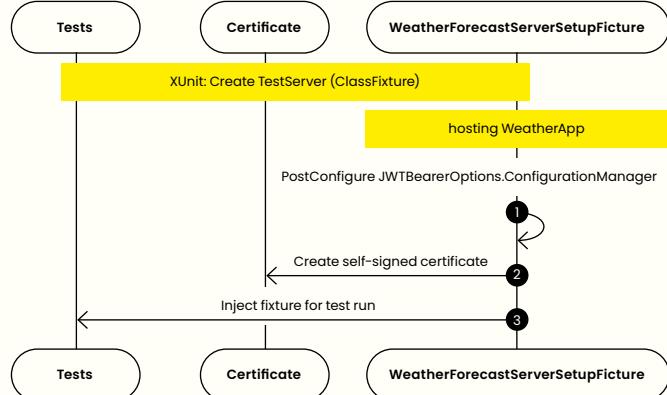
How are the tests setup?

Before some code is shown, let us introduce the classes, the responsibilities of those classes and the interaction between them. This should help to understand the code shown in the section below called Boilerplate.
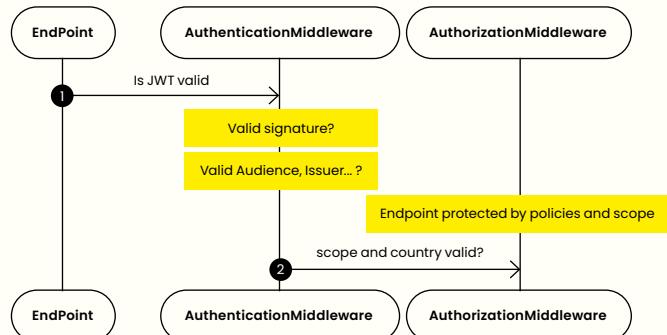
First, we define a couple of tests which are the requirements to help us ensure OpenID Connect Configuration and middleware are functioning together. They need to be simple, show intent and be explicit about it. All those tests follow the Arrange-Act-Assert (AAA)-syntax. To ensure the tests have access to a customizable JWT, a self-signed certificate will be created. That certificate will give those tests the possibility to:

• sign the JWT using the certificate private key.
• provide resource server access to the certificate public key using the /jwkset endpoint.
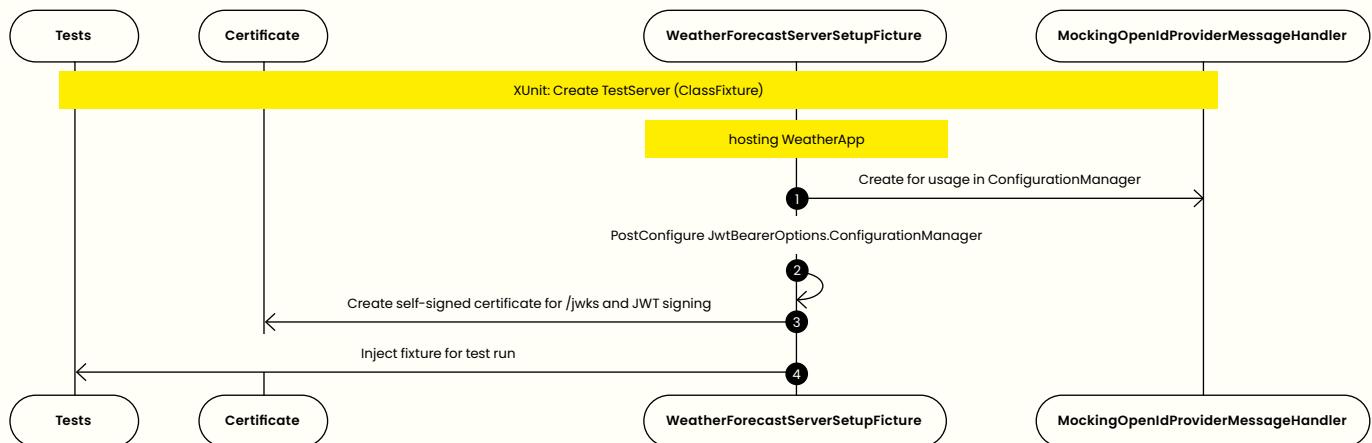
The tests-class implement IClassfixture<Weather-ForecastServerSetupFixture>. The classfixture represents the test server that will be created by XUnit. That will host what is defined in the WeatherApp application, i.e., what is defined in Program.cs
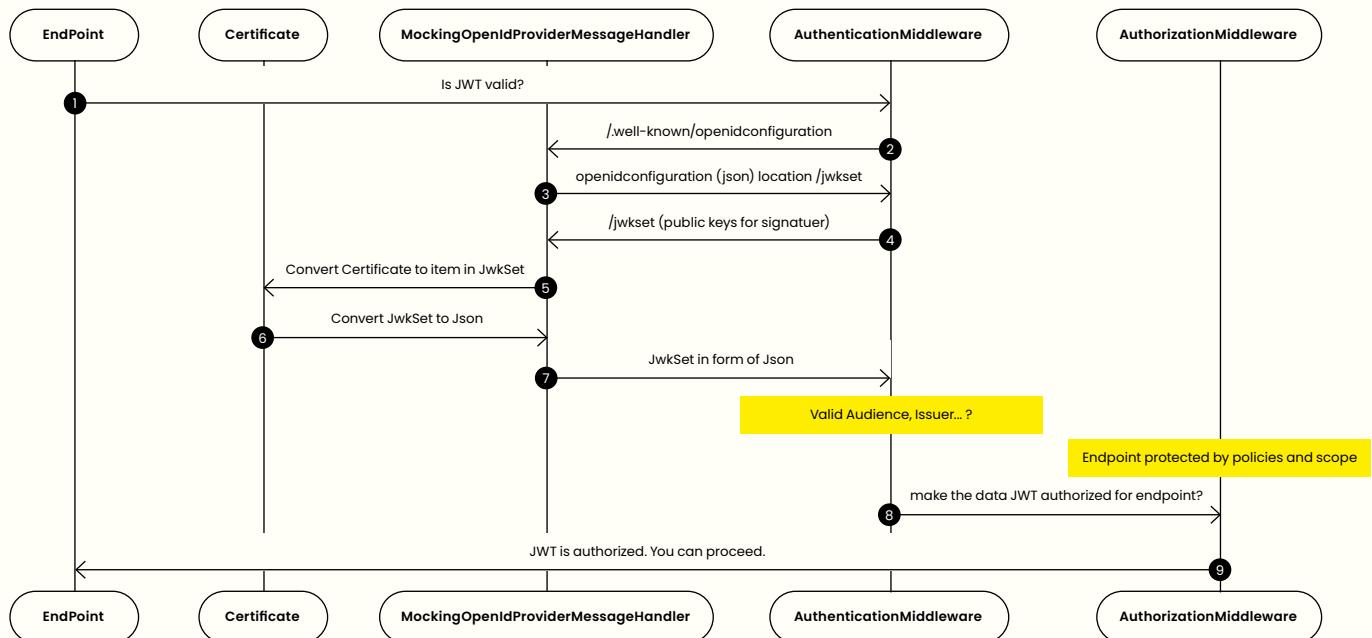
The application that is under test is a sample application called WeatherApp. That application has one endpoint defined called /weatherforecast. The authentication middleware validates there is a JWT. The JWT should contain a valid audience, issuer, and signature. After the token is validated, the authorization middleware kicks in. The JWT is checked to see if it contains data that authorized the request to access the endpoint. The endpoint is protected by two policies. To have a successful HTTP GET Operation, there needs to be a claim with the name country and value Belgium and a scope that should contain "Weather-Forecast:Get".
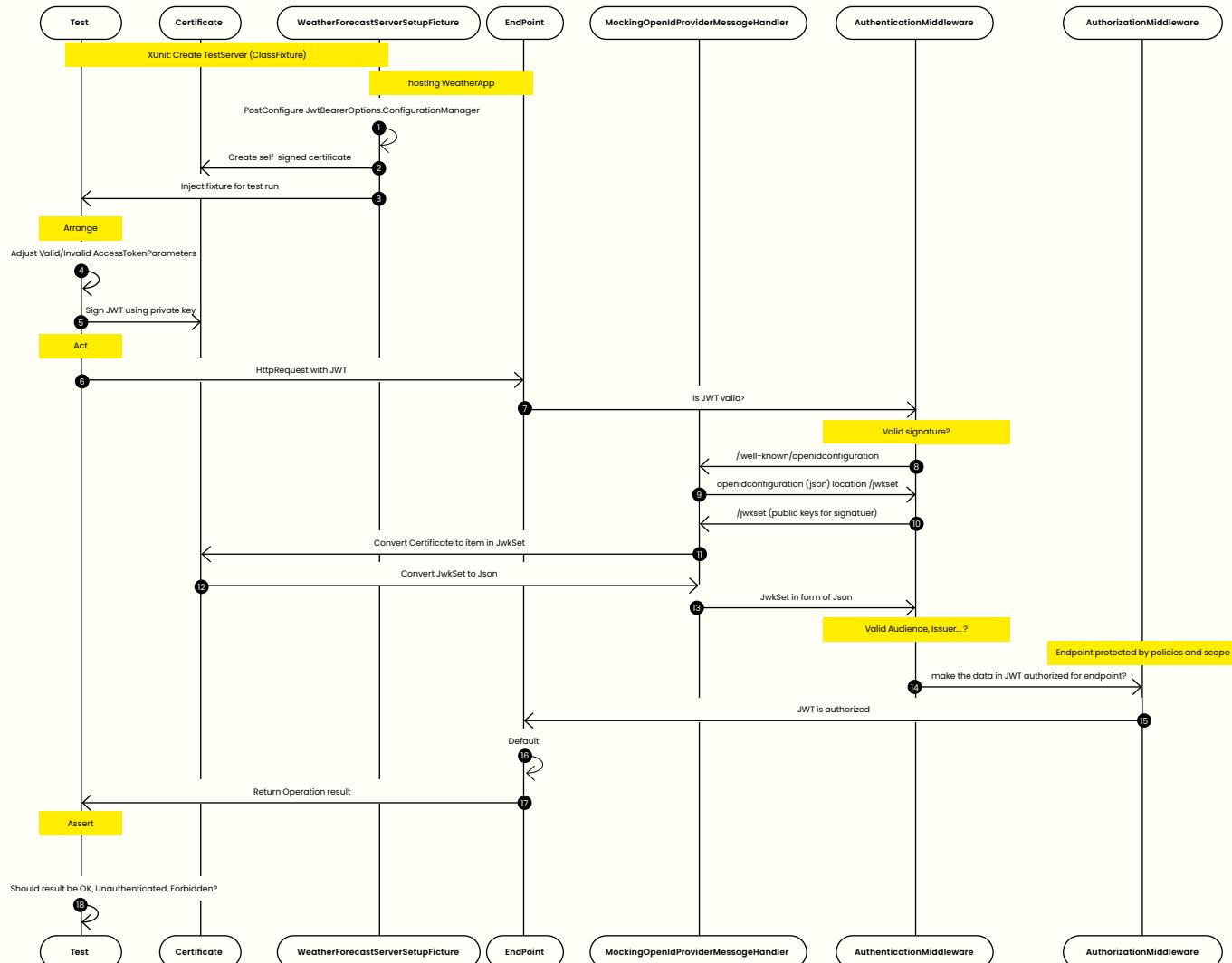
In the class fixture we will set the `JwtBearerOptions.ConfigurationManager`. That happens by using the `ServiceCollection.PostConfigure<JwtBearerOptions>` method. By post-configuring the options, the instantiation of the ConfigurationManager is influenced. A HttpClient that relies on a custom class `MockingOpenIdProviderMessageHandler` is injected.

It is the responsibility of the class `MockingOpenIdProviderMessageHandler` to intercept the retrieval of the OpenID Connect configuration. By using a custom OpenID Connect Configuration object, the location of the `/jwks` endpoint is manipulated. The call made to the `/jwks` endpoint is intercepted. When the location of the public keys' endpoint is called, the message-handler replies with our own JwkSet, based on our self-signed certificate. That certificate is also used for the signing of the JWT. Because it is using the private key of the certificate that has signed the JWT, the certificate public key will generate a valid signature.

The following image is the complete sequence diagram of the above text.
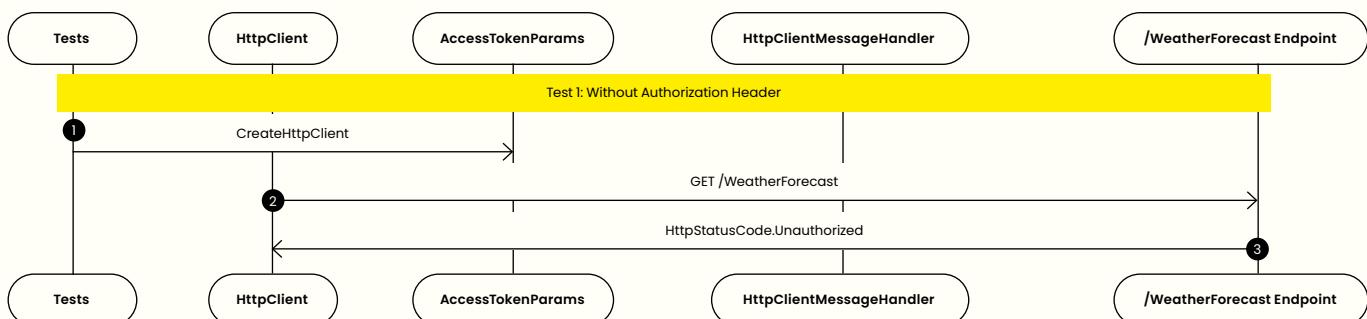
Setting up tests

Next, we will describe a series of tests to validate the behavior of an application's `/WeatherForecast` endpoint when handling different authentication scenarios.

The tests cover cases with no authorization header, a valid JSON Web Token (JWT), an invalid issuer and an invalid claim for the country. Each test has a diagram that illustrates the flow of interactions between the components involved in each test, providing a clear understanding of the expected outcomes.

A test without an authorization header should return with a http status code Unauthorized

The first test is the easiest one. A default http client is created. A request is made to the `/WeatherForecast` endpoint. The endpoint should respond with unauthorized.

```
var httpClient = _fixture.CreateDefaultClient();
var response = await httpClient.GetAsync("WeatherForecast");
response.StatusCode.ShouldBe(HttpStatusCode.Unauthorized);
```
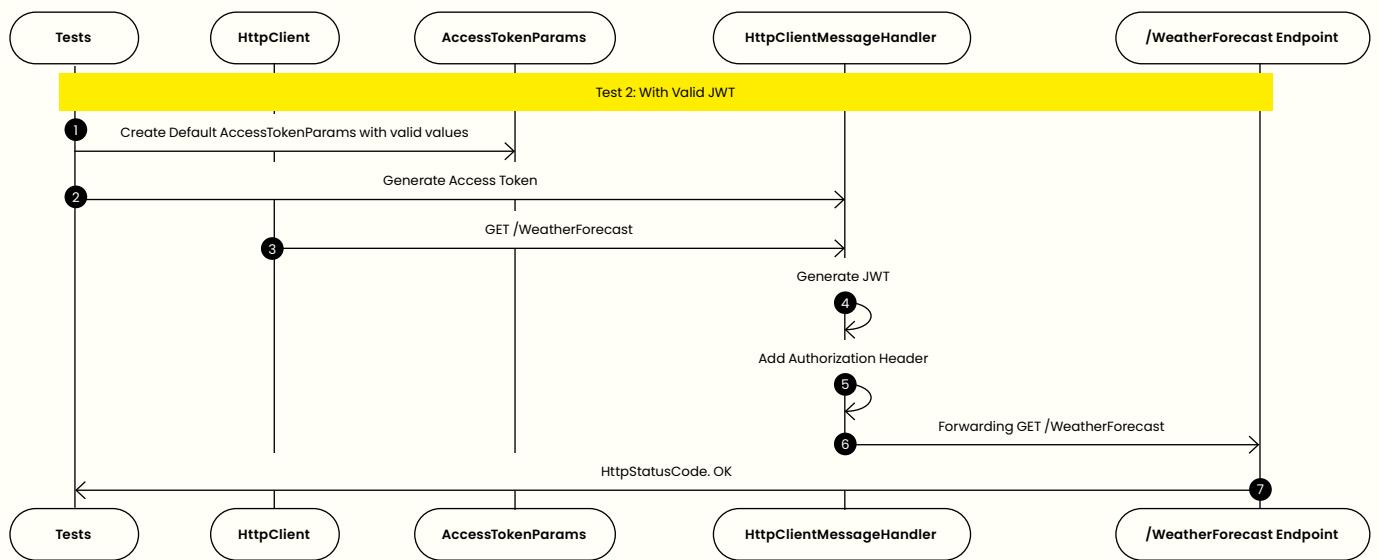
A test with a valid JWT in the authorization header should return with a http status code OK

The code below has the same structure as the previous test, but with a difference: a default instance of the `AccessToken-Parameters`. The default instance of `AccessTokenParameters` contains all the valid information needed to generate a valid JWT. That instance is passed to the `JwtBearerCustomAccessTokenHandler`. `JwtBearerCustomAccessTokenHandler` will generate the access token and add the Authorization header with the access token to the request.
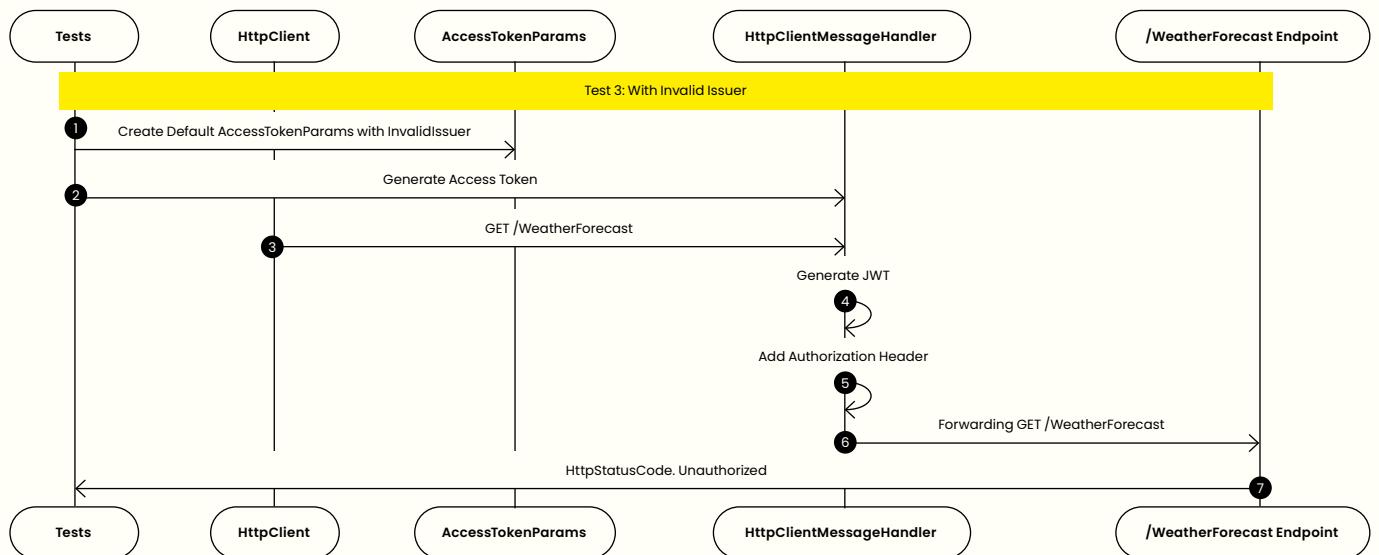
```
var accessTokenParameters = new AccessTokenParameters();
var httpClient = _fixture.CreateDefaultClient(new JwtBearerCustomAccessTokenHandler(accessTokenParameters));
var response = await httpClient.GetAsync($"/WeatherForecast/");
response.StatusCode.ShouldBe(HttpStatusCode.OK);
```

In the diagram below, you will notice the term `HttpClientMessageHandler`. That is a generalized name. This is the class `JwtBearerCustomAccessTokenHandler`. I used that term to stress that it is the `HttpClient` that uses a delegate that manipulates the `HttpRequest` before sending it.

A test with an invalid issuer should return with a http status code Unauthorized

The code below has the same structure as the valid test. A default instance of the `AccessTokenParameters` with an invalid issuer is passed to the `JwtBearerCustomAccessTokenHandler`.

```
var accessTokenParameters = new AccessTokenParameters()
  { Issuer = "InvalidIssuer" };
var httpClient = _fixture.CreateDefaultClient(new JwtBearerCustomAccessTokenHandler(accessTokenParameters));
var response = await httpClient.GetAsync($"/WeatherForecast/");
response.StatusCode.ShouldBe(HttpStatusCode.Unauthorized);
```
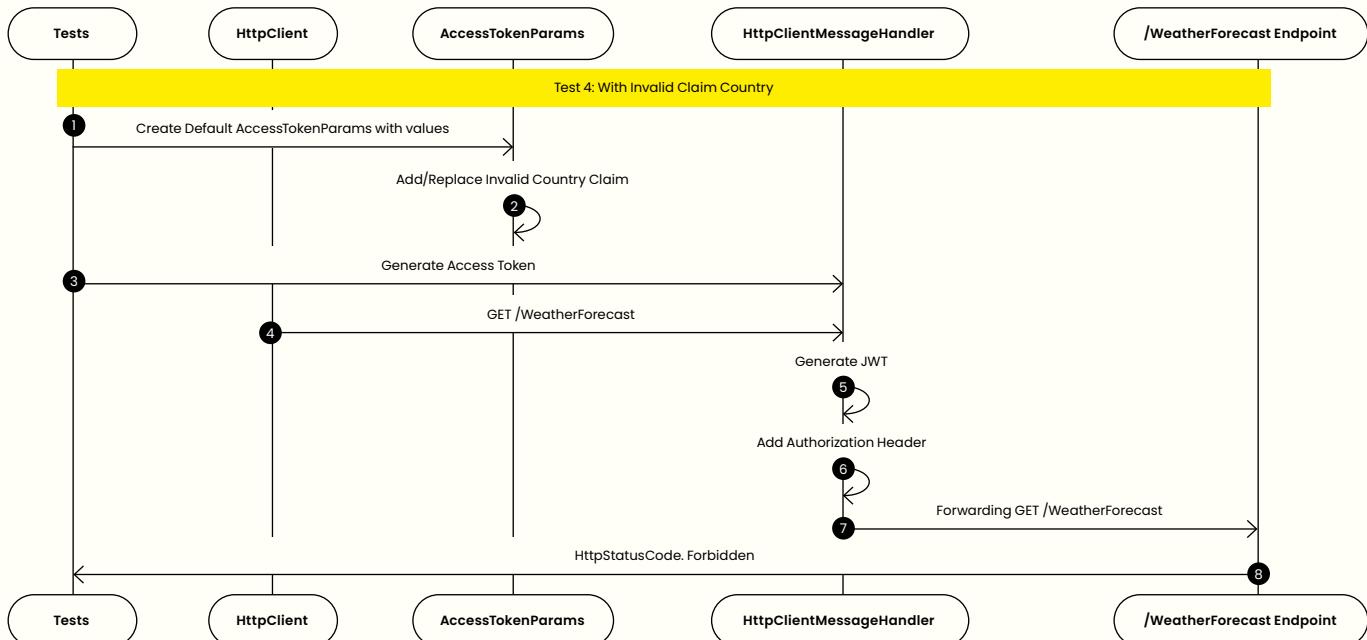
A test with an invalid claim country should return with a http status code Forbidden
The same structure applies to this test as well. The difference now is that a method is used to replace the valid value of the claim "country" with an invalid value.

```
var accessTokenParameters = new AccessTokenParameters();
accessTokenParameters.AddOrReplaceClaim("country", "invalidCountry");
var httpClient = _fixture.CreateDefaultClient(new JwtBearerCustomAccessTokenHandler
(accessTokenParameters, _testOutputHelper));
var response = await httpClient.GetAsync($"/WeatherForecast/");
response.StatusCode.ShouldBe(HttpStatusCode.Forbidden);
```


Setting up the Web Application
To secure `WeatherForecastController`, the authorize attribute is added at the class-level. That will tell the authentication middleware to protect all the endpoints inside that controller. Additionally, the authentication middleware needs to validate the JWT bearer when it detects an Authorization header according to the specified settings. There is a policy defined in the authorize attribute. The specified policies need to be added when you configure the authorization middleware.

WeatherForecastController
To protect the endpoints of the `WeatherForecast-Controller`, define two policies:
- all the controllers' endpoints can only be accessed from Belgium.

- the access token needs to define specific access to the GET /weatherforecast operation.

```
[Authorize(Policy = "OnlyBelgiumPolicy")]
public class WeatherForecastController :
ControllerBase {

[HttpGet()]
[Authorize(Policy = "WeatherForecast:Get")]
public WeatherForecast Get() {
```

Program
This section covers the creation of the web application, the configuration of the authentication and authorization middleware, and the ordering of their execution.

1. Create the builder for the WebApplication `var builder = WebApplication.CreateBuilder (args);`

2. Add the authentication middleware and configure with the help of the `JwtBearerDefaults` class. On the authentication middleware, configure the middleware so that it supports validation for JWT bearer tokens. In the section "`.AddJwtBearer`" there are a lot of options. For this article's purposes, we limit the options to what the middleware should do with claims and what of the JWT should be validated.

```
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme =
    JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme = JwtBearerDefaults.
    AuthenticationScheme;

}).AddJwtBearer(o =>
{
    o.MapInboundClaims = false;
    o.TokenValidationParameters = new Token-
    ValidationParameters
    {
        ValidIssuer = builder.Configuration
        ["Jwt:Issuer"],
        ValidAudience = builder.Configuration
        ["Jwt:Audience"],
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        NameClaimType = "sub",
    }; });
```

3. The authorize attributes in the controller class refers to certain policies. Those policies need to be configured on the authorization middleware. The policies `Only-BelgiumPolicy` and `WeatherForecast:Get` are defined by searching the JWT for the respective claims `country` and `scope`. They should have the value `Belgium` and `weather-forecast:read`.

```
builder.Services.AddAuthorization(authorizationOp-
tions => {
  authorizationOptions.AddPolicy("OnlyBelgium-
  Policy", policy => policy.RequireClaim
  ("country", "Belgium"));
  authorizationOptions.AddPolicy("WeatherForecast:-
  Get", policy => policy.RequireClaim
  ("scope", "weatherforecast:read")); });
```

4. After the middleware is defined, it is time to build the WebApplication. From there you tell the web application to use the configured middleware. The order is important: the sequence of the `Use***`-methods is the sequence that the middleware will be triggered. Having the authorization middleware before the authentication middleware make no sense.

```
var app = builder.Build();
…
app.UseRouting()
   .UseAuthentication()
   .UseAuthorization()
   .UseEndpoints(endpoints =>
   { endpoints.MapControllers();});
```

Boilerplate
Below you will find a summary of the boilerplate code that is needed.
• Creating a self-signed certificate
  • The conversion of the public key to a JWKSet
• Setting up the classfixture
  • Configure the `ConfigurationManager`
  • Mock endpoints by using a mocked `HttpMessage-Handler` in a preconfigured `HttpClient`.

Self-signed certificate
The identity provider has been configured to sign the JWT using RS256. Below you will find a method to mimic that.

The class `SelfSignedAccessTokenPemCertificateFactory` provides the functionality to create an object of the type `PemCertificate`. The instance will contain the certificate, public key and private key.

The certificate will have the following properties:
• a 2048 bit Key size,
• valid for 10 years,
• Ensure the certificate can be used for code signing (OID 1.3.6.1.5.5.7.3.3)
• a start date defines as the day before today
• bound to the domain i.do.not.exist.

```
using (RSA rsa = RSA.Create()){
rsa.KeySize = 2048;
var request = new CertificateRequest("cn=i.do.not.
exist", rsa, HashAlgorithmName.SHA256,RSASignature-
Padding.Pkcs1);
request.CertificateExtensions.Add(
  new X509BasicConstraintsExtension(true, false, 0,
  true));
```

```csharp
request.CertificateExtensions.Add(
  new X509EnhancedKeyUsageExtension
  (new OidCollection
    {
      new Oid("1.3.6.1.5.5.7.3.1")
    }, false));

var yesterday = new DateTimeOffset(DateTime.UtcNow.
AddDays(-1));
var tenyearslater = new DateTimeOffset
(DateTime.UtcNow.AddDays(3650));
X509Certificate2 cert = request.CreateSelfSigned
(yesterday,tenyearslater));

var certificatePem = PemEncoding.Write
("CERTIFICATE", cert.RawData);

AsymmetricAlgorithm? key = cert.GetRSAPrivateKey();

byte[] pubKeyBytes = key.ExportSubjectPublic-
KeyInfo();
byte[] privKeyBytes = key.ExportPkcs8PrivateKey();
char[] pubKeyPem = PemEncoding.Write("PUBLIC KEY",
pubKeyBytes);
char[] privKeyPem = PemEncoding.Write("PRIVATE KEY",
privKeyBytes);

var pemCertificate = new PemCertificate(
  Certificate: new string(certificatePem),
  PublicKey: new string(pubKeyPem),
  PrivateKey: new string(privKeyPem)
);

return pemCertificate;
```

There are a lot of possibilities to play around with the settings of the self-signed certificate and thus validate your security setup.

Create an object of the type JsonWebKeySet
When there is a certificate in a PEM format, a JWKSet can be created from it. The /jwks endpoint expects it in that format. There is a method on the class PemCertificate called ToJwksCertificate. The property PublicKey of the certificate offers a possibility to export the parameters needed to create a JsonWebKey. That instance is added to the Keys property of the class JsonWebKeySet.

```csharp
var certificate = X509Certificate2.Create-
FromPem(CertInPEMString);
var keyParameters = certificate.PublicKey.
GetRSAPublicKey()?.ExportParameters(false);
var e = Base64UrlEncoder.Encode(keyParameters.Value.
Exponent);
var n = Base64UrlEncoder.Encode(keyParameters.Value.
Modulus);
var dict = new Dictionary<string, string>()
    {
        { "e", e },
        { "kty", "RSA" },
        { "n", n }
    };
var hash = SHA256.Create();
var asciiBytes = ASCII.GetBytes(JsonConvert.
SerializeObject(dict))
var hashBytes = hash.ComputeHash(asciiBytes);
JsonWebKey jsonWebKey = new JsonWebKey()
    {
        Kid = Base64UrlEncoder.Encode(hashBytes),
        Kty = "RSA",
        E = e,
        N = n
    };
JsonWebKeySet jsonWebKeySet = new JsonWebKeySet();
jsonWebKeySet.Keys.Add(jsonWebKey);

return jsonWebKeySet;
```

Setting the classfixture
To create system tests, we need to setup the server that gives us our endpoint /WeatherForecast. That happens in a class fixture named WeatherForecastServerSetupFixture.

```csharp
public class WeatherForecastServerSetupFixture:
WebApplicationFactory<Program>
```
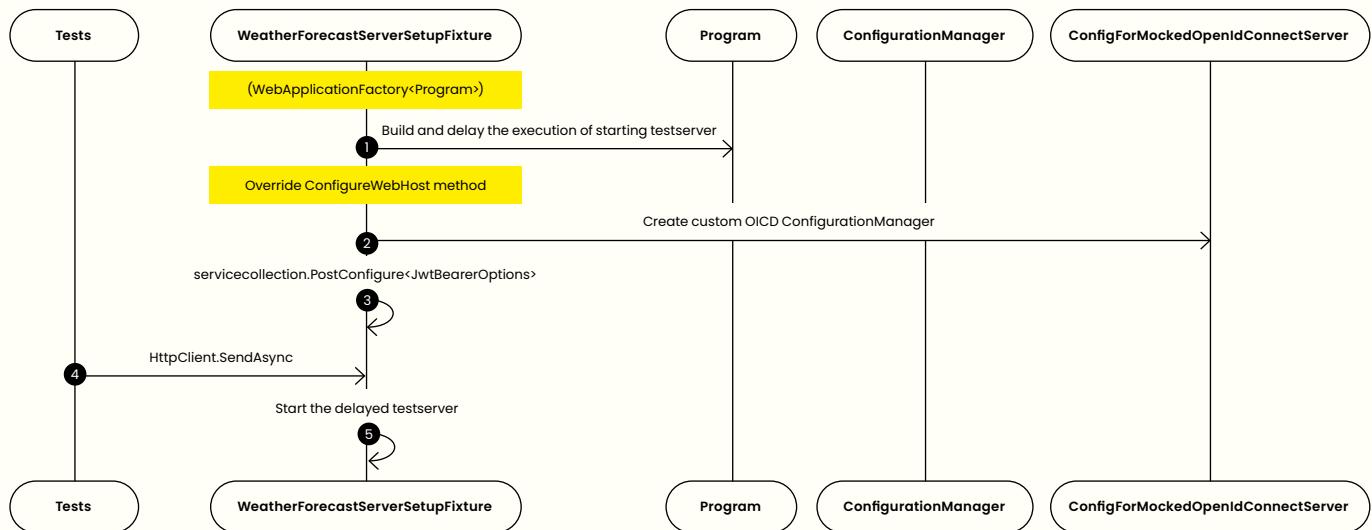
The WebApplicationFactory will use the class Program to build and delay the start of the test server, but only after adding, applying and/or overriding settings and services that have been set in the Program. The TestServer will start when it is needed, in this case when the HttpClient will send a message.

The WebApplicationFactory class offers several methods that you can override. Override the method Configure-WebHost in the class fixture WeatherForecastServerSetup-Fixture. The ConfigurationManager is a property on the OpenID Connect settings.

In the method `ConfigureWebHost`, we post-configure `JwtBearerOptions` with a predefined `ConfigurationManager` made by `ConfigForMockedOpenIdConnectServer`.

```
builder.ConfigureTestServices(services =>  {
    services.PostConfigure<JwtBearerOptions>(
      JwtBearerDefaults.AuthenticationScheme,
      options => {
        options.ConfigurationManager = ConfigForMockedOpenIdConnectServer.Create();
```

The `ConfigurationManager` is built with a preconfigured `HttpClient` in the class `ConfigForMockedOpenIdConnectServer`. The interception of the request will happen in the class `MockingOpenIdProviderMessageHandler`.

MockingOpenIdProviderMessageHandler

The constructor of the class `MockingOpenIdProviderMessageHandler` has two parameters. Two constants are used:
- `Consts.ValidSigningCertificate`: contains the certificate to generate the public key when a request is sent to the /jwks endpoint.
- `Consts.ValidOpenIdConnectDiscoveryDocumentConfiguration`: contains OpenID Connect Settings with predefined values.

The requests the authentication middleware makes will be handled by `MockingOpenIdProviderMessageHandler`.
By overriding the `SendAsync` method, the OpenID Connect settings and public keys are returned when requested.

```
if
(request.RequestUri.AbsoluteUri.Contains(Consts.WellKnownOpenIdConfiguration))return await
GetOpenIdConfigurationHttpResponseMessage();

if
(request.RequestUri.AbsoluteUri.Equals(_openId-ConnectDiscoveryDocumentConfiguration.JwksUri))return await
GetJwksHttpResonseMessage();
```

The response of the OpenID Connect Discovery request will contain settings copied from the real Idp provider you use, with some minor changes, e.g., the location of the /`jwks` endpoint. The /`jwks` endpoint contains generated public keys from our self-signed certificate. Later that same self-signed certificate will be used to generate signatures of the JWT.

IConfigurationManager<OpenIdConnectConfiguration>
All the building blocks are now in place to create the
ConfigurationManager<OpenIdConnectConfiguration>.

The following parameters are required to create the
instance:
- Consts.WellKnownOpenIdConfiguration: a valid URL of
  the fake Idp Provider
- OpenIdConnectConfigurationRetriever: retrieves the
  OpenID Connect config
- HttpDocumentRetriever: the instance that the OpenId-
  ConnectConfigurationRetriever uses to fetch the config.
  It uses an instance of a HttpClient configured with the
  MockingOpenIdProviderMessageHandler.

The above translate to the code below.

```
var handler = new MockingOpenIdProviderMessage-
Handler(Consts.ValidOpenIdConnectDiscovery-
DocumentConfiguration, Consts.ValidSigning-
Certificate);
var openIdHttpClient = new HttpClient(handler);
var httpDocumentRetriever = new HttpDocument-
Retriever(openIdHttpClient);

var openIdConnectConfigRetriever = new OpenId-
ConnectConfigurationRetriever();

return new ConfigurationManager<OpenIdConnect-
Configuration>(
Consts.WellKnownOpenIdConfiguration,
openIdConnectConfigRetriever,
httpDocumentRetriever);
```

Generating an access token
The code below shows how to generate a JWT using a
X509Certificate2.

In this code snippet, we demonstrate how to create an
encoded access token using a valid signing certificate.
First, we convert the certificate to an X509Certificate2 object.
Next, we create signing credentials using the certificate
and the RSA-SHA256 algorithm. We then define a Claims-
Identity object and set up a SecurityTokenDescriptor
with the necessary information such as the audience, issuer,
expiration time, and signing credentials. Finally, we use the
JwtSecurityTokenHandler to create and write the token,
resulting in an encoded access token.

```
var cert = Consts.ValidSigningCertificate.
ToX509Certificate2()
var signingCredentials = new SigningCredentials
(new X509SecurityKey(cert), SecurityAlgorithms.
RsaSha256);
var identity = new ClaimsIdentity(Consts.Claims);
var securityTokenDescriptor = new SecurityToken-
Descriptor {
  Audience = Consts.Audience,
  Issuer = Consts.Issuer,
  NotBefore = DateTime.UtcNow,
  Expires = DateTime.UtcNow.AddHours(1),
  SigningCredentials = signingCredentials,
  Subject = identity };
var handler = new JwtSecurityTokenHandler();
var securityToken = handler.CreateToken
(securityTokenDescriptor);
var encodedAccessToken = handler.WriteToken
(securityToken);
```

When generating the /jwks endpoint, the Consts.Valid-
SigningCertificate was used. The authentication
middleware requires a valid signature. The PemCertificate
object that contains the Certificate, PrivateKey and
PublicKey properties is converted into a X509Certificate2.

```
X509Certificate2.CreateFromPem(Certificate,
PrivateKey);
```

Setting invalid audience, issuer and subject or having a
wrong certificate should be used in tests to ensure an
unauthenticated or unauthorized error is the expected result.

Adding the access token to the request
There are multiple ways to add an access token to a request.
The WebApplicationFactory class offers a CreateDefault-
Client method, which creates a HTTP Client for you. What
that method does is create a HTTP client. It sets the URL of
the test server. When using that method, there is an optional
parameter where you can pass a DelegatingHandler.
JwtBearerCustomAccessTokenHandler is a custom class
that extends from that DelagatingHandler and overrides
the Send method it offers.

```
var encodedAccessToken = JwtBearerAccessToken-
Factory.Create(_accessTokenParameters);
request.Headers.Authorization = new Authentication-
HeaderValue("Bearer", encodedAccessToken);
return base.Send(request, cancellationToken);
```

When the HttpClient sends the request, `JwtBearer-CustomAccessTokenHandler` will execute the Send method and thus add the access token to the `AuthorizationHeader`.

The `JwtBearerAccessTokenFactory.Create` will create the access token.

Summary

This article touches on various concepts and techniques, such as creating self-signed certificates, generating JWK sets on the fly and managing access tokens. I encourage you try out these snippets or visit GitHub for the entire codebase. It can be a step towards gaining a deeper understanding of OpenID Connect and its integration into applications. That alone will help improve application security and performance.

By having another tool in your toolbox, you can create a robust testing environment. That allows you to test different scenarios in an automated manner. Those tests will include testing of valid and invalid access tokens. It ensures that applications handle authentication and authorization correctly. All with realistic configuration and without mocked classes.

With packages as a big black box, it is not always easy to understand what is happening behind the scenes. You should now have a better understanding on setting up a test server, which can be useful in other projects or testing scenarios.
</>

Sources

- Create a Controller-based API: WeatherForecast sample: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-6.0
- Insight in OpenID Connect endpoints: https://openid.net/specs/openid-connect-discovery-1_0.html
- More information about access tokens: https://oauth.net/specs/
- JWK / JWT / Certificate relationship: https://dirkbolte.medium.com/the-token-connection-61e22ff54fe0
- JWK: https://openid.net/specs/draft-jones-json-web-key-03.html
- JWT:
  a. Inspecting JWT:
     i.   https://www.jwt.io
  b. how to create them:
     i.   https://dotnetcoretutorials.com/2020/01/15/creating-and-validating-jwt-tokens-in-asp-net-core/
     ii. https://www.iversis.com.au/post/generate-json-web-token-from-a-pkcs-12-x509certificate
- Information about self-signed certificates:
  a. https://dotnetcoretutorials.com/2020/11/18/generating-self-signed-certificates-for-unit-testing-in-c/
  b. https://oidref.com/1.3.6.1.5.5.7.3.3
  c. Generating a self-signed certificate using Dotnet 6 standard libraries: https://chat.openai.com/chat
- How to setup your integrationtests:
  a. https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-6.0

READ MORE ONLINE

Kristof Riebbels

Extending Entity Framework Core

Entity Framework Core offers a broad framework to help create your database schema, and then store and access your data in said database. But what if you want to do more than what Entity Framework Core offers out of the box? What if you have an edge case that needs support? Extending Entity Framework Core might be a solution.

Author Victor de Baare

What is Entity Framework Core?

Entity Framework Core (EF core) is a Microsoft supported open-source object-relational mapping (ORM) framework. As the name implies it helps developers with persisting and accessing data which resides in a database without being encumbered with converting it from the database Data-Set's to .NET objects. Additionally, it also helps with creating a database schema based on the .NET objects, or the other way around, creating .NET objects based on your database schema. In summary, it eliminates most of the data-access and data-persisting code and frees up the developer's time to focus on more important areas of the code.

We will now focus on the persistence of data in your database, particularly a SQL database. When you decide to save data to your database, EF core will do this by using an insert, update or delete SQL statement. Often this is a "good enough" approach, but in some situations you might want to use a merge statement instead. For example, on every insert, update or delete statement a round-trip to the database is performed. With a few records this won't impose any issue. When the amount of records increases the numerous round-trips will slow down the application. To avoid the performance degration a merge statement could be used. A merge statemant would create a single round-trip to the database for every type of object which needs to be inserted, updated or deleted regardless of the amount of records.

To avoid going back to writing the code yourself for every object, you can write an extension on EF core. This way you would only need to write the code once and afterwards you'll let EF core do the heavy lifting. This article will dive deeper into how you can extend EF core. In the next article we will dive deeper into using merge statements. To do this you'll first need to know how a migration in EF core works.

Migrations

EF core uses migrations to update the database Schema based on the data model changes that the developer made. EF core compares the data model changes that the developer made with a snapshot of the old model that is known to EF core. Based on the snapshot, a migration file is generated to describe the operations which are needed to move from the old model towards the new model. The individual migration files are saved to a history table in the database. This way EF core can track which migrations have been applied and which have not, which is important if you want to update your database and have multiple migrations to apply.

Dotnet EF and Migration Initial → Design Time Generation → Migration.cs → Run Time Generation → Sql script

The migration takes place in two steps: the creation of a migration file based on your changes and updating your database schema. To create a migration file you can use the .NET EF core console commands. This will result in calling the CSharp part of the migration, the CSharpMigration-Generator. We will discuss CSharpMigrationGenerator later in the article. The CSharpMigrationGenerator will generate migration operations based on the changes that have been made. The changes will be persisted in a migration.cs file in which you can add extra operations or extra custom SQL. Updating the database is often performed during startup of your application. We will first discuss how we can extend the SQL script generation of EF core.

Extending the MigrationBuilder API.
The MigrationBuilder builds the migration and contains many different operations. But it cannot contain everything a developer might want to do. To accommodate the option of creating your own operations, the MigrationBuilder can be extended! You can use the '''sql()''' method to write your own little piece of code into a migration. The '''sql()''' method can be useful if a developer wants to update some records before or after a migration is performed. Another possibility is writing your own custom migration operations to extend the migrations.

If you decide to create your own custom migrations operations, the first step is declaring your own Custom-MigrationSqlGenerator and making it inherit from the SqlServerMigrationSqlGenerator.

```
public class CustomMigrationSqlGenerator:
SqlServerMigrationsSqlGenerator
{
  public CustomMigrationSqlGenerator(
    MigrationsSqlGeneratorDependencies dependencies,
    ICommandBatchPreparer commandBatchPreparer)
    : base(dependencies, commandBatchPreparer)
  {
  }

  protected override void Generate(
    MigrationOperation operation,
    IModel model,
    MigrationCommandListBuilder builder)
  {
```

```
    case CreateMergeOperation createoperation:
      builder.AppendLine("---Write your SQL code here");
      builder.EndCommand();
      break;

    case DropMergeOperation dropoperation:
      builder.AppendLine("---Write your SQL code here");
      builder.EndCommand();
      break;

    default:
      base.Generate(operation, model, builder);
      break;
  }
}
```

The next step is registering your custom implementation of the SQL generator in your EF core DbContext. After registering your custom implementation and running a migration, your custom generator will be used instead of the default generator for generating your migration code.

```
protected override void OnConfiguring(DbContext-
OptionsBuilder options)
  => options
  .UseSqlServer(_connectionString)
  .ReplaceService<IMigrationsSqlGenerator,
  MyMigrationsSqlGenerator>();
```

Now, all the necessary steps are done to implement your own custom operations. For example, if a merge operation needs to be created in the database you can create a CreateMergeOperation and add this to the Custom-MigrationSqlGenerator.Generate method. Then you can add the code to an existing migration so the operation is added to the SQL script when the migration is executed.

```
///Generated Migration.cs
protected override void Up(MigrationBuilder
migrationBuilder)
{
migrationBuilder.CreateMerge('TableName',
(columnsBuilder) => new {...});
}
```

```
///CustomMigrationBuilderExtensions.cs
public static OperationBuilder<CreateMergeOperation>
CreateMerge(
  this MigrationBuilder migrationBuilder,
  string tableName,
  Func<ColumnsBuilder, TColumns> columns)
{
  var operation = new CreateMergeOperation
  (tableName, columns);
  migrationBuilder.Operations.Add(operation);

  return new OperationBuilder<CreateMergeOperation>
  (operation);
}
```

The implementation still requires the developer to add custom code after performing the first part of the migration. Otherwise the migration wouldn't know another operation should be executed. A better implementation would be to add annotations to the entities you want to get custom operations on. During the generation of the operations you can check if an entity has a specific annotation. If the annotation is present you can execute the custom code.

With the SQL generator approach it still only works in the second part of the migrations: the SQL script generation. The preference would be to automatically add custom operations based on annotations to the first part of the migration. That way you would have nothing to do besides adding an annotation in the configuration of the model. Afterwards you would be able to see the new operations in the migration class that is generated in the first step.

To achieve this you would need to extend more services that are used for the migration. The MigrationsModelDiffer.cs and the CSharpMigrationOperationGenerator.cs are the two services that you would need to extend. We will discuss how to extend these services in the following paragraphs.

MigrationsModelDiffer

As the name implies this part of the migration will check for any differences between the new model and the old model. The beautiful part is you can actually include the check on potential custom annotations for the differences, if you want to include custom code when an annotation is present. You can also use this diff method to remove the custom code when the annotation is no longer present.

For implementation you would have to create a class inheriting from the MigrationsModelDiffer.cs class and then override the following method:

```
/// CustomMigrationsModelDiffer.cs
public override IReadOnlyList<MigrationOperation>
GetDifferences(IRelationalModel? source,
IRelationalModel? target)
```

The method is called by the migration to get the differences between the target and source entities. By overriding this we can check the target and source for the annotation and if changes are present we can instruct it to create an operation to update the database schema with the changes that we would like to see. The operations that are created here will be passed towards the next step in the migration; the CSharpMigrationOperationGenerator.

An example of the implementation for the custom operations is as follows:

```
public override IReadOnlyList<MigrationOperation>
GetDifferences(IRelationalModel? source,
IRelationalModel? target)
  {
        var sourceTypes = GetEntityTypesContaining-
        MergeAnnotation(source);
        var targetTypes = GetEntityTypesContaining-
        MergeAnnotation(target);

        var diffContext = new DiffContext();
        var customOperations = DiffCollection
        (source, target, diffContext, Diff, Add,
        Remove, (x, y, diff) => x.Name.Equals(y.Name,
        StringComparison.CurrentCultureIgnoreCase));
        return base.GetDifferences(source, target)
        .Concat(customOperations).ToList();
  }
```

The DiffCollections is an internal API method that helps make sure an operation is only created once for the target and source combination. The add and remove parameters in the DiffCollection are the functions which contain the knowledge on how to create the custom operations.

The last step that we need to do is to register the ModelDiffer in the service collection. This service must be registered with the DbContext as well. From here we can extend the previous code to include the model differ service.

```
protected override void OnConfiguring
(DbContextOptionsBuilder options)
    => options
    .UseSqlServer(_connectionString)
    .ReplaceService<IMigrationsModelDiffer,
    CustomMigrationsModelDiffer>()
    .ReplaceService<IMigrationsSqlGenerator,
    MyMigrationsSqlGenerator>();
```

CSharp Migration Operation Generator

The CSharpMigrationOperationGenerator will provide the .NET code which is parsed in the migration class. With the new operations included, the Csharp generator can now be extended to know what to do when such an operation is encountered. An important part here to remember is that the migrations are done in two parts which both are run at different times. For example the migration of the Csharp code is run during design time. The SQL migration is often performed during run time.

Victor de Baare

For the SQL migration this means we can use the DbContext OnConfiguration method to register the needed overrides to create the correct SQL script. The CsharpMigration-OperationGenerator, which runs during design time, is implemented differently. To make sure that your customer CsharpMigrationOperationGenerator can be called, adjust the imported Microsoft.EntityFrameworkCore.Design NuGet package.

When you import this package it automatically ensures that certain design time services are not exposed for your code. The CSharpMigrationOperationGenerator is part of the services which are filtered away. To actually be able to use all the services you should adjust the code in your .csproj file a bit.

For example when you import the package you get the following:

```xml
<PackageReference Include="Microsoft.EntityFramework-
Core.Design" Version="7.0.2">
  <PrivateAssets>all</PrivateAssets>
  <IncludeAssets>runtime; build; native; contentfiles;
  analyzers; buildtransitive
  </IncludeAssets>
</PackageReference>
```

Now you will need to adjust this to the following:

```xml
<PackageReference Include="Microsoft.EntityFramework-
Core.Design" Version="7.0.2">
  <PrivateAssets>all</PrivateAssets>
</PackageReference>
```

Once done you can extend the CSharpMigrationOperation-Generator with your own code. Before we start extending the generator, we must make sure the generator can be found during a migration. For the SQL generator we could use the OnConfiguration method of the DbContext. Due to the fact that the CSharp generator is a design time implementation, the SQL generator solution won't work. An alternative for this is using the DesignTimeServices.

For DesignTimeServices, Entity Framework has a separate interface that needs to be included. The interface exposes the ConfigureDesignTimeServices(IServiceCollection) method. EF core scans the start-up project for this interface to register potential design time services. If you want to ship potential extensions in a NuGet package the scanning for an interface might be a problem. The solution to this issue is using an assembly attribute in your startup project which contains the name of the class which implements the interface and the full namespace.

```csharp
[DesignTimeServicesReference("TypeName", "ForProvider")]
```

The typename parameter is the assembly-qualified name to be added to the servicecollection. The 'ForProvider' parameter is the name of the provider for which the DesignTimeServices should be used. This parameter is nullable, when left empty the service is added for all the present providers.

With the interface in place we can start implementing the CSharp generator. The implementation is very similar to that of the SQL generator. We will be overriding the `protected virtual void Generate(MigrationOperation operation, IndentedStringBuilder builder)` method. The MigrationsModelDiffer will be doing the work to provide the custom operations so the CSharp generator should work in providing the CSharp code of the implementation. In this case when we get an operation of the `CreateMerge-Operation` type we want to perform the following additional code:

```csharp
private static void Generate(CreateMergeOperation
operation, IndentedStringBuilder builder)
{
        builder.AppendLine($".CreateMerge(");
        using(builder.Indent()) {
            builder.AppendLine($".....");
    }
}
```

If you generate a new migration, you will see the migration will contain the CreateMerge statement now.

With this the whole chain is completed. We can now just add an annotation to the entities in the modelbuilder method on the DbContext. This will result in the merge statements being generated on the next migration. An example of the source code for generating customer operations can be found here: https://github.com/VictordeBaare/EntityFramework-Extensions.

`</>`

READ MORE ONLINE

Upgrade Your App to the Future: Migrating from WPF/WinForms to Blazor

This article is based on a WPF to Blazor migration project and Microsoft documentation. Desktop applications built with WPF or WinForms have been widely used for many years, allowing developers to create feature-rich desktop applications with complex user interfaces. However, with the increasing demand for cross-platform applications and modern user interfaces, it has become an appealing option to migrate legacy WPF/WinForms applications to modern web technologies, even as Microsoft continues to maintain WPF and WinForms.

Author Niels Nijveldt

Blazor is a relatively new web framework developed by Microsoft that enables developers to build web applications using .NET. With Blazor, you can leverage your existing .NET libraries, frameworks, and skills to create web applications that run in the browser (Blazor WebAssembly) or on a server (Blazor Server) without the need for plugins or JavaScript.

More detailed information about Blazor can be found in the article "Introduction to Blazor" by Mark Foppen in magazine #13.

In this article, you will learn about the potential process of upgrading your legacy WPF/WinForms application to a modern Blazor web application. We will also cover the steps you need to take, the decisions you'll need to make, and the potential pitfalls to avoid.

Why Blazor?

When deciding to upgrade your existing application, you'll need to determine the best approach. You probably will investigate what solution fits best for you. Do you want to start greenfield and start collecting the right specifications from scratch? Time and budget might be limited, then it's hard to make a decision with these options.

In that case, you might want to explore Blazor. Since you already have a lot of business logic in your existing .NET application, you can save time by reusing this logic. Also, having .NET knowledge within your team or organization provides a significant advantage. The learning curve is more gradual than learning an entirely new framework.

Niels Nijveldt

Compared to WinForms or WPF, Blazor offers several advantages:
- Blazor is supported by all major browsers[1] (Safari, Chrome, Edge, FireFox), including mobile
- Deployment/Delivery is easier to manage
- Plenty of widely supported UI frameworks
- Multiple different hosting methods

While both WPF and WinForms still have roadmaps[2][3], Microsoft's product teams are mainly focused on performance and bug fixes, and the capacity of these teams is limited. Blazor, on the other hand, is gaining popularity. We can expect many improvements with almost every ASP.NET Core upgrade, benefiting both application performance and developer productivity.

The preparation

Hopefully, your application's architecture separates view logic from business logic, possibly using MVC or MVVM or a similar architecture. If not, migrating this (business) logic from the old view to the new modern web view will require extra work.

Also, best practices widely used today may not have been common when your WPF/WinForms application was built. For example, dependency injection, NuGet packages, pipelines for deploying the application, and more will require additional time. Take this into account when planning and implementing the solution.

[1]   https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms?view=aspnetcore-7.0
[2]   https://github.com/dotnet/wpf/blob/main/roadmap.md
[3]   https://github.com/dotnet/winforms/blob/main/docs/roadmap.md

Moreover, ensure that your project's .NET versions, NuGet packages, and other dependencies are up-to-date, using at least the .NET Standard or the latest supported .NET version. Outdated libraries may cause problems, especially when using Blazor WebAssembly.

Lastly, consider your generic components. You might have created reusable WPF/WinForms components or used a library with components. For Blazor, you'll want to do something similar.

Numerous open-source or free-to-use component libraries are available within the community. Check if these libraries offer the features you need and explore them early to avoid switching between libraries and wasting time. MudBlazor[4] and Radzen[5] are well-known libraries with a wide variety of user interface components. Both MudBlazor and Radzen are free to use and contain a large list of components that are easy to use in your application. Keep in mind not all (component) libraries are just free to use. Make sure to understand their license model.

Moving from a desktop to the browser
Transitioning from WPF or WinForms to Blazor means moving from a desktop (probably Windows) environment to a browser. This shift may limit some functionality, such as interactions with the host. For example, if your application opens Microsoft Word and controls its usage, you'll lose that control when moving to a browser-based application. You can still enable users to open documents in Word, but you won't be able to control their actions within Word.

If your application relies heavily on the operating systen, consider looking into Blazor Hybrid[6] or .NET MAUI[7] as alternatives.

Making your application accessible in a browser also requires a different deployment approach. Depending on the chosen hosting method, your application should be made available on the internet or an (internal) network. This means you'll need to prepare or adjust your network infrastructure, set up SSL certificates to secure connections, and configure DNS to expose your application on a familiar and safe web address.

Blazor Server vs Blazor WebAssembly
When setting up your new solution, you'll need to choose between Blazor Server and Blazor WebAssembly. While it may not seem crucial, this decision significantly impacts your infrastructure and application reliability. It's a good idea to consider this choice early in your project. You can still switch between the two, but it will require time and effort.

If you're unsure which option is best, you can initially support both and test their reliability.

Understanding the difference between the two hosting models is essential. Blazor WebAssembly runs entirely in the user's browser, while Blazor Server renders HTML server-side, and client interactions are processed through SignalR.

Microsoft provides an overview[8] to help you choose the right hosting model for your needs.

Feature	Blazor Server	Blazor WebAssembly (WASM)	Blazor Hybrid
Complete .NET API compatibility	✓	✗	✓
Direct access to server and network resources	✓	✗†	✗†
Small payload size with fast initial load time	✓	✗	✗
Near native execution speed	✓	✓‡	✓
App code secure and private on the server	✓	✗†	✗†
Run apps offline once downloaded	✗	✓	✓
Static site hosting	✗	✓	✗
Offloads processing to clients	✗	✓	✓
Full access to native client capabilities	✗	✗	✓
Web-based deployment	✓	✓	✗

† Blazor WebAssembly and Blazor Hybrid apps can use server-based APIs to access server/network resources and access private and secure app code.

‡ Blazor WebAssembly only reaches near-native performance with ahead-of-time (AOT) compilation.

For Blazor WebAssembly to work in your environment, you'll need a service (preferably an API) to interact with your internal systems (e.g. databases). In contrast, Blazor Server can act as the single entry point for your environment, interacting with all internal systems.

However, Blazor Server has some drawbacks, such as maintaining the state of interacting users on the server. This can complicate matters, as scaling or rebooting your application could cause users to lose their session.

Additionally, you'll need to ensure no shared state between users to avoid undesirable or harmful situations.

Compared to your WPF/WinForms application, Blazor WebAssembly may be a more suitable choice, as both WPF/WinForms and Blazor WebAssembly run client-side. However, keep in mind that not all .NET libraries are supported in Blazor WebAssembly. It also has a longer initial load time, as all relevant DLLs need to be downloaded first. Moreover, since the application runs client-side you will need a public API to communicate to databases or internal services. Or you need to expose these databases

4 https://blazor.radzen.com
5 https://mudblazor.com
6 https://learn.microsoft.com/en-us/aspnet/core/blazor/hybrid/?view=aspnetcore-7.0
7 https://learn.microsoft.com/en-us/dotnet/maui
8 https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0

and internal services so your application can access it from the client, but you probably don't want this due to security measures.

As a result, your application will behave like a Single Page Application (SPA) with excellent performance. You can host your application as a static app, eliminating concerns about frontend scaling.

With .NET 8 Blazor will get an enhancement called "Blazor United". This will be a combination of Blazor Server and Blazor WebAssembly. This means when the DLL's are downloaded the application will use these and have a quick response time. When the DLL's are not available yet, it will use the Blazor Server capability to make sure you don't have to wait on downloading these DLL's. By using this the application users will get a better/smoother experience. If "Blazor United" is available for you then you should look into this as this offers best of both worlds.

User experience

One important aspect to consider is the change in user experience (UX) for your end-users. With your current application, users are accustomed to a desktop experience, often featuring a Microsoft Windows look and feel. Transitioning to a browser-based application will introduce a different experience. Elements such as URLs, opening tabs, and cookies might be new to some users. As a result, it's essential to provide clear instructions on how to use the application within a browser environment.

Authentication will also be different. With WPF/WinForms you might have username password authentication or maybe Windows Authentication. With Blazor you could use Azure Active Directory B2C to provide a safe solution.

Additionally, this transition presents an opportunity to make significant UX improvements to your application. If applicable, consider utilizing monitoring tools to analyze usage patterns and make data-driven enhancements. By focusing on user experience and addressing potential challenges, you can ensure a smoother transition and a more satisfying experience for your application's users.

Conclusion

In conclusion, migrating your legacy WPF/WinForms application to a modern Blazor web application can be a strategic and beneficial decision. Blazor enables you to leverage your existing .NET knowledge and resources while providing a more future-proof, browser-based solution that supports a wide range of devices and platforms.

The migration process requires careful planning, architectural considerations, and an understanding of the differences between Blazor Server and Blazor WebAssembly hosting models. By taking the time to analyze your current application's architecture, dependencies, and components, you can ensure a smoother transition to Blazor.

Moreover, embracing best practices and modern development techniques will help you create a more maintainable and scalable application in the long run. Keep in mind the trade-offs between Blazor Server and Blazor WebAssembly when choosing a hosting model that best suits your application's requirements and infrastructure.

As you embark on this journey, it's essential to consider the impact of the migration on user experience and address potential challenges. Transitioning from a desktop application to a browser-based solution presents an opportunity to make significant UX improvements. Utilize monitoring tools to analyze usage patterns and make data-driven enhancements to optimize the experience for your application's users.

By focusing on these aspects and providing clear instructions to help users adapt to the new application, you can ensure a smoother transition and a more satisfying experience for your users. Remember, while there may be challenges along the way, the end result will be a more versatile, powerful and maintainable application that embraces modern web technologies.

Will you upgrade your app to the future?
‹/›

READ MORE ONLINE

Identity Access Management in Microsoft 365

Where two worlds meet: Identity Access Management in Microsoft 365 and Azure. A deep dive!

With the increasing importance of cloud services, the way of accessing IT assets has changed. Assets are stored in the cloud and digital identities (IDs) are becoming more used outside the corporate network to access these assets. In corporate networks, one or more firewall systems are used as a perimeter to protect access to IT assets. But what if you are outside the corporate network? In the home office, the firewall in the DSL router provides a basic protection against cyberattacks. For users it is not clear what protection measures are taken in public networks, such as a public WLAN at an airport or in a hotel. You have to assume that public networks don't offer protection for a digital ID. This is one of the main Identity Access Management (IAM) principles of the Zero Trust Model. For more information about Zero Trust I want to refer to the article "Zero Trust - "Never trust, always verify"" in this magazine.

Author Patrick Fell

To be clear: the usage of IDs is no longer sufficiently protected outside the corporate network and access to IT assets is inevitably at risk. These attacks are not only a problem of private individuals, but also of companies.

The ID (and the devices used by the identity) must be de clared the new perimeter and protected when used.

This is a very important task of an IAM system, that deals with two management aspects:
1. the management of digital identities (IDs)
2. the management of these identities' access to IT assets (resources such as files, emails, web pages, databases, etc.).

Microsoft IAM
To manage digital IDs and accessing IT assets, the IAM ecosystem in Microsoft 365 (M365) offers the following features:
• a platform with a lot of functionality for managing IDs (objects)
• a database for storing digital IDs (objects and their attributes)
• ID authentication and authorization capabilities

The directory service Azure Active Directory (AAD) is the basis for all these features and can manage both the Authorization (AuthZ) and the Authentication (AuthN) of users:

AuthN

Verifies you are who you say you are!
• Login credentials
• Forms-based login
  • HTTP authentication
  • HTTP digest
• X.509 certificates
• Custom authentication methods i. e. MFA

Windows 11
User Name: Patrick_Fell
Password: *********
Login

Who are you?
Validate a system is accessed by the right person.

AuthZ

Decides if you have permissions to access IT assets!
• Access controls for files or URLs
• Access Control List (ACL)
• Discretionary Access Control List (DACL)
• Secure objects and methods

Read - Write
Delete
Send Mail

Are you allowed to do that?
Validate users permission to access IT assets.

Figure 1: AuthN and AuthZ

Beyond these technical features, governing digital IDs (ID Governance) and accessing IT assets (Data Governance) is another added value that M365 delivers.

The Directory Service (DS) component provides the basis for storing the digital ID in a database. Since some companies use several directory services, special DS functions ensure that a federation is established between the different directory services, e. g. to be able to synchronize the objects afterwards. This is also the case when using an on-premises Microsoft Active Directory (AD), which is connected with AAD in the cloud to form a hybrid cloud infrastructure.

Single Sign On (SSO) in AuthN is used so that the user does not have to re-authenticate each time when accessing IT assets. Multi-factor authentication (MFA) is used as an additional security layer, in which an additional secret is required from the user to which only he or she has access. A token-based procedure is used to authenticate the session, i. e. the connection to an IT asset. One of the newer features of AuthZ is to handle authentication without passwords (Passwordless Authentication). This functionality should please users, as they often have to remember dozens of passwords to gain access to IT assets.

AuthZ is used to control resource access. In addition to role-based access, attributes of the user object or certain

DS Services
- Identity Store
  - DS Database
- Federation
- Synchronization
- Virtual Directory

AuthN Services
- Single Sign On
- Multi Factor AuthN
- Passwordless AuthN
- Session & Token Management

User Management Sercices
- Provisioning
- De-Provisioning
- User Self-Service
- Delegation of Control

AuthZ Services
- Rules & Policies
- Roles
- Objects & Attributes
- PIM & PAM

BUSINESS PROCESSES

IAM Governance Framework

Reporting, Monitoring & Analytics

TECHNOLOGIES

POLICIES AND RULES

Figure 2: Components of an IAM Solution

conditions are often evaluated. These methods are called Role Based Access Control (RBAC), Attribute Based Access Control (ABAC), and Conditional Access (CA). For managing privileged access there is a special form, the so-called Privileged Access Management (PAM), often incorrectly referred to as Privileged Identity Management (PIM). PIM is about managing identities for administrators or superusers that control highly privileged access to IT assets. PAM deals with the management of accessing valuable IT assets such as access to privileged accounts from administrators. Exploring PIM and PAM in M365 will be covered in a future magazine.

Whether PIM, PAM or IAM, the principle of Least Privileged Access always applies, which is also a principle of the Zero Trust Model: at the time of access, the digital ID receives a minimum of the rights and permissions required to access the IT asset. In addition, these rights and permissions are only granted during the needed time of access itself and are automatically removed afterwards. This is also called Just-In-Time (JIT) and Just-Enough-Access (JEA).

Please refer also to the article "Ten tips and tricks to secure your Azure subscription" in this magazine. You'll find good tips to harden your IAM infrastructure in M365 and Microsoft Azure.

In the age of automation, user management services are an important component of an IAM system. This is not just about the automated creation – provisioning – of digital IDs, but about the entire identity lifecycle. This starts with

hiring a new employee and goes over to the change of access due to a change within the company, then to a time-out, e. g. a sabatical and ends with leaving the company. Some of these processes should be offered as self-service so the user can request new access to IT assets themselves. This also includes changing a password, which unfortunately still occupies the IT support departments of companies far too much.

DS – AAD

Microsoft Active Directory (AD) is probably the best-known and for many years the most widespread identity store worldwide. However, AD could not really be described as a mature IAM system, because too many activities on domain controllers (On-premise Windows Servers that hold the AD database) had to be carried out manually. Only complex automation through code development and attaching of intelligent solutions from third-party providers could add a certain variety of functions to the AD . This has changed dramatically with Azure Actice Directory (AAD) as a "virtual" counterpart of AD.

With AAD as an identity store, Microsoft has for the first time in recent years provided a complete solution for IAM, PAM and PIM as a cloud service in M365 and Microsoft Azure. With proper AAD licensing, features such as MFA, SSO, Passwordless Authentication, Password hash synchronization, Pass-through Authentication (PtA) and Conditional Access (CA) can be enabled with just a few clicks.

Microsoft has declared war on identity theft with these features. Let's take a closer look at what exactly is behind all the functions and which services can be used within the IAM ecosystem in the Microsoft Cloud. The main focus will be on the components AuthN Services and AuthZ Services.

AuthN Services & Components
Hybrid Identities
When companies operate in a hybrid world – on-premises and in the cloud – they often prefer to use on-premises AD as the source of truth for AuthN requests. This works best if the on-premises AD is coupled with the AAD with Microsoft's own synchronization service Azure AD Connect. In most cases, AD FS (Active Directory Federation Services) is used as a coupling system. This allows the Microsoft Cloud to send its AuthN requests from the customer's tenant to the domain controller in the corporate network. AD FS can be published to the Internet via a Web Application Proxy (WAP Server). The weak point is the availability of the systems for establishing the federation (AD FS) and the WAP systems. Without resiliency, authentication is at risk and will stop

working if the systems fail to function. A remediation could be:
• Password hash synchronization or
• Pass-through-Authentication (PtA)

With PtH, the cloud takes over the authentication of digital IDs. Password hash synchronsation and PtA works without the use of AD FS and WAP. Those are simpler variants than the usage of AD FS and WAP. Anyway, both variants and the use of AD FS and WAP are still recommended. Since AD FS and WAP require very complex and failsafe infrastructures, Microsoft will probably say goodbye to it in the long run. Hopefully customers will follow suit. When it comes to hashes, many people still think of the Mimikatz attack, a password stealing method by which the hash of a password can be successfully reused if the right protective measures have not been taken. For peace of mind: the password hash is double-hashed before it is synchronized via AAD Connect to the cloud. The following figures are intended to illustrate the differences between password hash synchronisation and PtA:

Azure AD Connect
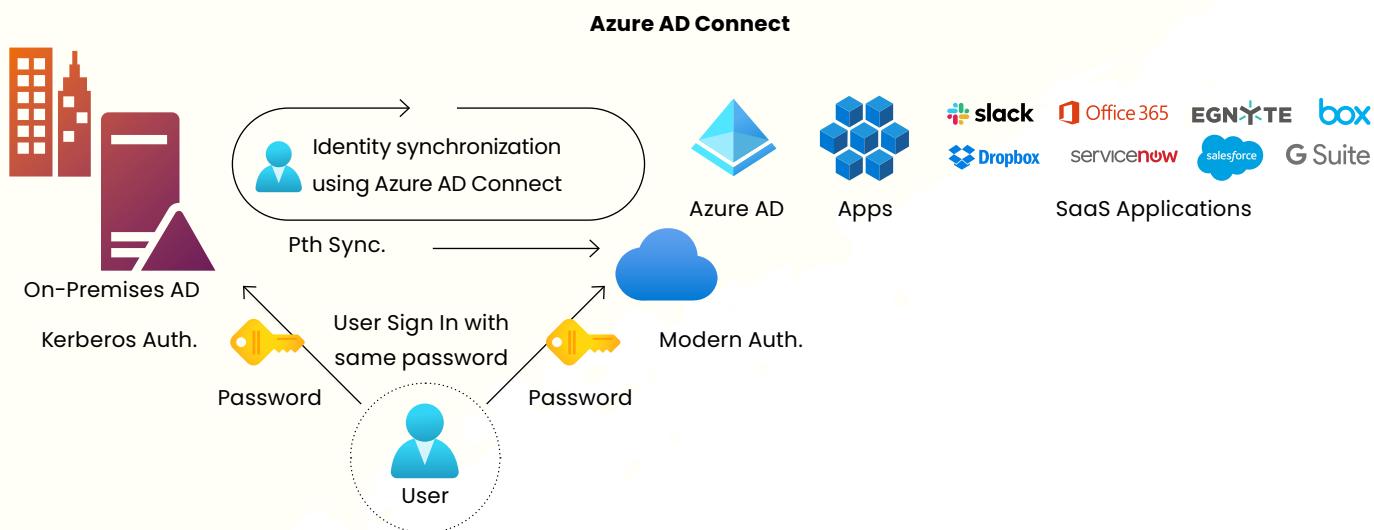
Figure 3: Password hash synchronization
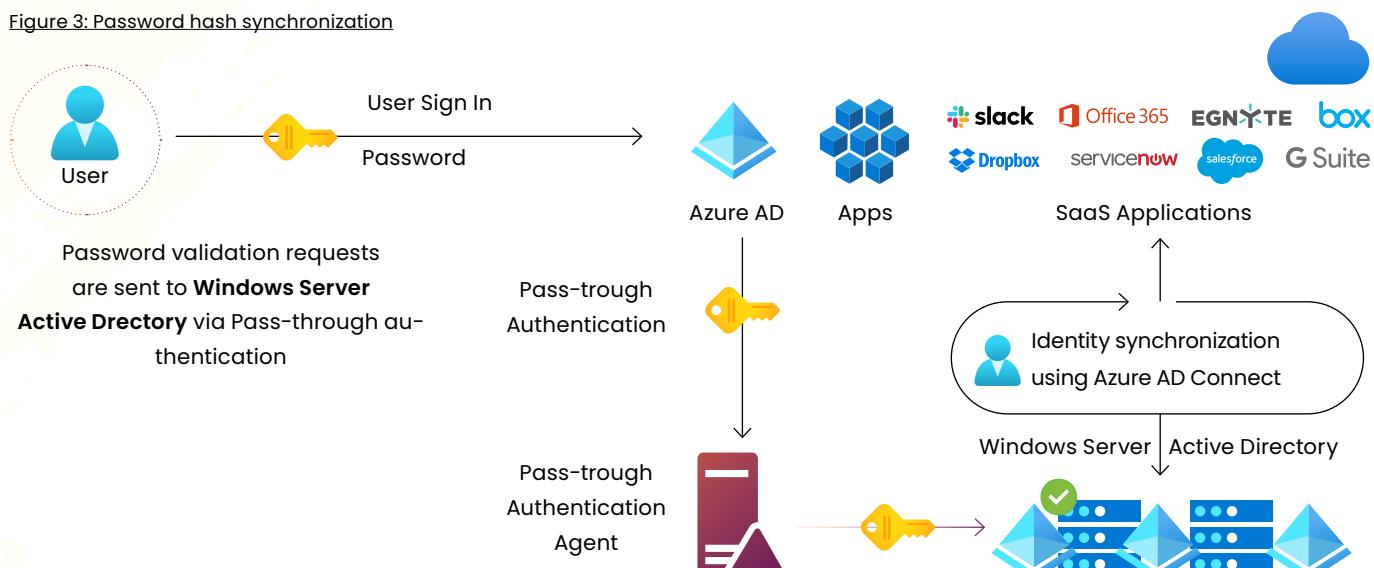
Figure 4: Pass-through Authentication

Cloud Native Identities

The best and most secure option is achieved with cloud native authentication capabilities, which can be supplemented with additional security measures from an Identity Provider (IdP) such as AAD. Cloud native authentication only needs access to the cloud. Connections to or from domain controllers in the corporate network are no longer required, nor is there any synchronization of user objects. The user objects are therefore provisioned directly in the cloud and only stored once. This differs to a hybrid identity, which exists in both the on-premises AD and the AAD.

Multi Factor Authentication (MFA) in M365

MFA is offered in M365 as well as in AAD in various forms:
- Legacy MFA
- Azure AD MFA.

Both variants are included in most subscriptions based on the free tier of AAD. Legacy MFA is enabled directly on the user object. However, we recommend the variant that activates MFA for all user objects by activating the security standards in AAD or Azure Portal (see the article "Ten tips and tricks to secure your Azure subscription" in this magazine). Unfortunately, exceptions cannot be made here, and this option is not available with the Conditional Access (CA) baseline. These must then be deactivated beforehand. Enhanced CA security policies, in turn, require disabling security standards and additional licenses (Azure AD P1 or Azure AD P2) to enable conditional access policies and get the full functionality for IAM.

Conditional Access in M365

Conditional access relies on signals to make decisions for general access and enforce finegrained access to IT assets. The whole thing works in real time and is one of the greatest achievements in IAM. Policies can be assigned to individual users, user groups or the entire company and are super flexible. With policies in monitoring mode, the effects can be checked in a simulation before the actual assignment. It would not be the first time that administrators lock themselves 😊 out of the tenant because of a policy that is too restrictive. Therefore, exactly two phases are supported:
1. Collect session details i. e. about the user, the device or the location
2. Enforce a CA policy in realtime

The downside: Microsoft get's paid for the complexity of the policy with the different requirements for licensing.

Future of Identity In M365

Microsoft Entra is a family of products and a very young representative of services within the Microsoft IAM eco-system. Entra offers a platform with which digital IDs can be subject to governance and permissions. Furthermore the access can be monitored and managed. So far, nothing new for an IAM system. With the Verify ID service, however, there is a new service offering, cross-cloud for multi-cloud environments, for a managed verifiable credential service. It is based on open standards. Not only will employees benefit, but also customers and partners who previously had to come up with their own verifying process for IDs.

How does it work? Verified ID automates identity verification based on open standards and supports privacy-compliant interactions between businesses and users.

AuthZ Services & Components

The complete ecosystem in Microsoft Azure and M365 for assigning rights and permissions is based on the RBAC prin-ciple. Pre-builtin roles (I guess if you take all services into account, you get several hundreds in number) with fix assigned rights & permissions allow administrators to assign users the appropriate access permissions.
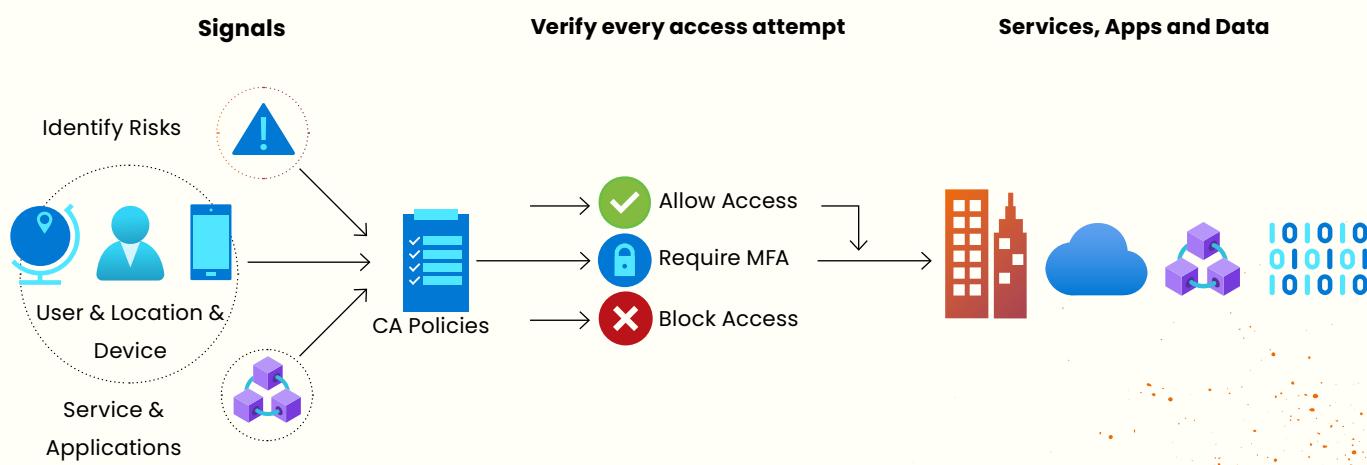

Figure 5: Conditional Access

Issuer

Assests to claims and grants a digitally signed credential to the user.

An issuer could be an ID verification provider a govrnment agency, a previous employer or any other organization that can proof Patick's credentials.

Individual

Receives and approves credential request from issuer, stores and manages credential in a wallet and presents it to verifier.

Verifier

Requests proof and, upon receipt, verifies the claims in credentials satisfy requirements. A verifier could be a employer, a hotel or a border control or any organization that requesting proof of Patrick's credentials.

Verifiable data registry

Write   Read & Trust

Figure 6: Verifiable Credentials

This is done by a user becoming a member of a role group. The role group in turn is assigned the roles that have rights and permissions. The rights and permissions are inherited to the user. Roles are dispearsed over M365 services and AAD. Some are used in AAD and the service, some are unique in AAD only and not visible to the service and some are used by more the one services. This can be very confusing.


Figure 7: Role-based authorization assignment in M365

You can very quickly get lost in the depths of rights and authorization administration in the Microsoft 365 Admin Center (see figure Figure 8: Role Assignments in M365). It is advised to get along with the roles and role groups proposed by Microsoft. This is especially true at the beginning of the M365 cloud journey. The existing roles are sufficient for the delegation of special rights and authorizations and can be assigned via policies.

Figure 8: Role Assignments in M365

However, a role-based access concept is needed that makes the control and coordination of access to IT assets transparent and reflects the current status. If your company is subjected to a security audit, e.g. according to ISO 27001 or BSI Grundschutz, the auditor can ask you about this concept. So remember that you are writing one. The HR-based lifecycle plays an important role: administrators do not want to implement role changes on demand. These should be requested by the user based on self-service or automatically adapted based on a job or department change. Then the trigger has to come from the HR department. The principle of SoD – Segregation of Duties must always be observed: Administrators may not add or remove themselves from certain role groups. This requires at least one other administrator to monitor and implement the process.

User Management

To be honest, there are not many great features in the area of automated provisioning and deprovisioning of user objects. However, this is increasingly due to the fact that the creation of new user objects, e. g. as part of a new hire, is not an independent discipline of IT. This is based on the HR-based lifecycle. Typically, it is the HR department who has to be the first to take action when it comes to the maintenance of master data for people. Only when a master date is successfully created, e. g. in SAP, a trigger can be fired. This automatically creates a user object with all the required attribute values in the DS database and provisions the user object. Two processes exists, the HR-Based Lifecycle and the Identity Lifecycle, which absolutely should not be decoupled from each other.

In IT, we use IAM to focus on the identity lifecycle rather than the HR-based one. However, we must ensure that interfaces for provisioning and deprovisioning from master data maintenance systems are correctly addressed. The initial filling of the identity store with digital IDs is therefore not the sole responsibility of IT. It always requires a fireing trigger from the master data maintenance systems, especially since licensing in Microsoft 365 is user-based. License management, like IAM, is therefore dependent on the correct maintenance of user master data.

Self-service Password Reset (SSPR) by the user is old hat. Especially with this self-service offer, IT support departments can be significantly relieved. According to Gartner, these requests accounted for 20% of all requests from employees if a self-service password reset cannot be used. There are several reasons that require a password reset, including:

• User objects are locked
• Users have been forgotten their passwords
• Passwords have expired and were not renewed in time
• Password changes by the user failed

The following self-service portal is available to the user in Microsoft 365 and Microsoft Azure. These portal can be reached via myaccount.microsoft.com by the user.

Patrick Fell

Security info and paswords can be managed here:
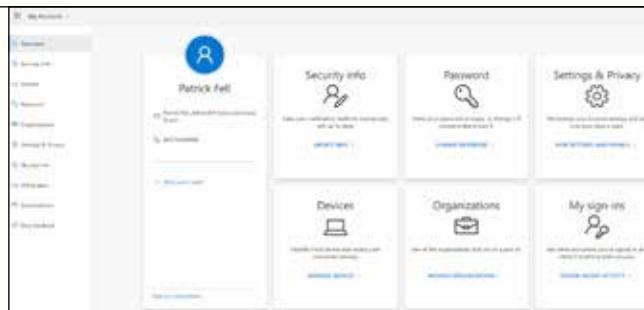
Figure 9: User Self-Service Dashboard

Figure 10: Manageing security information

However, SSPR must first be enabled in the Entra Admin Center:
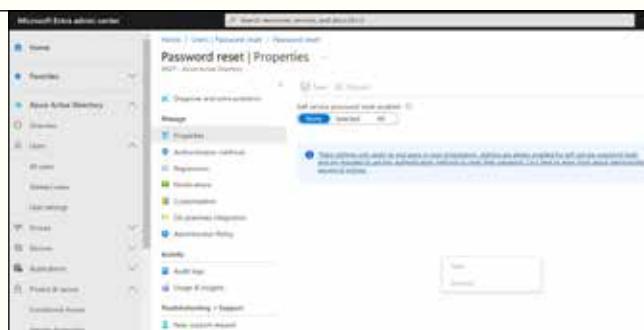

Figure 11: Enabling SSPR

Starting in January 2024, legacy policies for multi-factor authentication and self-service password reset will be discontinued. From then on, all methods will be managed via authentication method policies, including passwordless authentication:

Figure 12: Configuring Authentication Methods

Summary

In summary, there are five steps for securing the identity infrastructure via an IAM system that are very helpful.

Before you begin your journey to protect identities and use enhanced IAM solutions, you should protect not only privileged accounts but also all user accounts via MFA.

Step 1 - Strengthen your credentials, especially password length

Step 2 - Reduce your attack surface area, especially using modern authentication protocols and controlling entrypoints for authentication

Step 3 - Automate threat response, especially by using conditional access policies

Step 4 - Utilize cloud intelligence and automate IAM processes, especially by monitoring Azure AD

Step 5 - Enable end-user self-service, especially for self-service password reset (SSPR) and configuring security information

This could avoid approximately 90% of all successful cyberattacks on digital identities.
</>

READ MORE ONLINE

Unpacking Access Packages

Introduction To Azure AD Access Packages and How We Used Them In A Real-World Customer Scenario.

How great would it be if users could enroll to a set of Azure AD Groups, Applications or SharePoint sites themselves, instead of jumping through all kinds of bureaucratic hoops before access has been granted and the user can do their actual job?

Author Rik Groenewoud

Not only would this be great for the end user, but also from the administrator's point of view this would be the ideal scenario. Instead of maintaining a custom enrollment process with lot of manual steps this process shifts the action to the end users themselves. This makes it possible for the administrator to focus on maintaining a secure and compliant system, instead of doing repetitive simple tasks.

As I will show you in this article, access packages are here to do just that! I will dive deeper into what these packages are

all about. Firstly, I will explain what these packages are, what choices you have when setting them up and how a basic use flow will look like.

After the basics are clear, I will give a real-world customer use case of how we at Xpirit Managed Service leveraged Access Packages to create a highly automated enrollment process for a complex Identity and Access Management scenario.

What are Access Packages?

In essence, access packages are a way to manage access to Azure resources in a streamlined and efficient manner. With these packages, you can group together a set of Azure resources that are typically used by the same team or group of users and assign specific access permissions to that package.

In addition to streamlining access management, they also provide self-service capabilities for end-users. This means that users can request access to specific packages themselves, rather than having to go through an IT administrator or department. When a user requests access to package, an approver can review the request. This provides an additional layer of control and ensures that access to resources is always aligned with the organization's security and compliance requirements.

To be able to leverage access packages you need an Azure AD P2 or Enterprise Mobility + Security (EMS) E5 license.

Configuration of an Access Package

The packages live in Catalogues which are containers for one or more packages. After you have given the package a name and description you proceed by adding the resources you want the users to enroll on. These can be Azure AD Groups, Enterprise Applications and/or SharePoint sites.

The next step is to decide who can enroll to the package. There are 3 main options.

1. Users in your directory

This option makes it possible to further specify whether all members, or specific users, may request access to this package. Furthermore, you can decide if manual approval is needed. If yes, you can determine if users need to write a justification, if the user's manager or a specific other user (or users) will be the approver and in what timespan the decision must be made. It is even possible to create a second line of approvers for when the first approver did not decide in the given time span.

In short, with this option it becomes possible to really create a granular least privilege structure for your access packages. By doing so it becomes easy to align to the company's policies and governance regarding User Access Management.

2. Users not in your directory

With this option it becomes possible to specify particular external organizations, open up enrollment to all connected organizations or to select all users meaning all connected users plus any new external users. With this option you can add or skip the approval flow.
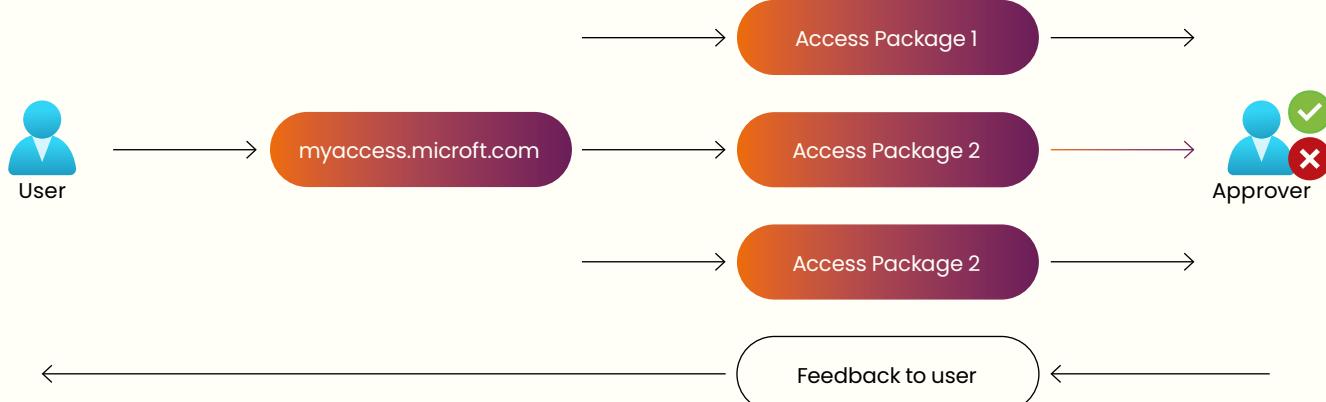
3. None (administrator direct assignment only)

This last option means that only administrators can add people to the access package. This is the best option if there is no approval flow in place and users should not be able to enroll themselves in the packages.

If needed, you can ask users for additional information when applying for access by using custom parameter fields. You can determine a lifecycle for the access package assignment. And finally you can determine whether an access review is needed. The idea behind these access reviews is to check if all package enrollments are legitimate and up-to-date.

Self-Service Onboarding

After you have setup the catalogue and created one or more access packages, a typical happy flow self-service scenario would look something like this:

The end-user goes to https://myaccess.microsoft.com and after login with his/her Azure AD Credentials, the landing page with all available packages will be shown. The user can request access for the desired packages after which an approver receives an e-mail with the pending approval. After approval, the user receives feedback via e-mail and gets the resource roles that are given to the access package.

Customer Use Case: Automated User Onboarding in a Web Application with Complex roles and rights structure

So far, we have looked at the fairly straightforward self-service scenario in which access packages can play a very useful role. But there are more use cases for access packages. At Xpirit Managed Services (XMS) we used them in a slightly different manner. We received a request from our customer to build an automated onboarding process for their Azure AD users. The user should be able to login to the web application using SSO and should be automatically assigned to the correct roles and projects.

What makes this challenging is the fact that this application has a complex roles and rights structure. It is made up of 20+ separate projects, all with three or more separate roles per project. A quick calculation shows that we are talking about 60+ project/role combinations. Several hundred users should be able to be enrolled in multiple projects and have potentially multiple roles within these projects.

Then there are special key users who should be able to have elevated rights in multiple projects and users who only need to have reading rights in all projects. To make it even more complex, the users are all from different companies working together in this project.

In the next part I will explain what we came up with to solve this challenge.

SSO and Roles Mapping

To make SSO possible an App Registration + Enterprise Application already was in place. The first step was to map all the roles from the application, to App Roles in the App Registration. In the JSON Manifest this looks like this:

```json
"appRoles": [
        {
            "allowedMemberTypes": [
                "User"
            ],
            "description": "ProjectX_RoleY",
            "displayName": "Project X Role Y",
```

```json
            "id": "8e9bd73b-e64f-46e5-b4b6-481234",
            "isEnabled": true,
            "lang": null,
            "origin": "Application",
            "value": " ProjectX_RoleY "
        }
]
```

The next step is to map Azure AD (AAD) Groups onto these App Roles. There is a 1-on-1 mapping between the App Roles and the AAD Groups. The mapping has to be done in the Enterprise Application:

To make this more rigid and maintainable, we placed the App Registration JSON manifest in a Git repository and created an Azure CLI script to update this manifest using the az ad app update cmdlet.

Finally, to make the mapping between the App Roles and AAD Groups, a PowerShell script loops over all AD Groups like:

```powershell
foreach ($group in $aadGroups) {
    $role = $roles | Where-Object -Property value -eq
$group.Displayname

    $params = @{
        PrincipalId = "$($group.Id)"
        ResourceId  = "[Resource-Id]"
        AppRoleId   = "$($role.Id)"
    }
    New-MgGroupAppRoleAssignment -GroupId $group.Id
-BodyParameter $params
}
```

With this mechanism the first part of the puzzle is solved. Now the users need to be enrolled in the AAD Groups. It is time for the access packages to make their entrance.

Access Packages

For every project, 3 separate packages were created corresponding the roles (and the AAD Groups) in the application: the Approver, Contributor and Manager.
In these packages the corresponding AAD Groups are selected and also a Default Users AD Group is added, which is mapped to the global reader role in the application.

Furthermore, for the Key Users and Reader Only users, separate packages were created with all the appropriate AD Groups. For this use case this functionality really shines because with a single enrollment the user is added to all these AD Groups at once.

Because the end users in this scenario don't know to which access packages they belong (at least for now), the self-enrollment options were disabled and instead "administrator assignment only" was the way to go. Also, because the administration and approval of user assignment is done beforehand, the approval flow could be disabled as well. No end-date on the enrollment was needed because users that are no longer eligible to be in an Access Package are automatically removed (I will explain more on this later).

The access packages are not created in an automated way because this was a one-time job and can be done pretty easily from the Azure Portal.

With the access packages in place, we came to our final step: the enrollment of users into the packages.

User enrollment

For the user enrollment the goal was to create a solution which fits the XMS way-of-working. At XMS we always seek to work together with our customers and thereby enable a customer to work in a DevOps way. By doing so. It is important that we don't create boundaries between different stakeholders or between the business and IT, but work together and build smart processes in which repeated tasks are automated as much as possible.

In this case, where a traditional service provider would setup a ticketing system in which the customer can ask to enroll new users, we wanted to make this a collaborative and smooth process. This process now consists of 3 steps:

1. For every access package a simple .CSV file was created in which the customer can add or delete users as desired:
   Displayname;Email;AccessPackageUser
   Z;userz@example.com;Project X | Role Y

2. These .CSV files are part of a repository and the customer can create a PR with the new changes. These PRs are validated by the XMS team.

3. After the PR is merged, an Azure DevOps pipeline is triggered that runs a script to enroll the users from the CSVs. It also creates a new AD Guest User if the user does not exists in our tenant. Finally, a check is done on removed users from the CSV file. These get removed from the packages as well.

Some code snippets from this script:

```powershell
# Invite user if not yet exists
New-MgInvitation -InvitedUserEmailAddress "$($user.Email.Trim())" -InviteRedirectUrl "https://example.com/invite -SendInvitationMessage:$true

# We check if the user already is member of the package and if not, the user is added.

$check = Compare-Object -DifferenceObject $assignments.Target.ObjectId -ReferenceObject $AADuser.Id -ExcludeDifferent

if ($null -eq $check) {
$policy = $accessPackage.AccessPackage-AssignmentPolicies[0]

New-MgEntitlementManagementAccessPackage-AssignmentRequest -AccessPackageId $accessPackage.Id -AssignmentPolicyId $policy.Id -TargetId $AADuser.Id

Write-Host "User $($user.Displayname) is added to Access Package"
}

# We check if there are differences between the CSV and the assignments and if yes, we remove the users.

$check = Compare-Object -DifferenceObject $assignments2.TargetId -ReferenceObject $userid
Write-Host "Differences in IDs: $($check2.InputObject)"

if ($null -ne $check2) {
    foreach ($assignment in $check2.InputObject) {
       #Get AssignmentId for user that has to be removed
       $assignmentId = ($assignments2 | Where-Object
       { $_.TargetId -eq $assignment }).Id
       New-MgEntitlementManagementAccessPackage-AssignmentRequest -AccessPackageAssignmentId $assignmentId -RequestType "AdminRemove"
       Write-Host "Removed $assignment"
    }
}

else {
    Write-Host 'No users removed'
}
```
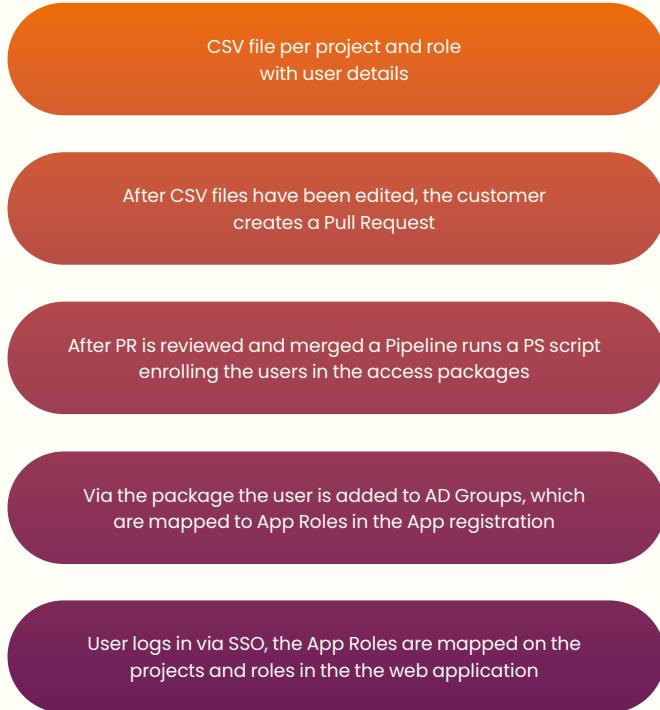
The diagram below summarizes the process flow of this solution

> CSV file per project and role
> with user details

> After CSV files have been edited, the customer
> creates a Pull Request

> After PR is reviewed and merged a Pipeline runs a PS script
> enrolling the users in the access packages

> Via the package the user is added to AD Groups, which
> are mapped to App Roles in the App registration

> User logs in via SSO, the App Roles are mapped on the
> projects and roles in the the web application

Future improvements

This process has been running for a few months and we are quite happy with it. Also, it is important to state that we take on these kind of challenges iteratively. What started with the customer sending us Excel sheets and us enrolling the users manually to the access packages (and thereby already adding value for our customer because the users could access the application with the correct roles), evolved towards the process it is now.

This does not mean it is a perfect solution and there is still room for further improvement. The next step would be to move towards self-service. We must make sure the access packages correspond to what the end users understand. We can appoint approvers at every separate company. With this in place we should be able to remove the CSV-files (and more importantly the maintenance of these CSVs) entirely. This will result in a simpler, leaner process.

To conclude

This article has shown what access packages are, what is their potential and how they can be useful in self-service scenarios.

In our customer use case, I showed that these packages can be a valuable piece of the puzzle when it comes to creating a maintainable solution for a complex Identity and Access Management scenario.

If you have any questions about this subject or how you could use access packages in your environment, don't hesitate to reach out!
</>

Rik Groenewoud

READ MORE ONLINE

ACCESS GRANTED

OAuth2 Device Authorization Grant proxy

Have you ever needed to call REST APIs from an embedded device or a console app? If so, you likely needed some OAuth2 credentials to prove who you are and what you are allowed to do.

Author Hans Bakker

Getting these tokens on your device in a proper way could be a pain - hardcoding credentials in your code is ugly, and hosting an embedded web server to let a user sign in is also something you likely hope to avoid.

Fortunately, there is the Device Authorization Grant[1] flow, also known as Device Flow. It is one of the standardized OAuth2 authentication flows and its usecase is to enable applications, having limited interaction capabilities themselves, to get authenticated.

Think of embedded devices or console applications for example - they cannot present the user easily with a login page without hosting an embedded webserver. These devices or applications will be called device in this article to emphasize the role they play.

Unfortunately, not all services offer the Device Flow. To mitigate this, it is possible to build a proxy that enables using this flow, while the proxy uses one of the more common other flows (like the Authorization Code flow) under the hood.

For this example, we will focus on creating such a proxy for Spotify's REST API. As a challenge, we will try building this proxy using free services in Azure.

While Spotify in practice does have a device code flow for some partners, and for its own Apple TV and Android TV apps, it does not support it for third-party applications. Some background is described in Reverse engineering Spotify's own Device Authorization Grant implementation.

---

[1] https://www.rfc-editor.org/rfc/rfc8628
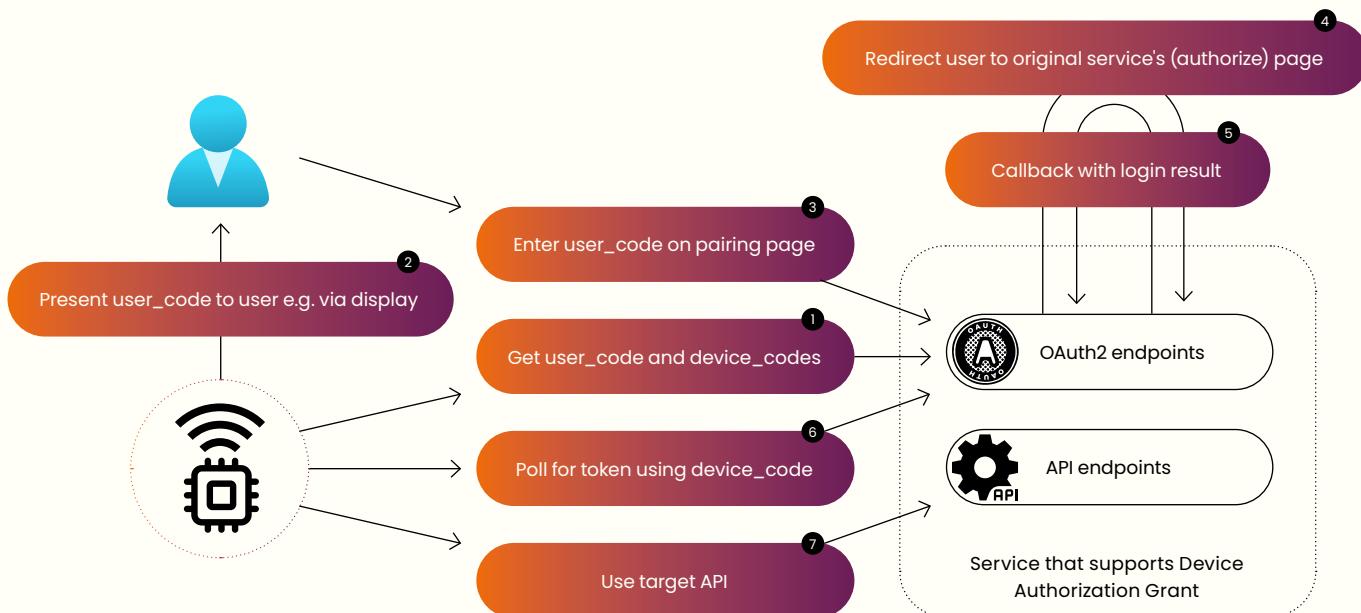
Figure 1: Device Flow (normal setup – no proxy)

How does the Device Flow work?

Before we go into the details of adding a proxy, let's discuss the original Device Flow which is shown in Figure 1. Instead of directly presenting the user with a login page, the device requests a pair of two codes (step 1): one for the device, and one for the user. The device presents the user with the user_code (step 2), which can be shown on a display as numbers or a QR code, or it can be read aloud in the case of a voice assistant. The user can then enter this user_code in a browser (step 3) on a different device that does have better interaction capabilities – a desktop or smartphone for example. After that, the user is asked to authorize the request (step 4) via the normal authorization flow of the service. The device can poll for OAuth2 tokens in parallel (step 6) using the device_code, which become available upon completion of the authorization by the user.

The device can use those tokens to make API requests (step 7) as with any other OAuth2 mechanism.

Device flow with proxy

As Spotify does not officially support this flow for custom applications, it is also possible to implement your own device authorization grant flow by hosting an extra component (called proxy below) between your device and Spotify. The modified flow can be seen in Figure 2.
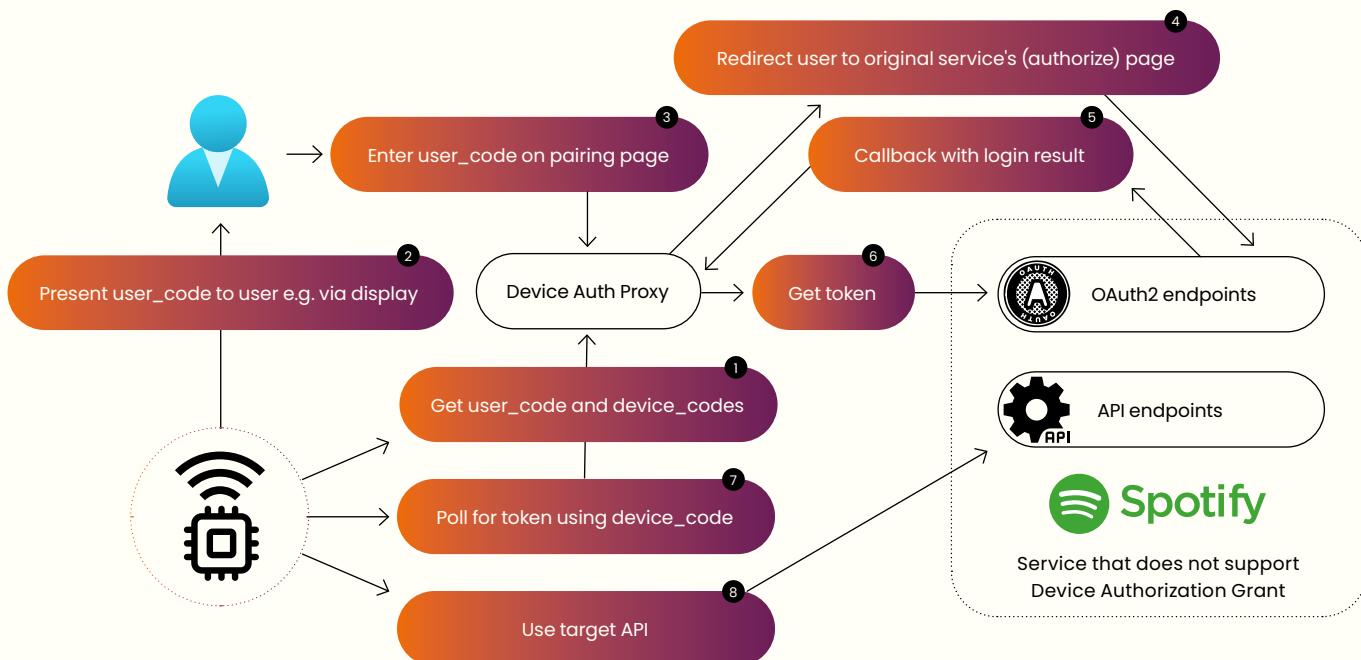

Figure 2: Device Flow with proxy

1. The device calls the authorize endpoint. Now, the proxy should generate a record containing the `device_code` and `user_code`, save them in the database for later and send them back in the response.
2. The user code needs to be presented to the user, via a display or a speaker for example.
3. The user enters the `user_code` in the user-facing page. The proxy checks the `user_code` in its database. If the `user_code` is not found, an error is shown.
4. If the `user_code` is found, the proxy redirects the user to Spotify to login and approve the request.
5. After login, the user is redirected to the proxy's callback page. The redirect contains the result of the user's action and an authorization code.
6. The proxy can fetch an `access_token` and `refresh_token` from Spotify's token endpoint using the authorization code it received in the callback. The proxy should store the credentials it received in the earlier-stored record, next to the `user_code` and the `device_code`.
7. The device can poll the proxy's token endpoint using the `device_code` for the availability of the tokens. When the device has successfully fetched the token, the record should be deleted from the database to prevent abuse.
8. The device can call the target API using the obtained tokens as usual.

Device flow proxy architecture
Requirements
The proxy should
- host an authorize API endpoint and a token API endpoint for the device to interact with
- generate the `device_code` and `user_code` and store it in a database
- host a user-facing page containing a small form where the user can enter the `user_code`
- retrieve and validate the `user_code`
- host a callback endpoint for Spotify to, after the user logged in, redirect the user to. This endpoint should process the data sent by Spotify, and should show a success or error page to the user.

For this, the proxy should have a very simple frontend with a few backend API endpoints, as can be seen in Figure 3.

Component choice
For our challenge, we will choose from the free services[2] that Azure offers.

⚠️ **Note:** Often the free services come with reduced specifications and / or a reduced SLA. Do not blindly use them for production purposes. However, they can still fit the purpose of a small application for personal use.
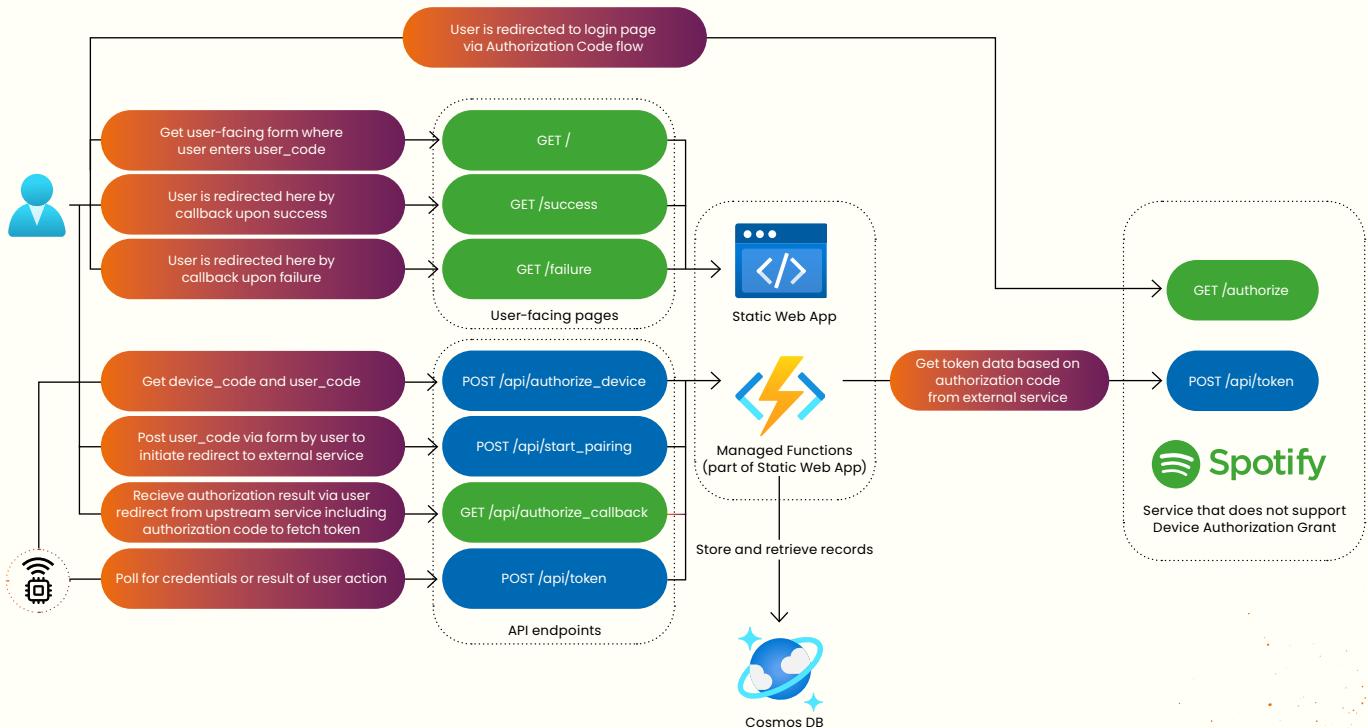

Figure 3: Detail architecture of proxy and API endpoints

There are two variants of free services:
1. services that are free during the first year of your Azure Account
2. services that remain free for the total lifespan of your Azure Account

For our Device Flow proxy, we need 3 main capabilities:
• a place to serve the frontend (form where user enters the `user_code`) and some result pages
• some API endpoints with compute and integration functionalities
• some database / storage to store the authentication records that are active

For this challenge we will use these components which fit in the second category (free forever).

This limits our choice to Static Web Apps, App Service and Azure Container Apps for the frontend / backend capability, and Cosmos DB for the database capability.

Frontend / backend
Azure Static Web Apps (SWA) can serve a static frontend and its free tier has a slimmed down version of Azure Functions called Managed Functions which can serve as a backend. For a simple web application it covers the needs, and it has no limits on the time it is running. For us, this is a good choice.

SWA has a free monthly amount of 100 GB bandwidth per subscription, 2 custom domains and .5 GB storage per app, which is more than enough for our proxy app.

The Managed Functions are a 'supported' API in the free tier of SWA. There are a few other products that are also available as supported APIs in SWA, but not as part of the free tier – they are `Bring your own Functions`, `API Management`, `App Service` and `Container Apps`.

The idea of using a supported API product in SWAs:
• the API and static website are served from the same domain via the SWA's built-in reverse proxy. This removes the need to add Cross-Origin Resource Sharing (CORS) headers on the API responses, so it makes the developer experience less complex.
• Routing is done automatically.
• The authenticated user context from the SWA is available to the API logic.

Managed Functions will cover our needs, but it is good to know that it does not support all features[3] you might have come to expect from other Azure Functions offerings. Notable differences are:
• No Function triggers other than HTTP triggers
• Some security best practices are not possible to be followed
  • Currently no `Managed Identity` support[4]
  • no support for Key Vault references

Alternatives considered to SWA were:
• Azure Container Apps has a lifetime free monthly amount of 180,000 vCPU seconds, 360,000 GiB seconds, and 2 million requests, but that has a dependency on a container registry. Azure Container Registry is only available for free for 12 months.
• App Service has a lifetime free monthly amount of 10 web, mobile, or API apps with 1 GB storage 1 hour per day – not chosen because preference for serverless model. 1h/day is probably enough but feels difficult to estimate how it works out.
• Azure Functions Consumption plan might look free (1 million invocations and 400000 GB-s), but it has a requirement of providing a storage account which is not free.

Database / storage capability
Cosmos DB is the only database or storage product that is offered as a free forever service. It fits our needs as it can store our authentication session records as JSON documents and is easy to work with.

The free tier of Cosmos DB offers a free monthly amount of 1000 request units per-second provisioned throughput with 25 GB storage which is more than enough for our proxy application.

Solution for time-based logic
We want to cleanup data for security reasons (discussed in the section below) after a certain period or when the data is obsolete. The Managed Functions offering in SWA only supports HTTP-triggered functions, so timer-triggered functions are not supported.

Technically a separate Logic App might be an option, but using the Cosmos DB-native Time to Live (TTL) feature is a much simpler and more elegant option[5].

Since we only need time-based cleanup and no other timer-triggered runs, we will go for the native cleanup in Cosmos DB by setting a TTL on the records.

[2] https://azure.microsoft.com/en-us/pricing/free-services/
[3] https://learn.microsoft.com/en-us/azure/static-web-apps/apis-functions
[4] https://github.com/Azure/static-web-apps/issues/88
[5] https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/time-to-live

Security considerations

Since we are essentially building a custom part of the authentication chain here, it is very important to pay attention to security.

Basics, like not committing secrets to git, not hardcoding secrets, using HTTPS, and so on will not be covered in this article. However, the RFC gives some specific guidance related to the Device Authorization Grant logic which is good to pay extra attention to. This extra guidance basically boils down to preventing brute-forcing and phishing of the secrets.

- use of a long enough user code and device code. This increases the amount of tries that it will require to guess codes.
- rate limiting on proxy endpoints to prevent brute-forcing. This reduces the amount of tries that a bad actor can perform.
- expiry and cleanup of authentication session record from the database after period X or when the flow is completed by the device (the OAuth2 tokens are received by the device). This ensures that the data is stored not longer than necessary.

Reverse engineering Spotify's own Device Authorization Grant implementation

While reading up on the Device Authorization Grant, I found an article showing that Spotify offers this flow for its own apps and for partners[6]. The Android TV app is one of them, and one benefit of Android apps (for us 😉) is that they are relatively easy to reverse engineer. It is possible to download its application package (.apk file) and process it in a decompiler. There are several online decompiler services where you can upload the apk file and download the decompiled java source files and accompanying resources as a zip file. After opening the unzipped files, they are easily searchable using Visual Studio Code. Now we need to use a bit of educated guessing and puzzling. The developers often try to obfuscate their code to make our search difficult by mangling variable names, function names etc. An idea to circumvent this is to search for identifiers or (parts of) URLs, that are related to the logic we are researching and that we expect the app to use. The reason for this is that the identifiers or URLs themselves are usually not mangled because they need to be used in HTTP request / response bodies or the URLs that need to be called. Candidates for search terms could come from reading the Device Flow's RFC[7], from Spotify's own authorization documentation[8] (expecting that this flow will have some common or similar endpoints) and by iterating further on what we find.

Hans Bakker

---

6   https://pragmaticwebsecurity.com/articles/oauthoidc/device-flow.html
7   https://www.rfc-editor.org/rfc/rfc8628
8   https://developer.spotify.com/documentation/general/guides/authorization/

Examples of concrete search terms that are leads for us are:
- `client_id`
- `accounts.spotify.com`
- `device_code`

This way, we could find
- the `client_id` of the app registration of Spotify's Android TV app
- that part of the app's functionality is built around a website hosted at https://api-partner.spotify.com/tvapp?platform=androidtv
- the special scopes that the app uses
- the URLs and calls that the app makes to obtain tokens.

I presented the findings in a topic on the Spotify Developer forum[9].

Conclusion

In this article we discussed the OAuth2 Device Flow, and looked at how we can build a proxy for services that do not offer this authentication flow. We discussed the requirements for such a proxy, and looked at what free components Azure offers to fulfil those requirements. We discussed the security considerations and how to deal with them. Finally, we had a look into how one could reverse engineer the Device Flow that Spotify has but does not currently offer to non-partner developers.

`</>`

Reference
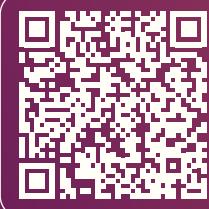
Other relevant articles:
- Everybody wins with the Device Flow
  https://pragmaticwebsecurity.com/articles/oauthoidc/device-flow.html
- Using the OAuth 2.0 device flow to authenticate users in desktop apps
  https://thomaslevesque.com/2020/03/28/using-the-oauth-2-0-device-flow-to-authenticate-users-in-desktop-apps/
- Authentication In Smart TV App - Device Code Flow
  https://www.c-sharpcorner.com/article/authentication-in-smart-tv-app-device-code-flow/
- Illustrated Device Flow (RFC 8628)
  https://darutk.medium.com/illustrated-device-flow-rfc-8628-d23d6d311acc

Other implementations:
- Spotify player for vintage Macs
  https://68kmla.org/bb/index.php?threads/building-a-spotify-player-for-my-mac-se-30.32182/
- MacAuth (ASP.Net Core based)
  https://github.com/antscode/MacAuth
- Add the OAuth 2.0 Device Flow to any OAuth Server (PHP based)
  https://developer.okta.com/blog/2019/02/19/add-oauth-device-flow-to-any-server

---

9 https://community.spotify.com/t5/Spotify-for-Developers/Device-Authorization-Grant-authentication-flow-for-custom/m-p/5485468

READ MORE ONLINE

Zero Trust – "Never trust, always verify"

In May 2021, President Joe Biden signed an executive order to adopt the Zero Trust security model for federated agencies. This has become a top priority for the US government. Federated agencies have until September 2024 to implement the Zero Trust model. The United States has experienced firsthand how cyber threats are becoming more sophisticated. In May 2021, the Colonial Pipeline, an American oil pipeline, fell victim to a ransomware attack that resulted in a six-day shutdown. The attack caused fuel shortages, flight rescheduling, filling stations running out of fuel, and skyrocketing fuel prices.

Author Patrick van Kleef

Why do we need Zero Trust

Times have changed, and we have adopted a more dynamic way of working. In particular, after Covid, people have shifted from working in offices to working from home. This means that people access workloads from unsecured public networks and use different devices such as mobiles or tablets. Consequently, applications should be available from more than just the corporate network. Previously, we had only one entry point for applications within the network. Nowadays, this is no longer feasible, and we have shifted away from a closed perimeter. As a result, our infrastructure has become more vulnerable to attacks from different angles.

Cybercriminals always strive to be one step ahead and are becoming increasingly intelligent and creative in their efforts to infiltrate our networks and systems. They seek out weak spots in our security perimeter, and sometimes those weak spots are the people themselves. Social engineering has become so sophisticated that psychological manipulation is used to gain access to high-privilege accounts. We used to think weak spots were only present in our software and hardware. However, once a cybercriminal has access to an employee account, they can gain access to internal systems and valuable data. Zero Trust is a response to this threat.

Principles of the Zero Trust Model
Verify explicitly

The traditional security model relied on implicit trust, assuming everything on the network was safe and anyone inside the network had unrestricted access. However, this assumption is outdated, and we can no longer rely on the idea that everything is safe behind the firewall. With Zero Trust, we verify every identity, regardless of whether the request comes from inside or outside the network. We aim to authenticate and authorize all data points, as Zero Trust assumes that bad actors can be found everywhere, including inside your organization.

Use least privilege access

Instead of granting sweeping access to identities, Zero Trust principles dictate that we should provide the least privileged access. Use Identity Access Management (IAM) to assign an identity only the minimal access rights required to complete an operation. In many cases, it is not necessary to give an identity permanent access, especially when dealing with highly privileged access. Instead, use Just-In-Time (JIT) and Just-Enough-Access (JEA) mechanisms.

Assume breach

Assume that there are malicious actors on the network and take steps to protect resources accordingly. When dealing with a hack, minimizing the blast radius is important. One way to achieve this is to isolate workloads as much as possible through network segmentation. However, be careful to keep your architecture simple, as complexity can introduce additional security risks.

Implement Zero Trust

The five steps to approach Zero Trust.

1. Define the protect surface. Break down your environment into smaller pieces that you need to protect.
2. Map the transaction flows. Investigate dependencies, inbound and outbound connections and how data flows through the network.
3. Architect a Zero Trust environment. Use the Zero Trust principles to design an architecture to protect your protect surface.
4. Create Zero Trust security policies. Use the Kipling method (who, what, when, where, why, how) to develop security policies.
5. Monitor and maintain. Monitor signals to detect any risks, remediate risks and improve the Zero Trust Architecture and security policies.

An organization's attack surface refers to the areas in which bad actors can gain unauthorized access to the network. The attack surface is typically quite large because the entire internet can be considered part of it. We refer to the applications or systems that we want to secure with Zero Trust as protect surfaces. An organization may have multiple protect surfaces, each containing a DAAS (Data, Applications, Assets, Services) element. These resources are defined within each protect surface.

To illustrate how to apply the principles of Zero Trust in practice, I will use the SmartMoney application from the fictional company OneFinance as an example. Please note that this article does not provide an exhaustive list of all Azure services and features that can be used to protect applications. Instead, the focus is on the SmartMoney application.
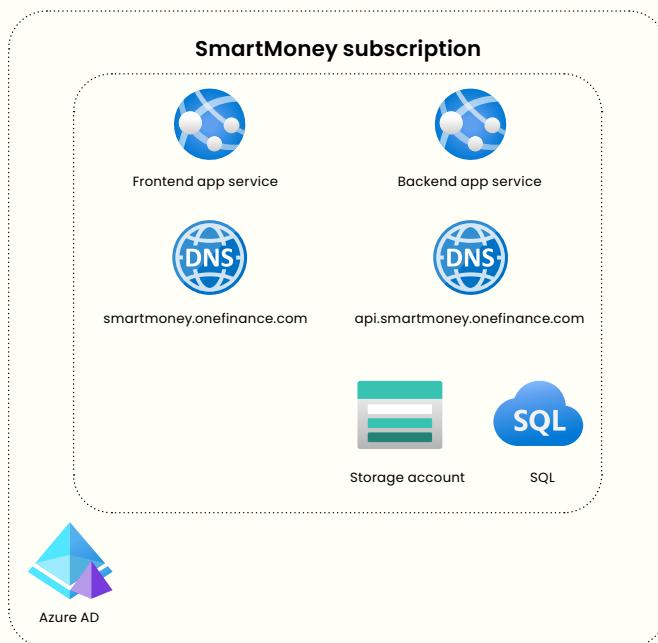
Use the principles of Zero Trust to secure SmartMoney

SmartMoney is an application developed by the fictional company OneFinance, which manages financial data for thousands of customers. SmartMoney helps customers gain insight into their personal finances and provides advice on becoming financially independent. The customer service department is responsible for managing all of the customer

data. Based on this data, the expert department provides advice to customers on how to save costs and create monthly budgets. Two years ago, OneFinance migrated all of its workloads to Microsoft Azure. Employees use their Azure AD account to authenticate. The SmartMoney solution is split into a frontend application and backend application that contains a set of APIs. Data is stored in Azure SQL and Azure storage account. See the current architecture in the image below.

During the COVID-19 pandemic, OneFinance, like many other companies, allowed employees to work from home to prevent business interruption. Before the pandemic, the application was only accessible from the office IP address. The list of allowed IPs was extended to ensure employees could work from home with the application.

OneFinance has many applications, some of which are used internally by employees, while others are publicly available to customers. SmartMoney is identified as a protected surface that we want to protect by following the Zero Trust principles. Other protected surfaces could include the HR system, the intranet, or the public website.

Current architecture

The Zero Trust security model consists of six defense areas: identity, endpoint, applications, data, network, and infrastructure. Each of these areas provides a layer of protection. In this article, I will focus on four defense areas to secure SmartMoney. I will begin with the network defense area.

## Create a micro-perimeter and use network segmentation

Traditionally, we would have centralized network-based perimeters to secure workloads in the network. A firewall is placed in front of the network to keep malicious users outside. Each workload running inside the network has the same attack surface. In this approach, inside the network, all requests are trusted. With Zero Trust, we create micro-perimeters for each protect surface.
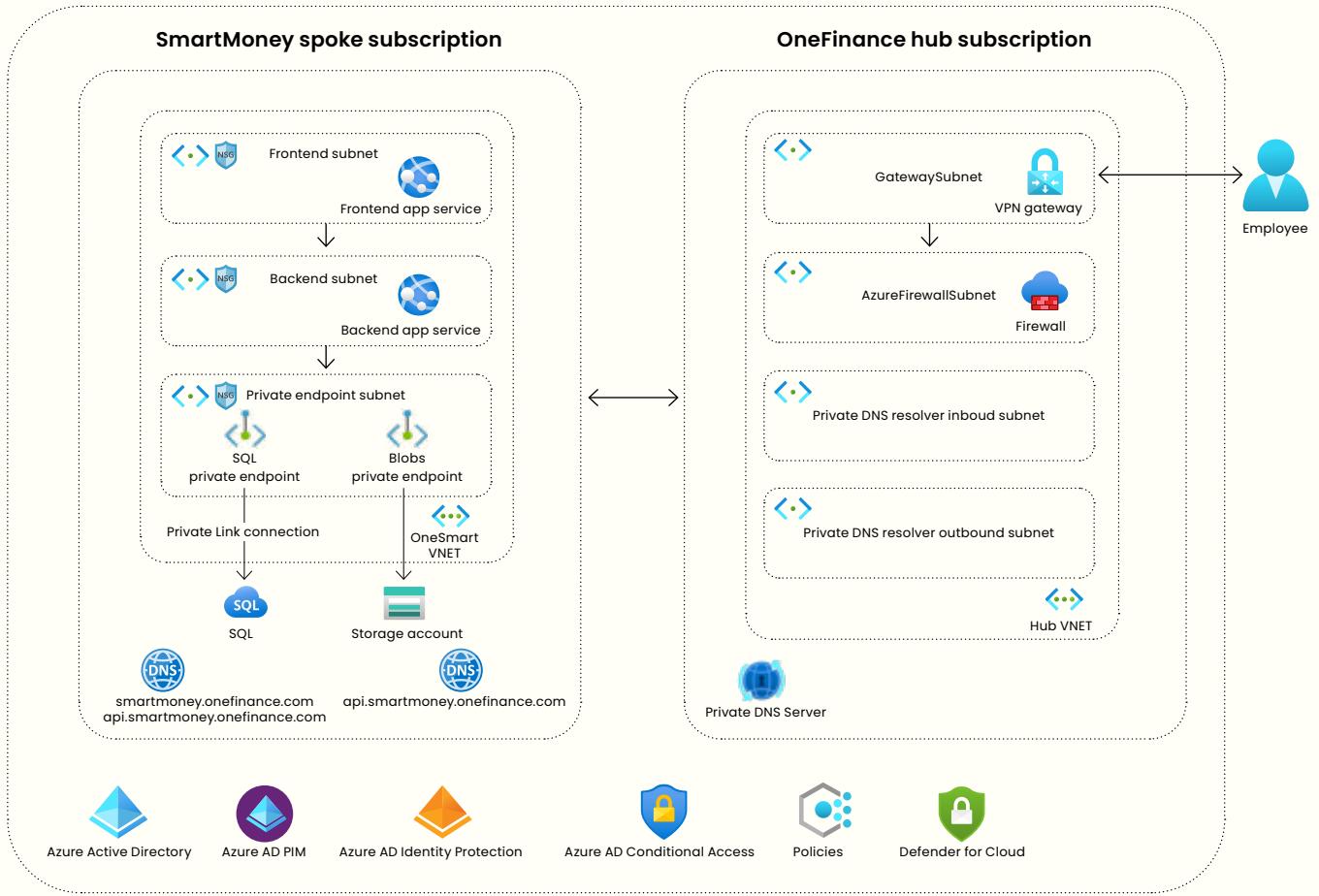
We should assume that a breach will occur at some point and that a malicious user get access to the network. An attacker could gain access through one of our applications if there is a backdoor or vulnerability in any of the third-party packages. We should isolate workloads by using network segmentation to minimize the blast radius. Each workload can be placed in its own network or subnet, and network security groups can be used to allow traffic only for specific purposes. All traffic should be denied by default.

On the next page (64), you'll find the new architecture for the SmartMoney application.. In the rest of the article, I'll guide you through the implementation and how each service provides protection based on the Zero Trust principles.

The SmartMoney application is publicly available. However, its access is limited to IP addresses from the OneFinance office and employees' homes. This approach leads to a large attack surface because anyone on the internet can potentially threaten the application. Even though enabling VPN is not necessarily a Zero Trust improvement, we want to reduce the attack surface as much as possible. By enabling VPN, we ensure that the application is only privately available. However, we should assume that at some point, a malicious user gains VPN access and is inside the network. For this reason, the workload should be created in an isolated VNET and preferably divided into multiple subnets. The architecture shows that the frontend application is in a separate subnet from the backend application. A network security group ensures that only traffic from the frontend subnet is allowed to the backend subnet. Other workloads from OneFinance, such as the ERP, run in their own VNET. No traffic is allowed between the ERP and the SmartMoney VNET.

In the new architecture, I follow the hub-spoke model, a commonly used architectural pattern. The hub is a central point for connectivity, and all inbound and outbound traffic flows through it. The firewall in the hub monitors and restricts traffic. Spokes can reuse services placed in the hub.

New architecture

Zero trust principles
- **Verify explicitly** Using network security groups, we can filter all traffic in the network. Security rules allow us to allow or block inbound and outbound traffic for specific IP addresses and ports.
- **Least privilege** Network security group - service tags help ensure that only AppServices in the frontend subnet can communicate with an AppService in the backend subnet.
- **Assume breach** Each workload runs in an isolated network or subnet.

SECURITY IS A SHARED RESPONSIBILITY BETWEEN CLOUD PROVIDERS AND THEIR CUSTOMERS

By default, PaaS services in Azure are publicly available. The SmartMoney application utilizes a storage account and SQL database for storing data. Although Azure provides secure services, it is important not to assume that just by using Azure, your workloads are secured. Microsoft explicitly states that security is a shared responsibility between Azure and the customer.

Azure offers many options to secure your storage account. However, your storage account may remain unsecured if you fail to make your containers private or use Shared Keys over Azure AD to authenticate. If your PaaS service is only used by resources in your VNET, it is recommended that you use Private Link. Private Link ensures traffic flows over the Microsoft backbone instead of the internet. A private endpoint and private DNS zone are created when enabling Private Link. When the backend application connects to the storage account, Azure detects that Private Link is enabled. The private endpoint is now used to communicate with the storage account.

In the architecture of SmartMoney, a new subnet is created for private endpoints of the storage account and SQL server. Network security groups are in place to only allow traffic from the backend subnet to the private endpoint subnet. As a result, resources inside the frontend subnet cannot reach those services directly.

Zero trust principles

- **Verify explicitly** By enabling private link, we ensure that only traffic from within the network can access the PaaS services.
- **Assume breach** If a vulnerability in the front-end application is exploited, the malicious user will not have access to the storage account or SQL server.

We have implemented a micro-perimeter for the Smart-Money application by placing it inside an isolated network. This will minimize the blast radius in case of a breach. The different components of the application are divided into subnets, and traffic is explicitly verified using network security groups. This was the first step in protecting our application. The next step is to secure the data.

Know your Data and secure it

Data is the foundation of everything we do. Some of the largest companies rely on data to generate revenue. However, data theft or a ransomware attack can cause significant damage to these companies and their end users. The first step in securing data is to discover and identify the data you have. Once this is done, classify the data with a sensitivity label so that you know which data requires a higher security level than others.

IF YOU DON'T KNOW YOUR DATA YOU CAN'T PROPERLY SECURE IT

Below is an example of data classification for the SmartMoney application.

- **Highly confidential** Customer's personal and financial data.
- **Confidential** Advice for customers based on their data.
- **General** SmartMoney manuals for new employees.
- **Public** Marketing text for the SmartMoney application.

Based on the sensitivity label, we can identify the impact of a data breach and data losses. Data discovery doesn't have to be a manual task. Azure SQL includes Data Discovery & Classifications, a feature that automatically scans your database to identify columns containing sensitive data. It also monitors and audits query results, labeling them with a sensitivity label. Based on the recommendations, you should take action to secure this data.

By default, all data stored by Azure is encrypted at rest. Platform keys are used to encrypt the data, but it's also possible to use customer keys (BYOK). You might think good, that means I'm protected against data breaches? Well no, encryption at rest means that your data is protected if an intruder gets access to a data center and steals the drive that holds your data. The data on the disk is encrypted, so useless for the intruder. Whenever you access data on the disk, for example, when using a storage account, the data is decrypted so you can use it. From this point, it's your responsibility to protect the data. One of the first things we can do is encrypt data in transit. In the SmartMoney application, we want to ensure that both the frontend and backend applications run on HTTPS. This protects us from the man-in-the-middle attack. TLS is enabled for Azure storage accounts by default; it's impossible to turn this off.

Encrypting data at rest and transit is the bare minimum we should always do. In some cases, especially with financial data, adding additional restrictions to sensitive information is essential. Assuming a breach means considering worst-case scenarios, such as what a malicious user could do with leaked credit card information.

The SmartMoney application allows employees to import bank transactions from customers. However, some data, such as the account number, should never be visible to customer service representatives. One way to achieve this is by using dynamic masking in Azure SQL. This feature automatically masks data when it is retrieved through a query. For example, a credit card number would appear as XXXX-XXXX-XXXX-1234 when masked.

If attackers gain access to database credentials, they can compromise the entire database. This would allow the attacker to create a backup or use SQL Management Studio to access sensitive data. To address this issue, Azure SQL offers the Always Encrypted feature. With this feature, data is encrypted at the client using a database driver and then stored in the database. The data can only be viewed in plain text by the client application. The data remains encrypted even if an admin accesses the database using SQL Management Studio.

Data is encrypted using a Content Encryption Key (CEK), which is stored in the database after being encrypted with a Customer Master Key (CMK). Typically, the CMK is stored in Azure Key Vault. By using this approach, sensitive data is protected in the event of SQL credentials being compromised.

Client application   SQL client driver

Data in plain text   *Data encrypted*

Zero trust principles

- **Assume breach** By using Always Encrypted, a malicious user doesn't have access to sensitive data. Only the client application can decrypt the data.

Ransomware attacks can cause significant damage to an organization. An attacker gains access to the network or PaaS service and encrypts all data, making it inaccessible to the organization. The only way for an organization to regain access to the data is to pay the ransomware to the hackers. Often, hackers announce a successful attack to the public to put more pressure on the organization to pay the ransom. Unfortunately, many organizations start worrying about ransomware threats when it's too late.

When following the Zero Trust approach, we should assume a breach from the start of a project. This means acknowledging the potential for a ransomware attack at any point. Therefore, we must protect our data against such an event. As mentioned earlier, many companies rely on their data, so protecting data should be their top priority. One way to secure data is by taking backups. SmartMoney manages many documents for customers in a storage account. Through Azure Backup Vault, we can take backups from the blob storage. There are two approaches to taking backups: operational and vaulted. Operational backups are a local solution, meaning data is stored locally on the storage account. This protects data from accidental deletion and corruption. With vaulted backups, the data is moved and protected in the vault. Usually, with a ransomware attack, the hacker will try to find backups and make them unusable. Vaulted backups are stored elsewhere and therefore protect you from ransomware attacks.

Zero trust principles

- **Verify explicitly** Only user accounts with high privileges can access the vaulted backups.
- **Assume breach** Using Vaulted backups protects us from ransomware attacks.

Securing data is essential for organizations like OneFinance. One of the main principles of Zero Trust is to assume a breach and keep it in mind from the start of the project.
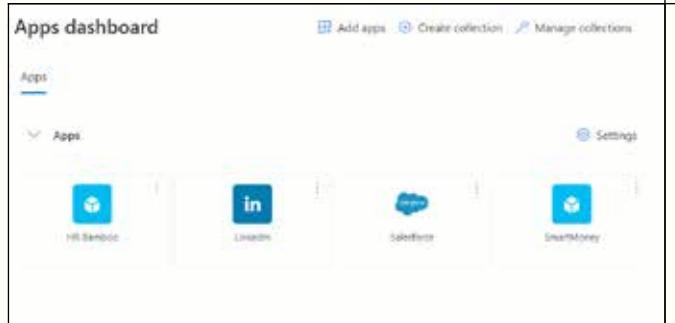
In Azure SQL, we protected our sensitive data using data masks and Always Encrypted. In addition, we used the Backup Vault to protect data in the Azure storage account from ransomware attacks. Up next is identity.

Verify and secure identities

One of the goals of Zero Trust is to eliminate trust. In the past, we would place a firewall in front of our network and implicitly trust all users within. However, with Zero Trust, we should trust nobody, whether they are inside or outside the network. All operations performed by an identity should be verified to ensure that access is appropriate for that identity.

In Microsoft Azure, all identities are centrally stored in Azure Active Directory. OneFinance has several applications, such as SmartMoney, HR system, ERP, and the intranet. To access each application, users need to authenticate. We should use Single Sign-On (SSO) whenever possible to allow users to use their same identity across applications. This approach makes identities easier to maintain, reduces the security risks of lost passwords, and provides a better user experience.

Azure Active Directory supports OpenID Connect, OAuth, and SAML for implementing SSO. Users can view and access applications they have been granted through the URL https://myapps.microsoft.com.

Benefits of using SSO

- One identity for all applications
- Withdraw access from one central place and apply for all applications
- Enforce strong authentication across all applications

The Zero Trust model requires verification of all external and internal requests to ensure security. Insider threats or social engineering attacks can lead to the exposure of employee credentials. We should implement strong authentication by enabling Multi-Factor Authentication (MFA) to prevent a malicious user from using compromised credentials.

Accounts are more than 99.9% less likely to be compromised if you use MFA.

Multi-Factor Authentication (MFA) can be enforced for identities in Active Directory. This requires users to provide an additional form of identification. Verification methods include Microsoft Authenticated App, FIDO2 security key, SMS, and Voice call. MFA can be enabled through Security Defaults when using Azure AD free or standalone Microsoft 365 license or with Conditional Access when you have an Azure AD Premium or Microsoft 365 Business. It can be enabled for specific users or groups. It's important to exclude your break-glass account. This special high-privilege account can be used in an emergency, and you want to prevent it from being locked out.

After the COVID pandemic, many OneFinance employees continue to work from home. SmartMoney stores a lot of sensitive data, and the company must avoid this data falling into the wrong hands. Therefore, we want to enforce MFA when employees try to log in to the application from outside the office. Conditional access policies enable us to select which users, devices, cloud applications, and locations require Multi-Factor Authentication.

Zero trust principles
- **Verify explicitly** Enabling MFA provides greater certainty that users are who they claim to be.
- **Assume breach** If user credentials are compromised the malicious user can't use them because MFA is enabled.

Patrick van Kleef
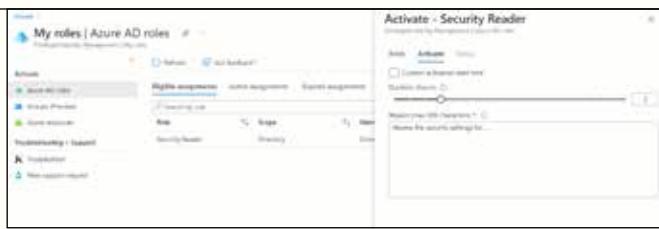
Using the least privileged access is one of the main principles of Zero Trust. With Privileged Identity Management, you can provide time-based and approval-based access. Users only get access to complete a specific task using least privilege access, eliminating sweeping access. For instance, if a user needs access to a storage account to read files in a particular container, we only give them (or their group) read access to the container, not the entire storage account.

When implementing the Zero Trust security model, you must assume a breach. Let's imagine that a user account was compromised, and the intruder used that account to access the storage account. If the least privileged access principle wasn't followed, the intruder would have access to the entire storage account, with all its consequences. However, by only assigning access rights to complete specific tasks, the blast radius of the breach would be minimal.

The customer service team manages financial information from customers in the SmartMoney application. They create new records, update data, and remove irrelevant information. The financial expert team can utilize the reporting feature to analyze this data and provide personalized advice to each client. The customer team should only have permission to create, modify, and delete records, while financial experts should only have permission to access the reporting feature.

Occasionally, customer data may need to be removed in SmartMoney, but it's important to limit who can do so. PIM allows for the implementation of Just-in-Time (JIT) access. Instead of granting permanent access to an identity, an identity can be made eligible for a role. If a user needs access, they must activate the assignment and provide justification for why they need access. They also choose how long they need this role. A manager must approve the assignment. Once the time period has expired, the assignment is automatically removed. To use PIM, an Active Directory P2 license is required.

Implementing Just-in-Time (JIT) and Just-Enough-Access (JEA) provides greater control over who has access to what and when. Typically, this is configured only once.

However, employees may receive promotions or switch departments, so scheduling access reviews regularly in Privileged Identity Management (PIM) is essential. We can check if access rights are appropriate for each user through these reviews.

Zero trust principles
- **Least privilege** Enabling JIT (Just-in-Time) and JEA (Just-Enough-Access) with PIM ensures that users only have access to complete a specific task for a short period of time.

From January 2021 through December 2021, we've blocked more than 25.6 billion Azure AD brute force authentication attacks - Microsoft

The number of login events can be massive, with users logging in to access applications every day and sometimes multiple times per day. This can result in millions of login events. Azure Identity Protection allows us to monitor user sign-in patterns and detect risks such as anonymous IP addresses, atypical travel, new countries, malware-linked IP addresses, unfamiliar sign-in properties, leaked credentials, or password spray.

With Identity Protection, you can enable a user risk policy to detect compromised accounts or a sign-in risk policy to detect unusual behavior. However, the number of signals can be overwhelming, and removing false positives can be difficult. It's possible to automate the response to risk detections, such as enforcing multi-factor authentication (MFA) when a sign-in risk is detected. For example, if a user signs in from a different country, we can ask that user for an additional authentication step by enforcing MFA. This allows users to self-remediate detected risks and stay productive without overwhelming administrators with sign-in issues. In case of emergencies, a break glass account can be used, but this can also lead to detected risks. Fortunately, excluding users like the break glass account from the risks policy is possible. Administrators can view sign-in, and user risk reports in the Azure portal. To remove false positives, it's possible to remove IP address ranges or countries from the detected signals. To use Identity Protection, you need an Azure AD P2 license.

Securing identities is one of the most critical defense areas of Zero Trust. Identities have access to applications and sensitive data. We have enabled single sign-on (SSO) and enforced Multi-Factor Authentication (MFA) by using conditional access. Privileged identity management (PIM) ensures that we follow the least privilege access principle of Zero Trust.

Misconfigurations are the leading cause of data breaches. In the next section, I will discuss how to prevent and detect security risks in our infrastructure.

**Use signals to protect your infrastructure**

When implementing the Zero Trust model, we aim to monitor all traffic going to and from a protect surface and remediate risks. We should not view security as a one-time project but rather an iterative process. All traffic is logged, and based on these logs, we can enhance security to become more robust. Introducing new workloads can create new security risks, so consistently monitoring the entire infrastructure for risks is essential.

When developing solutions in the cloud, the number of signals that are collected can be overwhelming. It is impossible to monitor all of these signals and remediate risks manually. Azure provides Defender for Cloud, which identifies and remediates risks across subscriptions. Defender has capabilities for cloud security posture management (CSPM) and cloud workload protection platform (CWPP). It constantly scans subscriptions and resources for security issues. A security score is provided based on identified security risks and recommendations. The higher the score, the better your cloud environment is secured. CWPP protects workloads from threats. Defender provides plans for servers, containers, databases, and storage. Configured workloads are scanned, and risks are reported based on their security level.

Defender for Storage analyzes the telemetry generated by blob storage. Based on this data, alerts are triggered. Telemetry includes operations on blobs such as create, update, and delete. This doesn't impact the performance of the storage account. Detected risks include unusual access to an account, malicious content uploads, data encryption, unusual data extraction, etc. Azure uses a technique called reputation analysis to detect malware. It means that files are hashed, and based on that, the likelihood is calculated if that hash is malware. When risks occur, alerts are triggered. It is essential to investigate alerts and check for any false positives. For instance, an alert could be triggered if a lot of data is downloaded at once. However, this might be an employee with a valid reason.

THROUGH 2022 AT LEAST 95 PERCENT OF CLOUD SECURITY FAILURES WILL BE THE CUSTOMER'S FAULT.

Azure uses the blob.core.windows.net endpoint to create blob storage accounts. Publicly available storage accounts are easy to discover; simply search for "site:*.blob.core.windows.net" on Google and you will get over 3 million results. A sophisticated hacker can easily write a script to find publicly available storage accounts, query for the containers and blobs, and find valuable data. From there, the hacker can discover the company and target them to gain access to the access keys. Once they obtain access, they have an entry point into the organization and can infiltrate further by uploading malware. Defender for Storage detects publicly available storage accounts and fires an alert. This is great because we can remediate that risk immediately.

This provides us with great information about our workloads in real time. However, prevention is better than detection. We should detect any misconfigurations before deploying them to production.

Define the desired state of your infrastructure using code and scan for misconfigurations.

Using Infrastructure as Code (IaC) has many advantages, such as reducing the risk of human error, ensuring consistency, enabling automation, and saving costs. By implementing a good DevOps pipeline that includes pull requests and reviews, faults can be detected early. Terraform is an open-source IaC tool that works with many cloud services. The desired state of the infrastructure is written in code and can be easily deployed. Because Terraform is open-source, many extensions are available for use. Security tools like tfsec and Checkov can scan Terraform code for misconfigurations and security issues, allowing us to provide engineers with feedback early in the DevOps pipeline.

When we design a Zero Trust architecture, we want to do this on an organizational level. The standards that are defined apply to all workloads. With Azure Policies, we can govern our Azure resources from one central place. For instance, using access keys is a potential risk because anyone can use those access keys to access the storage account. To ensure that storage accounts don't use access keys, we can activate the policy: "Storage accounts should prevent shared key access". We can deny or audit the resource depending on how the policy is configured. When creating a new policy, you first want to use the audit option to test which resources are affected. If you start using deny, you potentially could block teams.

For the SmartMoney application, we have already enabled Private Link for the storage account to ensure that the storage account is only reachable in the VNET. OneFinance has many workloads, and some are using storage accounts. To ensure that all storage accounts created in subscriptions use Private Link, we can use the built-in policy 'Configure Storage account to use a private link connection'. This policy will automatically configure Private Link if it's not deployed for a storage account. Policies can be created at the management group, subscription, and resource group level. OneFinance wants to enable it for storage accounts in all subscriptions, so the best place to create this policy is at the management group level.

Misconfigurations are the leading cause of data breaches. To prevent this, it is recommended to use a defense-in-depth approach by creating multiple layers of security.

It's a best practice to implement both a prevention and detection mechanism.

1. Prevention - Declaratively define infrastructure using Terraform.
2. Prevention - Use tools such as tfsec and Checkov to detect risks and security improvements early in the DevOps pipeline.
3. Prevention - Use pull requests and reviews.
4. Prevention/ Detection - Azure policies help align infrastructure with your organization's policies.
5. Detection - Defender for Cloud to analyze signals, detect, and remediate risks.

Closing words

The world has changed, and we must accept that bad actors are everywhere. Dynamic work environments, smart devices, and increasingly sophisticated cybercriminals increase the attack surface. To protect ourselves, we need a new security approach: Zero Trust. This approach eliminates trust and assumes breach.

With Zero Trust, we don't trust anyone, not even our employees. In this article, I explained the main principles of Zero Trust, why we need it, and how to implement it. Using an example application, I demonstrated how to follow Zero Trust principles using services in Azure. Note that this was not a comprehensive list of all Azure services and features you can use, but it should give you an idea of why we should use them.

</>

READ MORE ONLINE

Preventing Identity Crisis in Azure

As organizations move more and more operations to the cloud, ensuring these operations run securely is crucial. We use hardware tokens, complex passwords, One-Time-Passwords, and authenticator apps to authenticate human accounts. The question remains: how do we securely do this with system accounts? In this article, I'll walk you through the options and give examples of how to use each option best.

Author Loek Duys

The Principle of Least-Privilege

Before diving into more details, knowing about the least-privilege principle is essential. The principle of least privilege is an essential aspect of security in the cloud. It involves granting the minimum level of access necessary to perform a specific task. Minimizing permissions helps reduce the risk of security breaches and unauthorized access to sensitive information.

System accounts in Azure

Regarding system accounts in Azure, applying the principle of least privilege requires careful consideration of the level of access each account requires. For example, you may have a system account that only needs access to a specific subset of resources, such as read-only access to a database. In this case, granting full administrative access to the account would be unnecessary and increase the risk of security breaches.

Three types of Identity in Azure

To authenticate services to other services running inside Azure, you can choose from various options, such as Service Principals, Managed Identities, and Federated Identities. Each has benefits and drawbacks.

Service Principals

Using a Service Principal was the earliest method to authenticate systems to Azure Active Directory (AAD). A Service Principal is an identity created for use with applications, services, and automation tools to access specific Azure resources. You use them to authenticate and authorize applications to access specific Azure resources. Service Principals are similar to user accounts, but you use them for non-interactive scenarios. To authenticate using a Service Principal, you must provide the Client's Identifier and a Client Secret or Client Certificate. Both passwords and certificates have an expiration date, so your authenticating system needs to be able to deal with secret rollovers. The authenticating system does not need to run inside Azure; it can run anywhere as long as internet access is available. You can see the way this works in Figure 1.
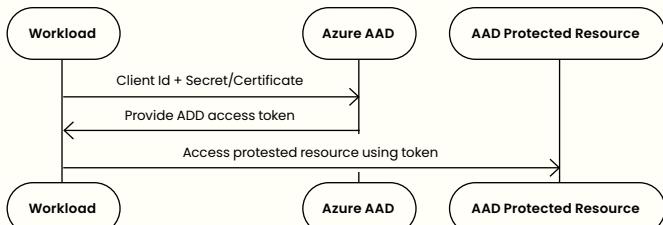

Figure 1: Using a Service Principal

When to use

Use this approach when you have complete control over the system requesting access. For example, Service Principals work very well when creating resources in Azure using a GitHub Actions pipeline. GitHub Actions has built-in functionality to pass the Client Secret of your Service Principal to the tasks that create the resources. You have complete control over both systems, making this a viable option. Make sure to assign the proper rights to the Service Principal, for example, by assigning it the Azure Role Based Access Control (RBAC) role 'Contributor' at the scope of a resource group or (at most) subscription. Having the Contributor role will allow the Principal to create resources but not access the data stored inside the resources, nor does it allow the assignment of roles.

Managed Identities

A Managed Identity is a Service Principal managed by Azure. You can use it to authenticate specific Azure resources to access other Azure resources. The main difference with regular Service Principals is that you don't need to store credentials to use them; Azure manages this and secret rollover for you. The downside of Managed Identity is the authenticating system must run inside Azure to get a Managed Identity assigned.

When to use

Managed Identities can only be assigned to Azure resources, limiting their use to Cloud services running within Azure. In my opinion, you should rely on Managed Identities as much as possible for authentication between Azure services. For example, you can configure a Managed Identity with an Azure Web App and allow it access to an Azure SQL Database, as displayed in Figure 2.
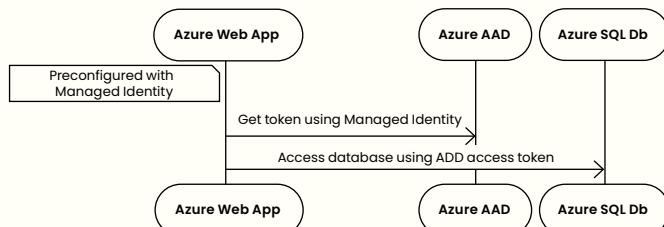

Figure 2: Example of Managed Identity

Similarly, you can use Managed Identity to allow your Web App to access services like Storage Accounts, Key Vaults, Service Bus, etc. Most popular Azure services support Managed Identity nowadays.

The Managed Identity connected to the Web App must be allowed to access the Azure SQL Database. As with Service Principals, you can do this using Azure RBAC. In this case, the Principal needs to be allowed to access data but not to

modify the resource itself. You can do this by creating a SQL user inside the database to log in using the Managed Identity. Next, you need to assign database-specific roles to it. This way, the Managed Identity can only be used for a single purpose, with minimal privileges.

Federated Identities

A Federated identity in Azure is authenticated to AAD using and trusting an external identity provider (IDP). You first authenticate a system with the external IDP and gain access to Azure resources through Identity Federation. When two IDPs are federated, one IDP trusts tokens issued by the other.

A Federated identity allows you to access Azure resources running outside Azure without needing Service Principal credentials. Because you are using an external IDP, you can also control the way it issues access & identity tokens, which opens up interesting scenarios. For example, you could let a locally running containerized IDP issue a token for a Service Account in Kubernetes. You can configure Azure AD to trust the containerized IDP and use its token to authenticate a Service Principal. It turns out this scenario already exists under the name' Workload Identity'. You can see how it works in Figure 3.
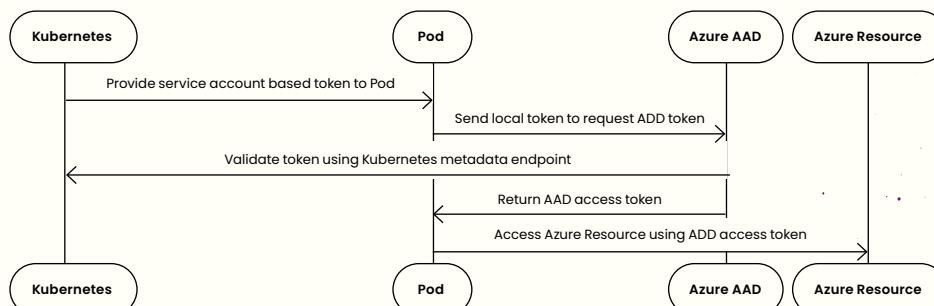
Figure 3: Example of federated Identity

When to use

Using federated Identity works well when running workloads in Azure Kubernetes Service. But the mechanism also works outside of Azure. It doesn't matter where your code runs as long as you have correctly configured Azure Active Directory, and AAD can reach your local IDP's metadata endpoint over the public internet. Federated Identity can also work when allowing systems you do not own to access your AAD-controlled resources. Unfortunately, Microsoft does not allow the remote IDP to be Azure Active Directory. Other than that, you can configure trusts to any remote OAuth-compliant IDP. Imagine you have a business partner who wants to access your Web API, protected by AAD, from one of their systems. They protect their systems using Duende's Identity Server platform. Instead of providing them with (expiring) credentials of a Service Principal, you could set up Federated Identity as described in Figure 4.
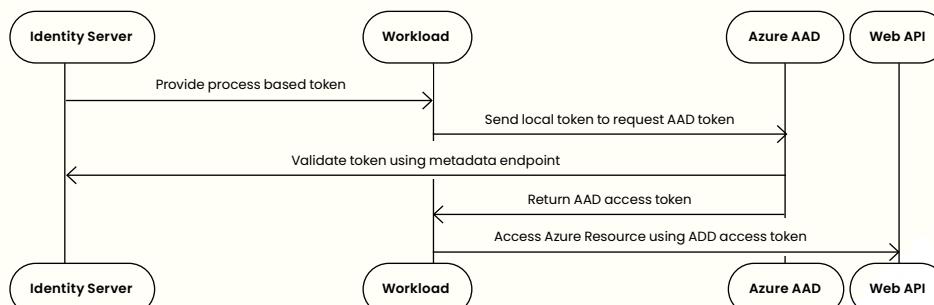
Figure 4: Business federation example

Conclusion

The move to the cloud brings a new set of security challenges. However, by understanding the various options available in Microsoft Azure and Azure Active Directory, you can secure your workloads and prevent an identity crisis. Whether you are using Service Principals, Managed Identities, or Federated & Workload Identity, applying the least-privilege principle to reduce the risk of security breaches and unauthorized access to sensitive information is essential.
‹/›

Loek Duys

Ten tips and tricks to secure your Azure subscription

Creating a new Azure subscription can be done in a few clicks but adopting the cloud in your organization takes more time and effort. At Xpirit, we help our customers on their cloud journey and one of the important factors is to make sure that your Azure environment is secure when migrating your workloads. In this article, we list ten tips and tricks that are a good starting point to make sure you can benefit from the possibilities of the cloud in a secure way.

Author Laurenz Ovaere

1. Protect your Azure Active Directory account with MFA

Azure Active Directory (Azure AD or AAD for short) is the identity provider for Azure and takes care of the authentication and authorization scenarios. Every time you access the Azure platform you will need to pass the AAD authentication in order to prove that you are really the person you pretend to be. A common way to do this is by providing your username and password. However, research shows that it is a lot safer to combine this with multi-factor authentication (MFA). With MFA you must confirm your authentication in a mobile app before access is granted to your account.

MFA based on a notification in the Microsoft Authenticator app is a free option available to any Azure AD user including the free tier. Higher AD tiers can also choose to receive a phone call, a text message or use a hardware token as extra verification.

Enabling MFA is no longer enough to protect your environment. The Microsoft Authenticator app evolves over time in order to avoid new types of attacks such as MFA bombing. In this type of attack, users are overloaded by approval requests from an attacker in the hope they click approve. The number matching feature is one example that avoids incorrect approvals by requesting a matching number shown on the logon screen of the user and will be enforced later this year. Showing additional context information to the user like the origin of the request is a second example that can further improve the security. This option needs to be enabled as an Authentication method policy in Azure AD.

2. Enable conditional access or make sure you enable the security defaults on your free Azure AD tenant

The Azure AD security defaults are a set of rules that can be used to get started with a preconfigured set of security settings in Azure AD. These are available to all Azure AD tiers including the free tier. In fact, you need to enable the security defaults to enforce MFA setup on the Azure AD free tier. This will also block legacy authentication protocols that are allowed to bypass multifactor authentication. With the security defaults enabled, you have no control when the MFA is prompted to the user. The MFA setup is enforced but Azure AD will decide automatically based on signals like location, device, role and task if a prompt is appropriate.

In case you want more granular control over when MFA is prompted to the user, you need to use conditional access, which is available in the premium Azure AD tiers. This will also give you the ability to block access from certain non-authorized locations, block risky sign-in behavior or block access from non-compliant devices. These are all examples of conditions you can define for certain users or groups before they get access. Enabling conditional access can improve your setup in two ways. First, you can be stricter before you grant access, which limits the attack surface of your environment. If you have no access to your Azure environment from a non-compliant device neither will an attacker. Second, you can make a distinction between types of access and tighten the authentication requirements for administrators compared to standard users. This will increase your security without too much impact on all your users.

3. Use just-in-time access for tasks that require higher privileges

After successful authentication, Azure role-based access control (RBAC) is the authorization system that provides fine-grained access control to Azure resources. The roles assigned to your user profile define which actions you can take with a certain resource. Administrators typically have extra roles assigned to their profile to take privileged actions to Azure resources. The problem is these role assignments are permanent and so could be used by attackers that have access to your account or open doors for accidents by yourself.

Privileged Identity Management (PIM) is an Azure AD premium feature available in the P2-tier that fixes these problems. Azure AD roles with higher privileges are no longer assigned permanently with PIM but just-in-time after a request by the user and for a limited duration. This method comes with a lot of benefits. For example, you can require extra approvals by other members before the access is granted, you can enforce an extra MFA prompt or ask the user to enter a reason why they need this access for the time duration requested. The last one results in a clear audit history for your environment.

4. Use managed identities where possible

Beside users, we also have services that need access to our Azure resources. For example, an Azure Web App that writes data to an Azure SQL database, an Azure Function App that reads messages from Azure Service Bus, etc. Both use cases can be solved by sharing secrets with our services. However, when relying on secrets we also need to manage them and store them securely. This is where managed identities for Azure comes in.

With Managed identities for Azure, our services have an identity assigned in Azure AD. We can assign specific roles to that identity just like we do for users with Azure role-based access control (RBAC). The service itself will use its identity in the background to obtain an Azure AD token that can be used to access the requested service. The advantages are that there are no secrets to manage, it is more secure and free to use.

5. Store secrets, keys and certificates in Azure Key Vault

After implementing Managed identities for Azure you won't have a lot of secrets anymore to manage. However, there are two scenarios that still require key management. First, services that do not integrate with Azure AD still use secrets, keys or certificates that you need to store in a safe place. Azure Key Vault is a well-known Azure service desig-ned to store these in a safe way.

Second, when you choose to manage the encryption keys for Azure Disk Encryption yourself with Customer Managed Keys (CMK), you will also need to store these keys in Key Vault. In contrast, Platform Managed Keys (PMK) are managed by Azure in the background but provide less flexibility.

Combining both managed identities and Azure Key Vault makes the solution even better. An Azure AD integrated service can use managed identity to access Azure Key Vault

and request a secret, key or certificate. That way our Key Vault is the single dedicated place for secret information and we can use Azure role-based access control (RBAC) to manage the access rights.

6. Organize your Azure resources effectively to improve your access and policy management

When the number of resources grows, it becomes very hard to control your role-based access control lists or Azure policies at the resource level. However, to manage access on a higher level while still applying the principle of least privilege, you need to combine resources with the same access rights. Azure resources can be organized on four different levels: management groups, subscriptions, resource groups and the resources itself. An effective organization makes it easier to manage, track costs and secure your resources.

There is no generic way of organizing resources that works for everyone but here are some rules of thumb to get you started. For application development, production and non-production resources are typically managed in different subscriptions to make a clear boundary between both and ease the access and policy management. Resources owned by separate teams can be stored in different subscriptions. And lastly, management groups can be used to group subscriptions that belong to the same department and share some common access rights or policies.

7. Make private networking the default for your Azure resources

When creating resources in Azure, most of the time they are exposed to the public internet. For example, an Azure Web App, a storage account or a virtual machine are all accessible over the internet without further modifications. There are situations where you want this public exposure. Think about your company's public website. But in general, it is better to start with private networking and explicitly expose certain endpoints. By marking these endpoints as public explicitly, you can also improve the protection if required. Adding Azure Application Gateway or Azure Front Door for example can give you Web Application Firewall (WAF) capabilities to further improve the security of these endpoints.

Marking endpoints as public or private is not as easy as it sounds. It all starts with a good network design. The hub and spoke topology discussed as part of the Azure landing zones in our previous magazine is a good starting point. The hub in the virtual network can be connected to an on-premise network using a VPN or ExpressRoute connection. Ensuring a safe and private connection. Furthermore, the use of network security groups, private endpoints, private DNS and VNET integration will make sure your services use private networking to connect to each other instead of the public internet. Note that some of these features are only available in the higher pricing tiers. Part of our services at Xpirit is to help you finding the right balance between costs and features.

8. Protect your data with encryption

All the previous tips and tricks are here to avoid access of unauthorized users to your Azure environment. Another extra way to protect your environment is to make sure your data is not readable to a potential attacker due to encryption.

At first, we can make use of encryption in transit. This means we will encrypt data before sending it over the network. The typical example here is TLS encryption used with HTTPS. By disabling HTTP endpoints or redirecting traffic to HTTPS, we ensure our data is unreadable to attackers.

Second, we also have encryption at rest. Azure Storage provides automatic server-side encryption to storage accounts. This makes the data unreadable to unauthorized users. Virtual machines running Windows or Linux can also benefit from encryption with Azure Disks. This feature lets you encrypt both the data and OS disks.

9. Make sure you apply pending updates

Applying updates is an important part of security and when choosing a cloud platform like Azure, some updates are done by Microsoft. However, a cloud platform comes with shared responsibilities and there are parts of your environment that you are responsible for and that you need to update regularly. It is important to know your responsibilities before you use a certain service. I will give some examples below.

For Platform-as-a-Service (PaaS) products, as the name suggests the platform is managed by Azure. For example, you don't need to apply security patches to the .NET runtime on an Azure App Service. However, applying package updates to your software is an example of components you are responsible for. A tool like Dependabot can inform you about pending package updates. More on this in our previous edition of our magazine.

Virtual machines are Infrastructure-as-a-Service (IaaS) products that require more maintenance from your side. The hardware components like CPU, memory, disks, cooling, etc. are managed by Azure but you are responsible for the installed software. Azure Automation can already help you with the update management of your operating system, but the other software patches require extra action from your side.

10. Monitor your environment and improve continuously

At last, even with all security measures in place it remains important to monitor your Azure environment and improve continuously where needed. Microsoft Defender for Cloud (formerly known as Azure Security Center) integrates different security monitoring, compliance checks and alerts in one single dashboard. This makes it a good place to look for potential improvements to your environment.
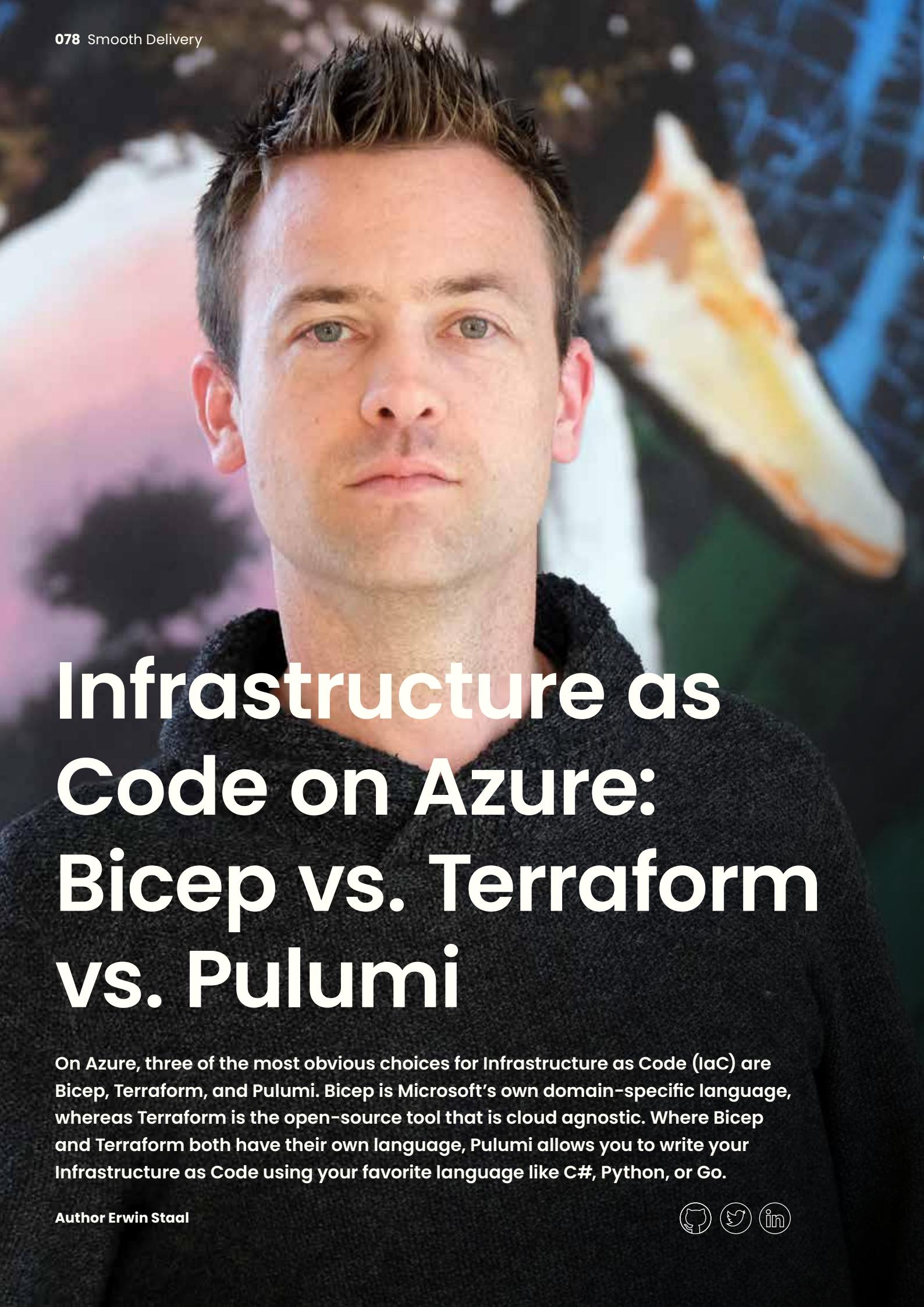
Conclusion

Hopefully these ten tips and tricks can get you started on your cloud journey in a secure way. Please reach out if you have questions about the security of your Azure environment. Our Cloud Security Scan offering can evaluate the current state of your environment and define potential security optimizations.
</>

READ MORE ONLINE

Infrastructure as Code on Azure: Bicep vs. Terraform vs. Pulumi

On Azure, three of the most obvious choices for Infrastructure as Code (IaC) are Bicep, Terraform, and Pulumi. Bicep is Microsoft's own domain-specific language, whereas Terraform is the open-source tool that is cloud agnostic. Where Bicep and Terraform both have their own language, Pulumi allows you to write your Infrastructure as Code using your favorite language like C#, Python, or Go.

Author Erwin Staal

The goal of this article is to compare the three tools and leave you with enough understanding of each of them to make a good choice when it comes to an Infrastructure as Code tool for your next project. We will, of course, talk about the features of each of these tools but, more importantly, will also compare the developer experience with each of them. Let's dive in and see who will be the last one standing and who will throw in the towel first.

Introduction
Before we start comparing the three contestants of today, let them first all properly introduce themselves.

Bicep
Bicep is a domain specific language (DSL) created by Microsoft. It is their answer to the problems we had with its predecessor, ARM templates. ARM templates have been around for many, many years. Describing your IaC in ARM templates was done using JSON. That made ARM templates very verbose, hard to read, and even harder to maintain. Splitting up a template into multiple smaller, reusable components was way too hard. Bicep is here to solve those problems and it really does make our life easier. Unlike the other two tools in this contest, Bicep can only be used to configure resources on Azure. More on that later. A very simple Bicep template looks like this:

```bicep
@description('The name of the storage account')
param name string

@description('Azure region of the deployment')
param location string

@allowed([
  'Standard_LRS'
  'Premium_LRS'
])
@description('Storage SKU')
param storageSkuName string = 'Standard_LRS'

resource storage 'Microsoft.Storage/storage-
Accounts@2021-09-01' = {
  name: name
  location: location
  sku: {
    name: storageSkuName
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
    allowBlobPublicAccess: false
    allowCrossTenantReplication: false
    allowSharedKeyAccess: true
    minimumTlsVersion: 'TLS1_2'
    supportsHttpsTrafficOnly: true
  }
}

output storageId string = storage.id
```

The above example deploys a storage account on Azure. It gets a few inputs, specifies the resource, and returns an output that can be used by other templates or scripts.

Terraform
Terraform is an open source IaC tool owned by HashiCorp. It was created in 2014 and, like Bicep, is a DSL. A big difference between Bicep and Terraform is that Terraform can manage infrastructure on all big cloud platforms and other services. What we in Terraform call 'Providers' enable Terraform to work with virtually any platform or service with an accessible API. These providers are often created by the owner of the targeted platform but there are also numerous providers created by the community. Most of them are open source. You could even build your own if you need to. The example below creates a storage account using Terraform.

```terraform
terraform {
  required_version = ">= 1.1.7"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 3.27.0"
    }
  }
}

provider "azurerm" {
  features {}
}

variable "name" {
  type = string
}

variable "resource_group_name" {
  type = string
}

resource "azurerm_storage_account" "storage" {
  name                 = var.name
  resource_group_name  = var.resource_group_name
  location             = "westeurope"
```

```
    account_tier              = "Standard"
    account_replication_type = "LRS"
}

output "storageId" {
  value = azurerm_storage_account.storage.id
}
```

As you can see, this configuration is slightly more complex compared to the Bicep version which has the same result. In Terraform you first need to configure a provider since you could be working with one or more providers. A provider could have a specific configuration, hence the 'azurerm' provider block. Defining a variable is a bit more verbose. You can, unlike in Bicep, move them into a different file which could increase readability. The same goes for outputs, the terraform block or the provider configuration.

Pulumi

Pulumi is our third contestant of today. It was named after the company that created it in 2018. What makes Pulumi different is that it allows you to write your infrastructure as code in your favorite language: TypeScript, JavaScript, Python, Go, .NET, Java, and YAML. Like Terraform, it allows you to manage infrastructure on basically any cloud or service provider. Even if it is not supported by default, you could always easily create your own code to interact with a service provider's API. The example below creates a Storage Account in Azure using C# and Pulumi.

```csharp
using Pulumi;
using Pulumi.AzureNative.Resources;
using Pulumi.AzureNative.Storage;
using Pulumi.AzureNative.Storage.Inputs;
using System.Collections.Generic;

return await Pulumi.Deployment.RunAsync(() =>
{
  var config = new Pulumi.Config();

  // Create an Azure resource (Storage Account)
  var storageAccount = new StorageAccount(config.
  Get("storageAccountName"), new StorageAccountArgs
  {
      ResourceGroupName = config.Get("resourceGroupNa-
      me"),
      Sku = new SkuArgs
      {
          Name = SkuName.Standard_LRS
      },
      Kind = Kind.StorageV2
  });
```

```csharp
  // Export the primary key of the Storage Account
  return new Dictionary<string, object?>
  {
      ["storageId"] = storageAccount.Id
  };
});
```

The first thing you need to do here is grab the Nuget package for Azure. From there, every resource is created by instantiating an instance of a specific class and passing in a few parameters.

You might be wondering, wasn't there a version of Terraform in which I can use a programming language as well? Yes, that is true! It is called Terraform CDK (Cloud Development Kit). I have left it out for a few reasons. The most important one is that I simply have never used it. I personally don't see it used in the wild often. That might be because the product has not reached version 1.0 yet and Terraform itself warns for breaking changes. It also supports a few less languages compared to Pulumi.

Single or Multi Cloud support?

An important consideration while picking any of these tools is what platforms you are deploying onto. Two of the three tools we discuss here, Terraform and Pulumi, have multi cloud support. That means that they can interact with multiple cloud providers and services. You could, for example, create an App Service in Azure and then pass that URL to Cloudflare to register in DNS. As you see, that is not limited to the just the three big cloud vendors but includes monitoring tools, DNS providers, GitHub, Azure DevOps, and much more. A downside can be that these tools do not always support every feature of a cloud vendor or do not support a new feature on launch day. These services often rely on the community to implement features. For Terraform you need to wait for the provider to be updated. Since they are mostly open source you can do that yourself of course. Pulumi uses two types of providers, "bridged" providers which use Terraform providers to map out the different functions available for each API on the cloud provider, or "native" providers which map out the functions directly from the Cloud Provider API. With the bridged providers you obviously have the same issues as with Terraform native. For the Azure provider that is not the case since it is a native provider. It is updated every night automatically by recreating it against the Azure APIs.

To give you an idea of what a multi cloud implementation could look like in these tools, let's start with an example in Terraform. The below code will create an Azure App Service and configure a custom domain on it. The domain is registered with Cloudflare.

```terraform
resource "azurerm_service_plan" "app_service_plan" {
  name                = "asp-${var.project_name}
                        -${var.environment}"
  resource_group_name = var.resource_group_name
  location            = var.location
  os_type             = "Linux"
  sku_name            = "B1"
}


resource "azurerm_linux_web_app" "app_service" {
  name                = "app-${var.project_name}
                        -${var.environment}"
  resource_group_name  = var.resource_group_name
  location            = var.location
  service_plan_id     = azurerm_service_plan.app_
                        service_plan.id

  site_config {}
}

resource "cloudflare_record" "domain-verification" {
  zone_id = "72e0e6d795ec809b9158033c4a4c73d3"
  name    = "asuid.tf-demo.staal-it.nl"
  value   = azurerm_linux_web_app.app_service.custom_
            domain_verification_id
  type    = "TXT"
  ttl     = 3600
}

resource "cloudflare_record" "cname-record" {
  zone_id = "72e0e6d795ec809b9158033c4a4c73d3"
  name    = "tf-demo.staal-it.nl"
  value   = azurerm_linux_web_app.app_service.
            default_hostname
  type    = "CNAME"
  ttl     = 3600
}

resource "azurerm_app_service_custom_hostname_
         binding" "hostname-binding" {
  hostname            = "tf-demo.staal-it.nl"
  app_service_name    = azurerm_linux_web_app.
                        app_service.name
  resource_group_name = var.resource_group_name

  depends_on = [
    cloudflare_record.domain-verification,
    cloudflare_record.cname-record
  ]
}
```

First, the app service plan and the web app itself are created. Next, we create the two records in Cloudflare. You can distinguish the two providers we use by looking at the name of the type we create. Stuff for Azure starts with 'azurerm', stuff for Cloudflare with 'cloudflare' The last step is to set the domain on the App Service ones they exist in Cloudflare.

Using Pulumi and C#, the exact same functionality is achieved using the following code:

```csharp
using Pulumi;
using Pulumi.AzureNative.Web;
using Pulumi.AzureNative.Web.Inputs;

class AppService : Pulumi.ComponentResource
{

  [Output("AppServiceEndpoint")]
  public Output<string> AppServiceEndpoint {
  get; private set; }

  public AppService(string name, AppServiceArgs args,
  ComponentResourceOptions? opts = null)
      : base("azure:custom:appservice", name, opts)
  {
      var appServicePlan = new AppServicePlan
      ($"asp-{name}", new AppServicePlanArgs
      {
        ResourceGroupName = args.ResourceGroupName,
        Kind = "App",
        Sku = new SkuDescriptionArgs
        {
            Tier = "Basic",
            Name = "B1",
        },
      }, new Pulumi.CustomResourceOptions
      { Parent = this });

      var app = new WebApp($"app-{name}",
      new WebAppArgs
      {
        ResourceGroupName = args.ResourceGroupName,
        ServerFarmId = appServicePlan.Id
      }, new Pulumi.CustomResourceOptions
      { Parent = this });

      AppServiceEndpoint = app.DefaultHostName;

      var domainVerification = new Pulumi.
      Cloudflare.Record("domain-verification",
      new Pulumi.Cloudflare.RecordArgs
```

```
{
    Name = "asuid.pulumi-demo.staal-it.nl",
    ZoneId = "72e0e6d795ec809b9158033c4a4c73d3",
    Type = "TXT",
    Value = app.CustomDomainVerificationId,
    Ttl = 3600,
}, new Pulumi.CustomResourceOptions
{ Parent = this });

var record = new Pulumi.Cloudflare.Record
("record", new Pulumi.Cloudflare.RecordArgs
{
    Name = "pulumi-demo",
    ZoneId = "72e0e6d795ec809b9158033c4a4c73d3",
    Type = "CNAME",
    Value = app.DefaultHostName,
    Ttl = 3600,
}, new Pulumi.CustomResourceOptions
{ Parent = this });

var exampleCustomHostnameBinding =
new WebAppHostNameBinding("exampleCustom-
HostnameBinding", new()
{
    HostName = "pulumi-demo.staal-it.nl",
    Name = app.Name,
    ResourceGroupName = args.ResourceGroupName,
}, new CustomResourceOptions { DependsOn =
{ domainVerification, record }, Parent = this
});

this.RegisterOutputs();
    }
}
```

Our last contender, Bicep, is always on par with any feature available in the Azure cloud on launch day. For Bicep, that is a lot easier since it is a single cloud tool built and maintained by Microsoft itself. A big downside of this single cloud tool is that you are limited to managing your Azure infrastructure. More specifically, you can only interact with what we call the Azure control plane. Azure operations can be divided into two categories - control plane and data plane. Simply put, you use the control plane to manage resources in your subscription, you use the data plane to manage the internals of a resource. For example, Bicep allows you to create a SQL Database but does not let you create a user in that database. Bicep also does not let you interact with Active Directory. Creating an Enterprise Application and using it in your IaC is not an easy task. There are mainly two alternative approaches here; run some code in your CI/CD pipeline and feed the result to

Bicep on deploy or use the DeploymentScripts resource. The DeploymentScripts resource allows you to run an Azure CLI or PowerShell script during the execution of Bicep. We could use that to accomplish the example with Cloudflare we saw in Terraform and Pulumi. What happens under the hood when you use a DeploymentScripts resource is that an Azure Container Instance will be created, and a container will be deployed to run your script. This is slow and does not support enterprise features like network integration. This often limits the use of that resource.

Creating the App Service and records in Cloudflare using Bicep would look like this:

```
param name string
param location string
param cloudFlareToken string

var record = 'bicep-article-demo'
var domain = 'staal-it.nl'

resource appServicePlan 'Microsoft.Web/server-
farms@2019-08-01' = {
  name: 'asp-${name}'
  location: location
  sku: {
    name: 'B1'
    capacity: 1
  }
}

resource webApplication 'Microsoft.Web/sites-
@2018-11-01' = {
  name: 'app-${name}'
  location: location
  properties: {
    serverFarmId: appServicePlan.id

    siteConfig: {
      netFrameworkVersion: 'v6.0'
    }
  }
}

resource cloudflare 'Microsoft.Resources/deployment-
Scripts@2020-10-01' = {
  name: 'cloudflare'
  location: location
  kind: 'AzurePowerShell'
  properties: {
    forceUpdateTag: '1'
    azPowerShellVersion: '8.3'
```

```
arguments: '-hostname "${record}" -domain
"${domain}" -destination "${webApplication.proper-
ties.defaultHostName}"'
environmentVariables: [
  {
    name: 'CLOUDFLARE_API_TOKEN'
    secureValue: cloudFlareToken
  }
]
scriptContent: '''
  param([string] $hostname, [string] $domain,
  [string] $destination)

  $zoneid = "72e0e6d795ec809b9158033c4a4c73d3"
  $url = "https://api.cloudflare.com/client/v4/
  zones/$zoneid/dns_records"

  $addresses = (
      ("awverify.$hostname.$domain",
      "awverify.$destination"),
      ("$hostname.$domain", "$destination")
  )

  foreach($address in $addresses)
  {
      $name = $address[0]
      $content = $address[1]
      $token = $Env:CLOUDFLARE_API_TOKEN

      $existingRecord = Invoke-RestMethod
      -Method get -Uri "$url/?name=$name"
      -Headers @{
          "Authorization" = "Bearer $token"
      }

      if($existingRecord.result.Count -eq 0)
      {
          $Body = @{
              "type" = "CNAME"
              "name" = $name
              "content" = $content
              "ttl" = "120"
          }

          $Body = $Body | ConvertTo-Json -Depth 10
          $result = Invoke-RestMethod -Method Post
          -Uri $url -Headers @{ "Authorization" =
          "Bearer $token" } -Body $Body -Content-
          Type "application/json"

          Write-Output $result.result
      }
```

```
      else
      {
          Write-Output "Record already exists"
      }
  }
  '''
  supportingScriptUris: []
  timeout: 'PT30M'
  cleanupPreference: 'OnSuccess'
  retentionInterval: 'P1D'
  }
}

resource symbolicname 'Microsoft.Web/sites/
hostNameBindings@2022-03-01' = {
  name: '${record}.${domain}'
  parent: webApplication

  dependsOn: [
    cloudflare
  ]
}
```

As you can see, we need a lengthy PowerShell script that we had to write ourselves to get done what the other tools nicely abstracted away for us. You also do need to know another language and tooling to get the same result.
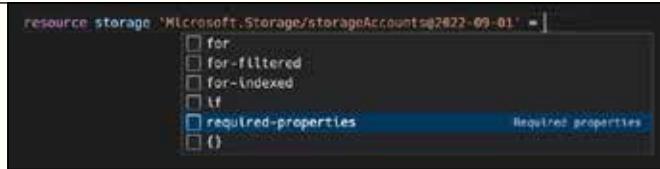
Developer experience
An important difference between these three tools is the language they give you to write your Infrastructure as Code. Both Terraform and Bicep are what we call a Domain Specific Language (DSL). A DSL is what you would think it is; a language created for a very specific domain. In general, these DSLs are a bit easier to learn compared to a Programming Language (we will talk about programming languages in a bit). That is partly because they are less complete and guide you a bit more in a certain direction of doing things. That is not to say that they are easy. You still need to know how to properly structure things not to make a big mess out of your IaC.

What sets Pulumi apart from the other two tools is you can use your favorite programming language to write your IaC. You can use Java, Node, Python, .NET (F#, C#, VB) or GO. Any of these languages are way more flexible than a DSL. They are better known by more people, well supported in a wide range of IDE's and other tooling, and communities around them tend to be much larger. A downside could be that a programming language is harder to learn. You need to be quite an experienced developer to write maintainable and testable code, for example. The power and flexibility,
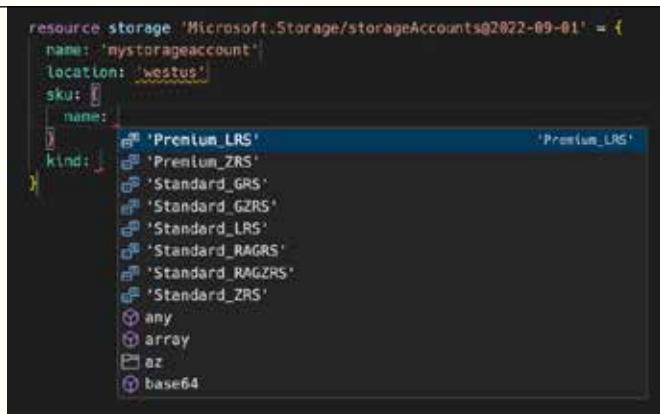
however, you get with a programming language gives you unlimited expressiveness and control far beyond what Bicep and Terraform could offer. You could use different tools like Azure App Configuration to store your configuration, use Key Vault to store secrets, or use feature flags to hide new infrastructure or only deploy that for a specific client first. It is also very easy to incorporate Pulumi code in a tool you build. If you are a platform team building a self-service portal, Pulumi could be integrated to handle the deployment to various services. Onboarding a new team could be a single click of a button that creates a repository in GitHub, create a subscription in Azure, creates the connection between GitHub and Azure, and set the correct permissions on both environments.

IDE and other tooling
A big part of the developer experience is the availability of proper support in your favorite IDE and additional tooling. Starting with Bicep we can say that support in VS Code is really good after installing the extension. It supports validation, Intellisense, Snippets, Code navigation, code completion, formatting and even a few quick fixes when making a typo, for example. A feature I often use is the completion of a resource using the 'required-properties' option. That will be presented to you when creating a resource as shown below:

When I hit enter, the complete resource will be created, and I get to fill in the blanks that are required. It then also helps me to fill in the correct value for properties that have a defined list of allowed values preventing me from picking the wrong option or making a typo.

Another interesting feature is the one that allows you to import an existing Azure resource. You provide the identifier, and the bicep template will be generated. That is a quick win when you want to create infrastructure as code for resources that were created manually in the past.

The VS Code extension for Terraform is not as complete as the one for Bicep. It should support things like Intellisense but I find it lacking quite regularly. It does support snippets but only for a limited set of its built-in features such as using a for-each or creating a variable. It does not have snippets for specific resources. That, of course, makes sense since it would then need to support snippets for many, many providers. There are extensions for that, but you need to find and install them yourself and, at least for Azure resources, they are not as complete as in Bicep. Something like presenting a list of options for a well-known value like the storage SKU in the above Bicep example is not available in at least three extension that I've tried.

The developer experience in the IDE for Pulumi is completely different. It depends on the language you choose to write your IaC in and the editor that you use. The experience is no different than writing any application using that tool and IDE. I can only speak for C# in combination with VS Code, Jetbrains Rider or the full version of Visual Studio. What I find is the developer experience is way better than for the other two tools. A full-fledged IDE like Rider or Visual Studio in combination with a powerful language is simply hard to beat.

If we look beyond the IDE at additional tooling, then we see that Terraform does a really good job. There are tons of additional tools out there that help you keep things manageable and secure. To name a few: there is TFLint (A static analysis tool for Terraform code that helps you detect and fix style issues, syntax errors, and best practices violations), TFSec (Uses static analysis of your terraform code to spot potential misconfigurations), terratest (a Go library that provides patterns and helper functions for testing infrastructure), checkov (Checkov is a static code analysis tool for infrastructure as code) and terraform-docs (Generates Terraform modules documentation in various formats). All these tools nicely integrate with git pre-commit allowing you to prevent unwanted code in your main branch.

Bicep is a bit behind but is quickly catching up here. Quite recently, for example, support for checkov was added.

Integrated into Bicep itself is a linter that has quite a few rules that are checked automatically. For those that worked with ARM Templates, these rules are the same as in the ARM Template Toolkit. Bicep is also supported by PSRule, a cross-platform PowerShell module to validate infrastructure as code. It currently has over 250 rules for best practices defined by Microsoft. PSRule also allows you to write your own tests and thus is extensible. Since Bicep is transpiled (Transpiling is a specific term for taking source code written in one language and transforming into another language that has a similar level of abstraction.) into an ARM template, any tool that is supported there is supported for Bicep although that is not always ideal since you then get your results on a generated ARM template instead of your specific Bicep template.

The ecosystem for Pulumi seems to be a lot smaller compared to the other two tools. Of course, you can use more general tools that would work with your language of choice. A testing framework is a good example. I have not found any security scanner like TFSec or SonarCloud, that supports Pulumi. SonarCloud, for example, could scan your C# code, but it will not look at how you specifically configure a storage account.

Developer flow

Creating infrastructure with any of these tools starts with writing the code. When you want to deploy that code, things look a little different. Terraform and Pulumi both take the default route of first figuring out what needs to change, show you that result, ask you whether that is correct, and then allow you to confirm or abort the deployment. In Terraform you often use the 'terraform plan' command to figure out what needs to change, and then run the 'terraform apply' command to apply the changes. Pulumi has similar functionality. Below is an example on what Pulumi would present us when we deploy a simple storage account as you've seen in the introduction of Pulumi.
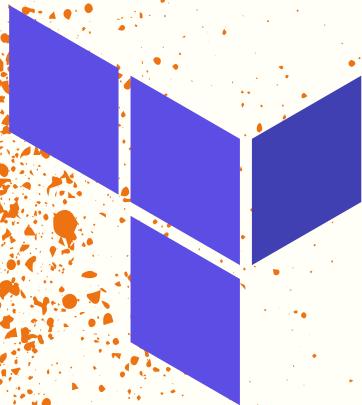
Pulumi and Terraform can give you this functionality because they store every single detail of the infrastructure you deploy in what we call a state file. That allows them to compare your current templates with what should be deployed and what the configuration in your cloud or services is. That does mean that this state needs to be stored somewhere. Both tools offer a SaaS service to handle that for you, which they obviously charge you for. You can also store these state files in, for example, an Azure storage account. You do need to keep them secure since, especially for Terraform, they do contain plan text password and keys.

Bicep does not know this state thing. Azure is its state. Whenever you run a deployment, your Bicep templates will be compared with the current state in Azure and changes will be applied. You can, optionally, use the -what-if flag. That will then show you what changes will be made. That feature, however, still contains a few bugs and therefore is not as reliable as in the other two tools[1].

Another difference with the other two tools is where the changes are being executed: server-side or client side. Bicep is run server-side. Your Bicep templates are transpiled into ARM and then sent to the Azure Resource Manager. The other two tools execute your changes locally using their engines by calling APIs. That has a few advantages. One of them, for example, is that Terraform and Pulumi allow you to wait for x amount of time between the deployment of two resources. That can sometimes be convenient when assigning permissions and waiting for a few seconds before those are in effect. Bicep does not offer such a feature.

[1]   https://aka.ms/WhatIfIssues
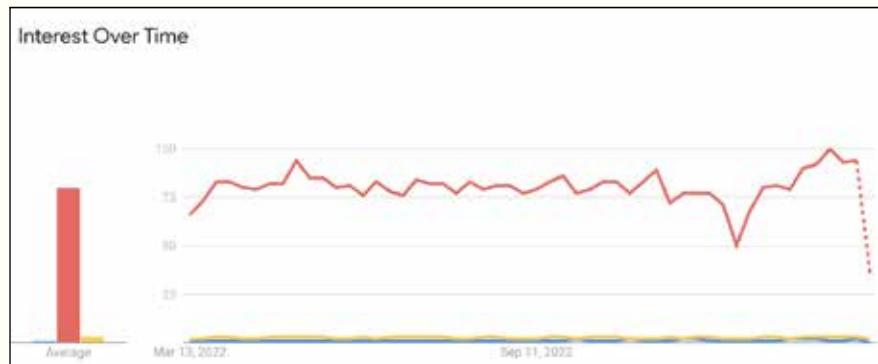
Community

As with writing any piece of software, when writing your IaC you will run into trouble, get errors that you cannot explain or have questions on how to best do things. It is therefore essential that the weapon of choice here has a large and vivid community to which you can reach out. It is a bit hard to quantify large and vivid. What I have done is look at both the search trends in Google and StackOverflow. Let's start with the first one. When we look at the number of questions posted using a specific tag then we see the following data:
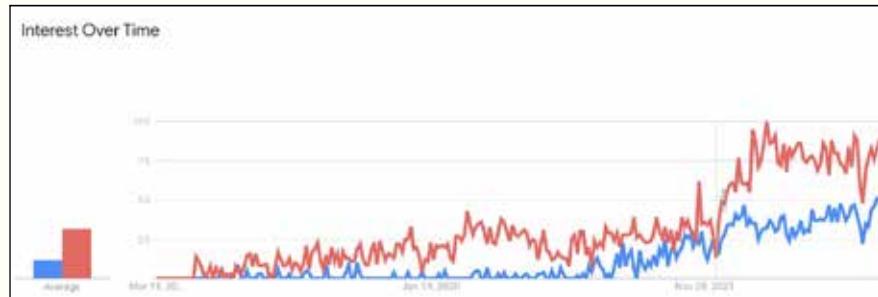
What we learn from this is Terraform has the most questions and activity on the platform by far. Results for Bicep and Pulumi are quite similar. We do have to point out here that the questions for Bicep will, of course, be also always Azure specific. That will not be the case for Pulumi, so there might be fewer relative questions and answers there.

The next graph shows Google search trends for the following three keywords over the last year: azure bicep (just bicep gives a lot of other results as you can imagine...), Terraform and Pulumi.

Red: Terraform, Blue: Bicep, Yellow: Pulumi

What we see here is that, again, Terraform is by far the biggest in terms of being searched for on Google. Bicep and Pulumi are so far behind that we cannot get any useful numbers for them from this graph. The next graph shows only Bicep and Pulumi for the last 5 years.

Red: Pulumi, Blue: Bicep

What we see here is Pulumi is more popular compared to Bicep at a first glance. When we do consider that the target audience for Pulumi is way bigger, you could argue that you probably get better results for Bicep as they will be more relevant to you. The upwards trend is quite similar for all three tools.

So, which one?
Now that we've looked at these three tools from different angles, it is time to name the winner. Unfortunately, as always, that depends on various factors. Let's start with Bicep. I find it to be the easiest tool to learn and explain to others. Tooling is good, and it is very easy and quick to deploy your first resource. Not needing to store state somewhere is a small plus. It can only configure items in the Azure cloud which very much limits its use-cases. I would pick Bicep when working on small environments only in the Azure cloud with an in-experienced team.

Terraform is much more powerful and therefor a good choice for more advanced environments that span multiple cloud or service providers. It also lets you configure the internals of a resource (remember the control plane vs data plane discussion with Bicep?) allowing you to really configure your infrastructure end-to-end using a single language. With power comes complexity and responsibility. Having to manage state and learn to work with it makes Terraform have a bit steeper learning curve compared to Bicep. I would go for Terraform for any project that is not super simple, unless...

Pulumi is by far the most powerful tool of the three. The expressiveness, power, and freedom a programming language offers is unparalleled. If you can live with a smaller ecosystem, which might change, and have engineers that know how to use a programming language, Pulumi would be my weapon of choice.
</>

All code used in this article can be found here:
https://github.com/staal-it/article-bicep-terraform-pulumi.

READ MORE ONLINE

Adding Load Testing to your CI/CD workflows in GitHub Actions

Load testing is a technique that focuses on evaluating the performance of an application under normal or expected load conditions. The goal is to determine how the application behaves when it is subjected to the expected levels of usage and traffic. Load testing is often used to verify that a system can handle the expected number of users and transactions, and to identify any performance bottlenecks or issues that may impact the user experience.

Author David Sanchez

Microsoft Azure offers a service called Azure Load Testing. One of the key benefits of using this service is that it allows you to test your application's performance at scale without having to invest in expensive hardware and infrastructure. Additionally, it is highly configurable and can be used to test applications hosted on a variety of platforms, including Azure, on-premises servers, and third-party cloud providers.

What do we need?
In addition to an Azure Subscription, and a GitHub account, we will need an Apache JMeter script, which typically consists of a series of test elements, including thread groups, samplers, listeners, and assertions. The thread groups define the number and type of virtual users that will be simulated, while the samplers define the specific actions or requests that will be performed by the virtual users. The listeners capture the performance data generated by the test, and the assertions define the expected results of the test and verify that the actual results match the expectations.

Here is an example of what the JMeter script looks like. You can find this script in the sample code that I created as part of this article https://raw.githubusercontent.com/ dsanchezcr/LoadTestingDemo/main/LoadTestingScript.jmx.
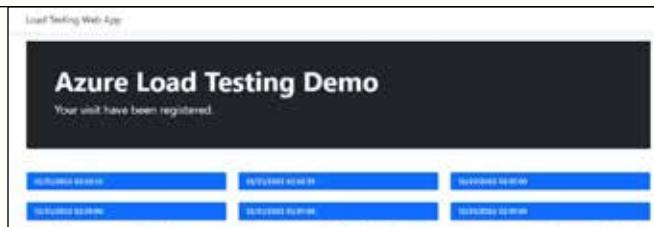
Getting Started
In the following example, we are going to use Azure Load Testing in our GitHub Actions workflow to detect when our web app has reached a performance issue. We are going to define a Load Test scenario with a specific number and type of virtual users that will be simulated, as well as the test duration and the type of workload to be simulated, which in this case is just an HTTP Request. In addition, you can also use either Visual Studio or the Azure Portal to create and configure your load test scenario.

Once the load test scenario is defined, we can review the results and the monitoring data, which includes metrics such as response time, CPU usage, and network traffic, as well as custom performance counters that we can define. With this data we identify bottlenecks and optimize the application's performance.

The scenario

I developed a simple Web App built with ASP.NET Core using .NET 7 that connects to an Azure Cosmos DB and adds a record of each visit to the page and retrieves the data from all the visits.
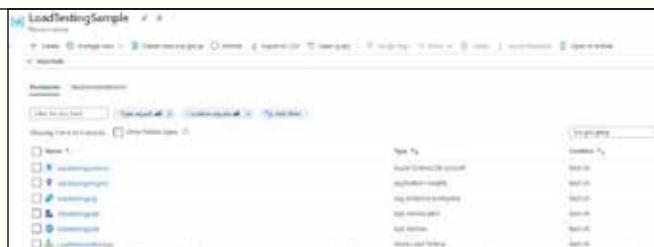
Here is a screenshot of how the application looks like:

The environment

This web app is running on an App Service Basic plan, and it has Applications Insights to monitor the performance of the application. The Cosmos DB is set with the free tier (1000 RU/s and 25 GB). I want to find out if the application running in this environment can support up to 100 c oncurrent users.

Here is a screenshot of the Resource Group created with the Azure resources for this application.
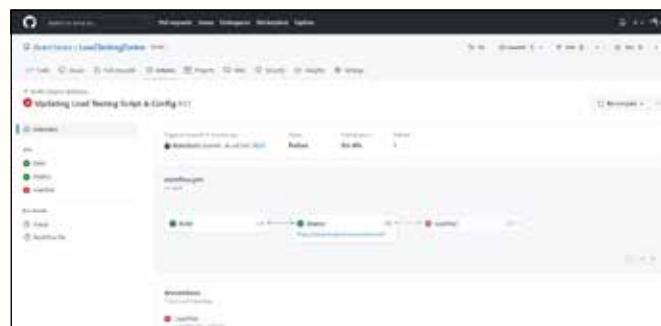
The repository

You can check out the GitHub repository in this link (https://github.com/dsanchezcr/LoadTestingDemo). There you can fork the repository, use the ARM template to deploy the Azure services needed and run the Load Testing in your environment.

⚠️ **Note:** Microsoft Azure only allows you to create one Cosmos DB Free Tier resource per subscription, you might get an error if you already have one Cosmos DB Free Tier in your subscription.

This repository has a GitHub Action that Builds & Deploys the application and runs the Load Test in Azure Load Testing. You can find the workflow in the Action tab of the repository.

Here is a screenshot of what the Action looks like. You can see it is failing in the LoadTest job.

The GitHub Action

The workflow consists of three steps and runs on every push. This first step builds the .NET application, the second step deploys the application to the Azure App Service and the third step runs the Load Test job using the following files that are in the root folder. The first file is the JMeter script used to specify the steps of the Load Testing, the second file is the configuration of the Load Testing.
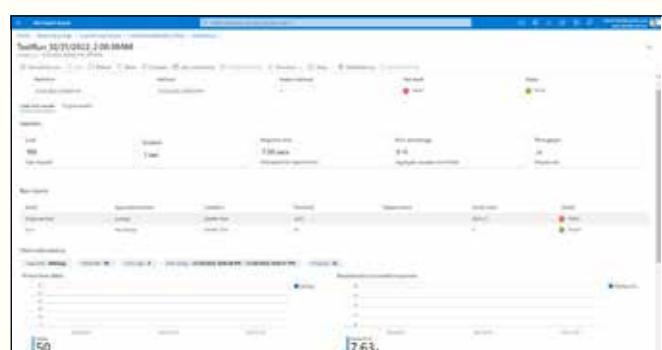
- LoadTestingScript.jmx
- LoadTestingConfig.yaml
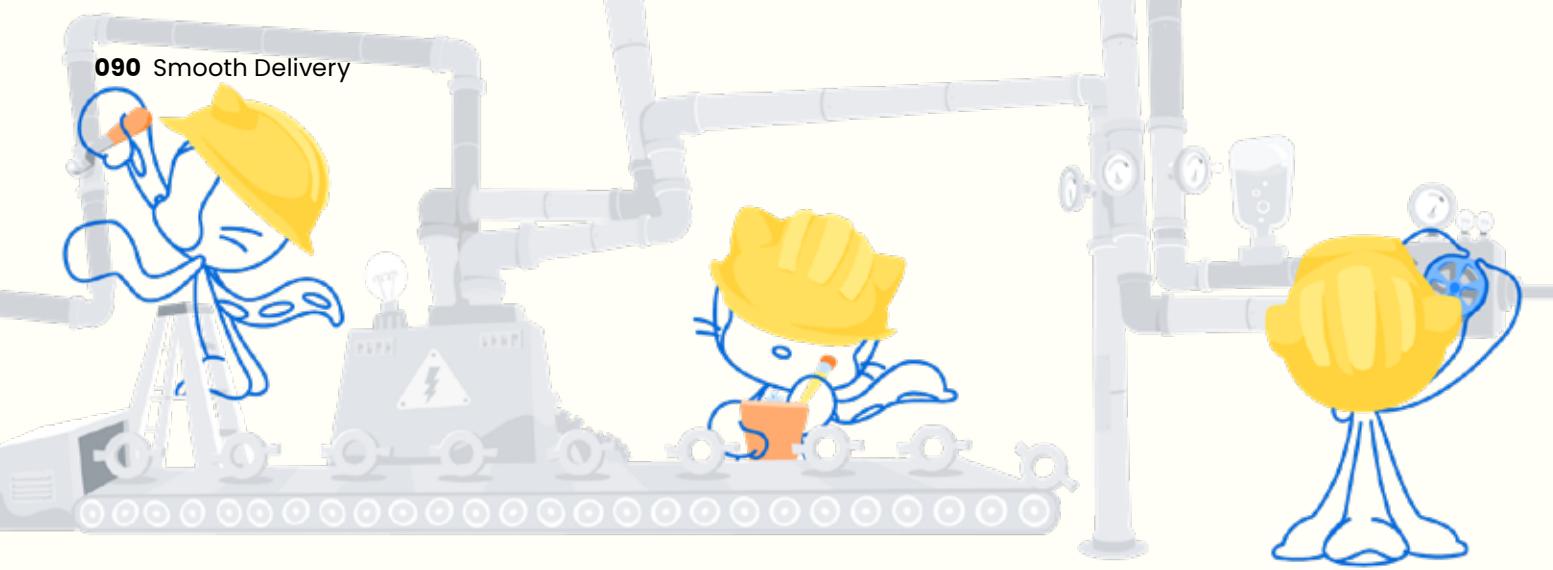
The Azure login is required to communicate with the Azure Load Testing service to send the JMeter script and the configuration for the test. In this configuration, we can define the number of engines we want to run the test and the failure criteria, in this case, the threshold we have is an average response time lower than 5 seconds and error percentage lower than 20%.

The Results
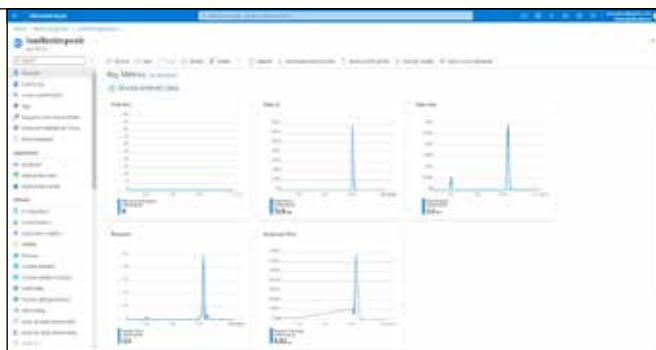
As you can see from the previous image, the Load Test failed because the average response time was higher than we expected (5 seconds). We can get more details about the test run in the Azure Portal.

Here is a screenshot of the result of the test run in the Azure Portal.
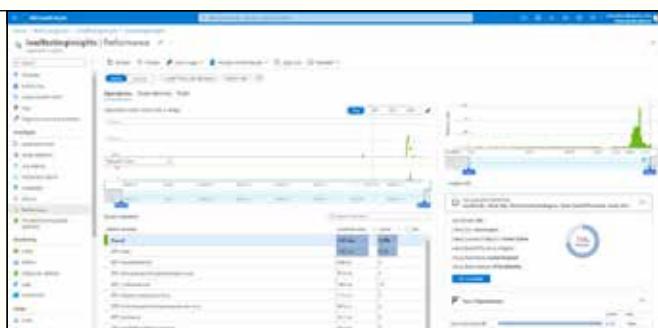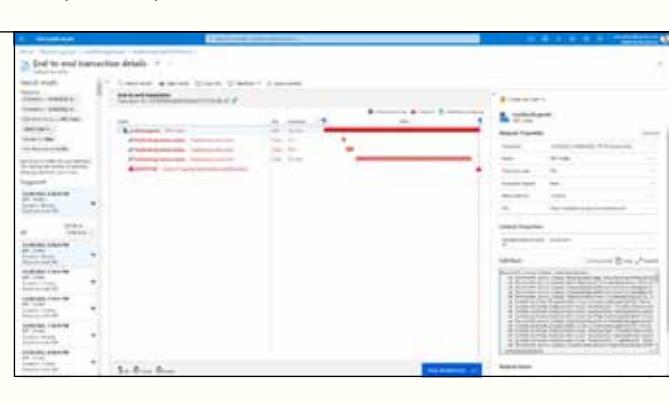
In the Azure App Service, we can see the metrics with the response times (higher than 5 seconds) and the number of requests with the Data in and Data out. Here is a screenshot of the key metrics in Azure Portal:

In addition, I added Application Insights to monitor the web app. In the Azure Portal we can see the performance issues and failures. Here is a screenshot of the performance section for the web application:

From the image above you can see where the requests came from. In this case, I am running Azure Load Testing in the East US region (Virginia). Here is a screenshot of the exception captured in the transaction:
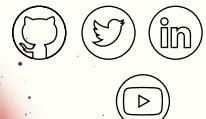
Conclusion

Load Testing should not be run in a production environment. Try it on a Dev/Test, QA or pre-production environment. Even if you are running on deployments slots, remember that the web app will still run on the same App Service Plan, and this could affect your production environment or cause a Denial-of-Service Attack.

If you would like to learn more about Azure Load Testing, I recommend you review the service documentation using this link: https://learn.microsoft.com/azure/load-testing. </>

David Sanchez

READ MORE ONLINE

TOGETHER WE DRIVE CHANGE.

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

www.xpirit.com