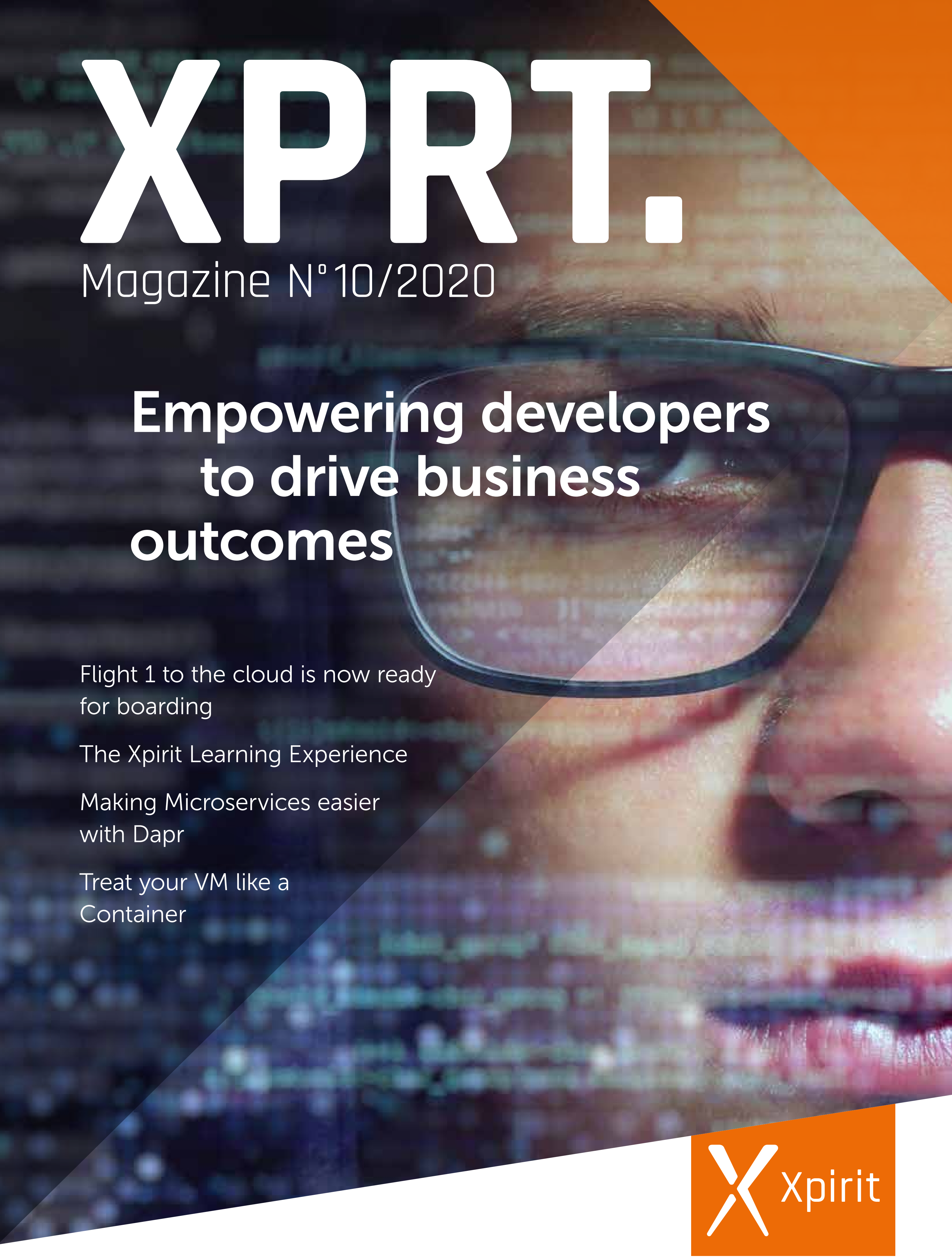


XPRT.

Magazine N°10/2020

Empowering developers to drive business outcomes

Xpirit

Think ahead. Act now.

PROUDLY PART OF XEBIA GROUP

DEVOPS BOOTCAMP

GLOBAL, LOCAL & VIRTUAL

ACCELERATE DEVOPS ADOPTION WITH THIS EXCLUSIVE DEVOPS EXPERIENCE

HAVE YOU EVER WANTED TO EXPERIENCE WHAT IT'S LIKE TO WORK IN A TEAM THAT PRACTICES REAL DEVOPS? DO YOU WANT TO RUN A DEVOPS BOOTCAMP?

Then this is the event for you! You learn how to build software with immediate feedback loops and push it to production, multiple times a day, without hesitation. You will be able to translate everything into your daily practices and initiate your DevOps transformation based on experience instead of text-book examples.

DO YOU WANT TO RUN A DEVOPS BOOTCAMP? CONTACT MAX FOR ALL OPTIONS.

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

Colofon

XPRT. Magazine N° 10/2020

Editorial Office
Xpirit Netherlands BV

This magazine was made by
Alex de Groot, Alex Thissen, Anne Meijer, Arjan van Bekkum, Chris van Sluijsveld, Loek Duys, Duncan Roosma, Erick Segaar, Gill Cleeren, Pascal Naber, Rob Bos, Geert van der Cruijsen, Immanuel Kranendonk, Jasper Gilhuis, Jesse Houwing, Erik Oppedijk, Kees Verhaar, Maira Camu, Manuel Riezebosch, Reinier van Maanen, Marc Bruins, Marcel de Vries, Natascha Former, Sofie Wisse, Martijn van der Sijde, Max Verhorst, Niels Nijveldt, Michiel van Oudheusden, Pieter Gheysens, Marc Duiker, Sorin Pasa, René van Osnabrugge, Roy Cornelissen, Sander Aernouts, Thijs Limmen, Jordi Borghers, Thomas Browet, Lesly Bernaola, Pieter Nijs, Stéphane Eyskens,

Contact
Xpirit Netherlands BV
Laapersveld 27
1213 VB Hilversum
The Netherlands
+31 (0)35 538 19 21
mverhorst@xpirit.com
www.xpirit.com

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

Translations
TechText

In this issue of **XPRT.** Magazine, our experts share their knowledge about Digital Transformations & Continuous Integration.

Stay Safe, Stay Home, Stay Relevant, Challenge Accepted!

Just in time before the holiday starts, we are proud to present to you with the tenth edition of our Xpirit magazine. It has been quite the journey since we presented you with our first edition in early 2015 only a few months after we started our company. Sharing knowledge is part of our DNA, and this is something that we are determined to keep doing.

Author Marcel de Vries (Chief Technical Officer)

The past few months have been extremely challenging for our talented team, while they needed to work fulltime from home and at the same time be a fantastic parent, teacher, partner, and employee. Of course, this wasn't easy, and yes, it took everyone a lot of energy to make it work, but still, there is so much energy and passion left that the team was determined to deliver you this magazine just before the start of the holiday season. What better time could we pick to bring you a series of articles of new learnings, new technologies, and new insights in the way we can use technology and people to move our society forward.

In this edition, you will find a variety of articles from deeply technical to more inspirational. And that is where the magic lies, combining the in-depth technical with the human side of our industry. We also received some great articles from well-known industry influencers, of whom we find it a privilege to call them close friends. Leni Lobel, who is a renowned industry expert in the domain of Cosmos DB was so kind as to write an exclusive article for this edition on Data Modeling and Partitioning in Cosmos DB. And Martin Woodward, who recently moved from Microsoft to GitHub, and who provided us with his insights and knowledge on how to successfully implement inner source in your company in order to manage shared code across various teams.

It is with great pride that we are presenting you with this new edition of our magazine, even in these challenging times. We hope you will all enjoy reading our knowledge and insights and that they will inspire you and help you in your career in the years to come. Stay safe, stay healthy, and enjoy! </>

Get challenged and inspired...

We are Xpirit.

Together we drive change.

WE BRING THE ENERGY TO CREATE THE FUTURE NOW!
do epic sh!t
Xpirit

Sink or Swim? How do you Survive a Forced Digital Transformation?

Behind every digital transformation, there's usually a solid, well-considered plan and matching strategy. Due to COVID-19 and the suddenly changing economy, many companies feel forced to digitize immediately. All of a sudden, everyone is expected to be able to work remotely. It requires much effort to make everything work digitally, straight away.

Author Marcel de Vries

How do you survive such a forced digital transformation?
Marcel de Vries, CTO, and founder of Xpirit, a subsidiary of Xebia, the Dutch Powerhouse in Amsterdam, say's,
"We compare the current situation somewhat with swimming lessons. First, you carefully put your toe in the water to make sure it's not too cold, then suddenly you're pushed into the water, how well do you swim? The big question is, what is your reaction when you have recovered from the initial shock? Do you swim back to where you fell in, or do you adapt and swim in the direction of the new environment? If you give it some thought, swimming in the new direction turns out to be better than you ever expected!" According to de Vries.

Many companies have spontaneously accelerated into a digital transformation, with all employees forced to work from home. They may have had a digital transformation plan on their to-do-list but didn't implement it. At moments like this, decisions reached in crisis mode. Which generally take weeks or even months, can have enormous, incalculable consequences.

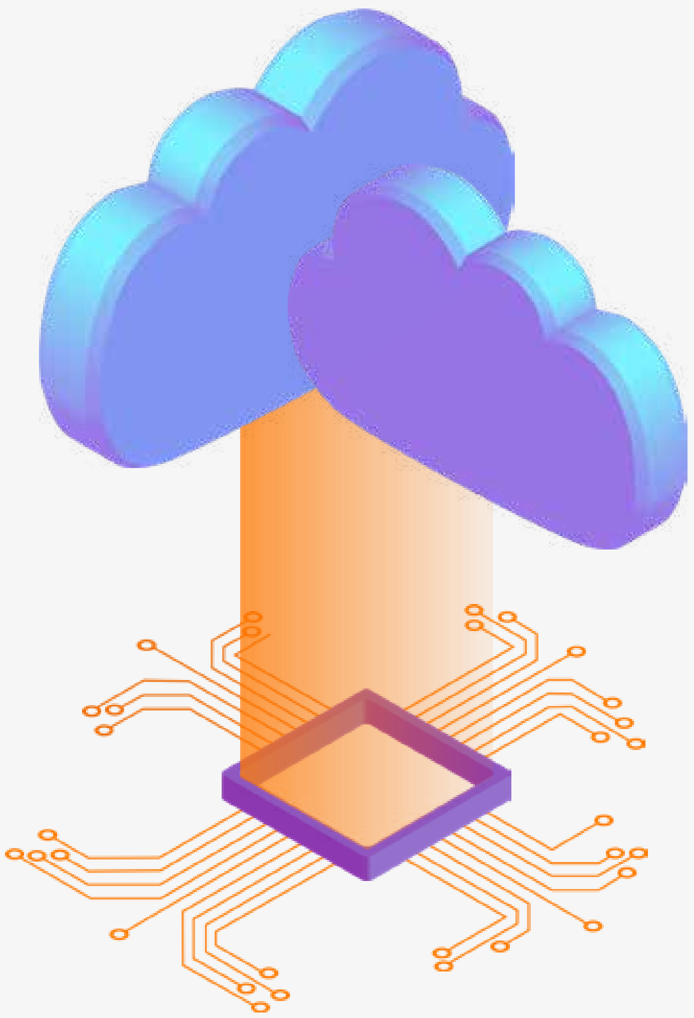
"On one hand, you immediately start with digital collaboration. Think of using Zoom or Microsoft Teams. On the other hand, you face the challenge of applying remote engineering. Your engineers located throughout the country suddenly have to bring software to production with their teams. How do you do that when you are not near to a colleague? How do you do that when you need to depend on the development software in your company? How do you ensure that everyone can access it safely and still stay compliant?"

Keeping their head above the water is currently a top priority for every company. In a crisis like this, essential issues as risk and compliance do not come first. In the past, you could easily take the time to look at all possibilities, and companies are now choosing business continuity. De Vries says: "It simply takes too long to carry out an extensive risk assessment. You need to immediately make sure that your entire business does not come to a halt. You will solve problems later! That is a logical choice if you are in crisis management mode."

"Due to the current situation, there is also more leeway for temporary solutions. It makes everything more fluid, after which you can still do the remediation to fix it. Adjust afterward and see what you should have done differently or better. That's exactly the right mode for companies. Work-based on feedback and adjust! We instructed companies to implement this model before the crisis. It can still help them now be more efficient. We know what it means to do a digital transformation. We have been doing this for the past five years. We know what's involved. Let us help you optimize your current state and help you accelerate. After all, you are now in the water, so let us teach you how to swim the fastest and smartest. That is what we do, and instead of planning for it, we now can help you act on it right away."

The IT Consultant as a lifeguard

According to de Vries, executing a so-called forced digital transformation is a blessing in disguise. "Many companies had cold feet for the transformation but are now thrown into the deep at once, where they have to learn to swim immediately,



so they're forced to deploy their digital transformations faster. Now that you're in the water, it is instrumental that you learn how to swim more efficiently or how to last longer if you swim in a different direction. We have known the pool for a while..."

"We often encounter resistance and cold feet at companies; at some point, you sometimes feel like giving them a push. But of course, that would not be appreciated."

Above all, companies should not think that they are already there and absolutely should not go back to the 'old' way. "You are already in the pool. Go for a swim and keep your head above water. Please take advantage of it and start seeing it as an opportunity. If you go back to the old way, all the pressure and misery has been a waste of time. At banks and insurance companies, we see that the new way is working. Some of them already plan to downsize their offices.
Embrace what is good and only throw away what does not work. Otherwise, there are many missed opportunities.
You also see change-oriented organizations. They are moving ahead and taking advantage of what is currently happening. These are the companies that emerge as digital winners. Those are the organizations that say; it will never be as before. It is wiser to go ahead and win the race. We are now swimming anyway, so let's aim to be first across the pool."

"Our consultants enjoy entering this phase. Beyond the resistance. You feel and experience the pain at the moment, and you have to take advantage of that."

Cost-saving, more safety, better compliance, and cooperation; remote engineering

Many companies that have now taken the first deep dive into digitization are currently confronted with high, unnecessary costs and do not yet work efficiently. Their current implementations are not designed to be compliant and still have a high risk of not being implemented safely. The number of cybersecurity attacks is currently unprecedented. It creates enormous problems in the future that must be repaired. "Before data is out on the street, it is better to assess and adapt fast.

There are also many smarter ways to optimize collaboration. It would be best if you adapted to the model of the high performing IT organization, working in value streams, according to DevOps. Secure and compliant by default."
Says de Vries.

The costs for a digital transformation quickly recouped

Especially when you first start helping with the cost side of a company. "Everyone will be confronted with high digitization costs in the coming months. We can often quickly provide insights on what can be done much cheaper and more efficiently. Our Cloud assessment e.g., provides you with advice on where you can save costs and which investments you can or should make to earn back even more in the long term."

The choice for a digital transformation depends on which angle one comes

De Vries says: "Is the goal cost optimization? Then we can do a scan, looking at how you set up Cloud subscriptions, what your cloud spend is, and how cloud native your software architecture currently is. Based on this, we can advise you on the measures to take that save costs. For example, by adjusting your architecture, optimizing functionality, and using the technology already there, we can make smart changes to your software so that you can adopt native cloud functionalities and no longer have to choose for expensive options. Think about setting up more efficient plans for your virtual machines or making some software that utilizes the serverless capabilities of the cloud. In the cloud, you pay per use. That's different compared to the billing you have been used to form your datacenter or hosting provider. We can also show you exactly when you have recouped the investment. Architectural changes can often be recouped in 3 months or half a year, depending
on the size, of course."

When it comes to collaboration, DevOps, and remote engineering, you need to look at how teams can work together more effectively and efficiently
If you are adopting DevOps tools like Azure DevOps or Github as a service, it would be best to make your teams work smarter together. Smart collaboration and ease of CI / CD setups are a few examples. It's the crucial moment you can embrace DevOps principles. You will significantly increase your efficiency by actually doing continuous deployment and bringing your software to production several times a day."

"Everything is going remote; there is no reason to be in the office. Unfortunately, that is sometimes still in the minds of people. Fortunately, we can do Everything remotely."

"Where one normally has a software development team in an office, these developers now work from home. It is shifting from local engineering to remote engineering. Offering many opportunities, overlooked with an office mindset. You can now have a larger talent pool you can draw from since locality is not an issue. You can develop software 24/7 at lower labor costs. Azure DevOps, GitHub, Visual Studio Online, and Azure Cloud are useful services to support that. With this technology, you can work worldwide. "The tooling for remote engineering is simply available, but unfortunately, it is still not mainstream for many companies. Today, hosting your software development tools yourself is not efficient and productive anymore. Just purchase the already available SaaS product, then you don't have to manage the infrastructure anymore yourself, and you can use that time to create value for your customers." say's de Vries.

Xpirit has been successful in setting up and guiding companies towards the Cloud for more than five years
They have gained a lot of knowledge and experience in the field of what does and does not work and learned a lot from it. By learning from mistakes, they can protect customers from this. "Everyone has the right to make their own mistakes, but sometimes it is beneficial to know what you will encounter in advance. The choice is yours. Do you want to experience it first before you run into it, or do you learn it from someone else?"

Xpirit is ideally suited to help with the knowledge injection you need now. They do not pull it out of your hands, which creates a dependency, but come to help with the knowledge that is now needed to immediately start working faster and more efficiently, after which your teams can all 'swim themselves'. Say's De Vries: "Instead of taking everyone out of the water, we will teach your team how to swim best in this pool. Because of the experience we have gained over the years at banks, insurance companies, and ISVs, we have gathered quite some insights into what works and what doesn't. e.g., the compliance requirements and laws that come up every time. We now know how to set up Azure in such an environment, compliant with requirements, such as SOC 2 Type II, Cobit, ISO and SOX, etc. We even made standard solutions for it."

Adopt & Embrace
"Where are you now? Look back and decide what you want to keep and what you want to throw away. Don't go back to what was. The only way is forward. It's your chance. Go for it. With our knowledge and expertise, we are happy to help you look retrospectively at how the transformation has been for you thus far and where the opportunities are for you. Companies that say we want to, adopt and embrace, are the digital winners of the future", says de Vries. </>

CUSTOMER STORY
Experience highs in the cloud Application development with higher speed, higher quality and higher efficiency

knab
CUSTOMER STORY
Continuous dynamics demand continuous delivery

https://pages.xpirit.com/xpirit-customer-stories

Marcel de Vries
Chief Technical Officer

"You are the master of your own destiny and success in life. It is all the result of your own actions and choices made in life. Never blame others, but look at how your own behavior and actions bring you where you are and define who you are."

https://xpirit.com/xpiriter/marcel-de-vries/

Encouraging Inner Source

There are many things that developers can argue about. Tabs vs spaces, vim vs emacs, the position of '{' after an if statement – and don't get us started on variable names. However, one thing that unites developers is the love of solving problems and the joy you get from a particularly well executed solution. I can't remember the number of times I've got a bit of code just right and I spend the next 10 minutes just switching a toggle on and off to re-run that bit of code and delight in how well it works. This is usually the point when you go show it off to your team-mates. The best software engineers love to share and love to get better at the craft of writing software.

Author Martin Woodward (Director of Developer Relations - GitHub)

Many times, this is how open source projects start. You work on a problem at home, come up with a solution you like so you post it to GitHub. Before you know it, someone else looking for the solution to a similar problem discovers your solution and tells you about an issue they found, and ideally a suggested fix that you can pull from them that makes your code better.

If you think about open source projects, it's amazing that they work at all. A collection of individuals around the planet all working in different locations, different timezones, often speaking different languages and using the software to solve a problem in different applications all with different business needs and different deadlines. And yet it does. 99% of the applications we see deployed today contain some open source, and even more amazingly 80-90% of the code you ship is made up of your open source dependencies with 10-20% being the code that you wrote.

But inside your organization, you also have lots of shared code and shared logic. It might be how you validate and format customer reference numbers, common logic for talking to in-house developed services or Terraform code used to deploy an application into production. And yet often we find we are better at sharing code with random strangers on the internet than we are with the team sitting upstairs in your own company. This is why many leading companies like Microsoft, Google, IBM, NASA and more turn to inner source practices to support sharing inside their organizations.

"Our new GitHub Verified Partner status shows that GitHub and Xpirit work together to help customers both on the GitHub platform as well as keeping our strong position in Azure DevOps."

What is 'inner source'

Inner source is the sharing of knowledge, skills and code inside your organization using open source style workflows. Easy to say – but what does that actually mean in practice?

It means providing systems inside your organization to allow people to share with each other. It can be as simple as a file share or a SharePoint site, but with modern tools we can do better than that. For years we've had scalable source control systems inside most companies.

However these source control systems are typically configured with restricted read/write access. Only the teams who need access to source control have it and they typically guard that access jealously. Inside a company, when a developer gets access to source control it traditionally was always read/write access and there would be much grumbling when the 'dumb' team from across the building checked in some change which broke your build and stopped your team from being its usual awesome self that day. It is a maxim of any enterprise that the further a team is from your team in the org chart or geographically, the more likely they are to do 'dumb' things. When you realise this is true in every company and in every team you work in you quickly realise it can't in fact be true – maybe it's communication challenges that are to blame. Also, why are the processes in place in your organization such that a team is able to mistakenly block you from working?

By adopting open source style collaboration inside the organization, you set up a version control system that is distributed. You have permissive read access but keep write access restricted to the maintainers of that code (i.e. the same small core team as before). Others in the company can read the source code of everyone else, take a fork of it and then send over a pull-request if they would like to suggest some changes and have them reviewed by the team who maintain that component. GitHub is obviously particularly well suited to supporting these types of workflows as you can set your projects as private or shared with anyone in your organization and then easily control the behaviour of forks and pull requests – note that this is still all private to your organization. You are just following the workflows of open source, with the crucial difference that you are not making the code public to your competitors.

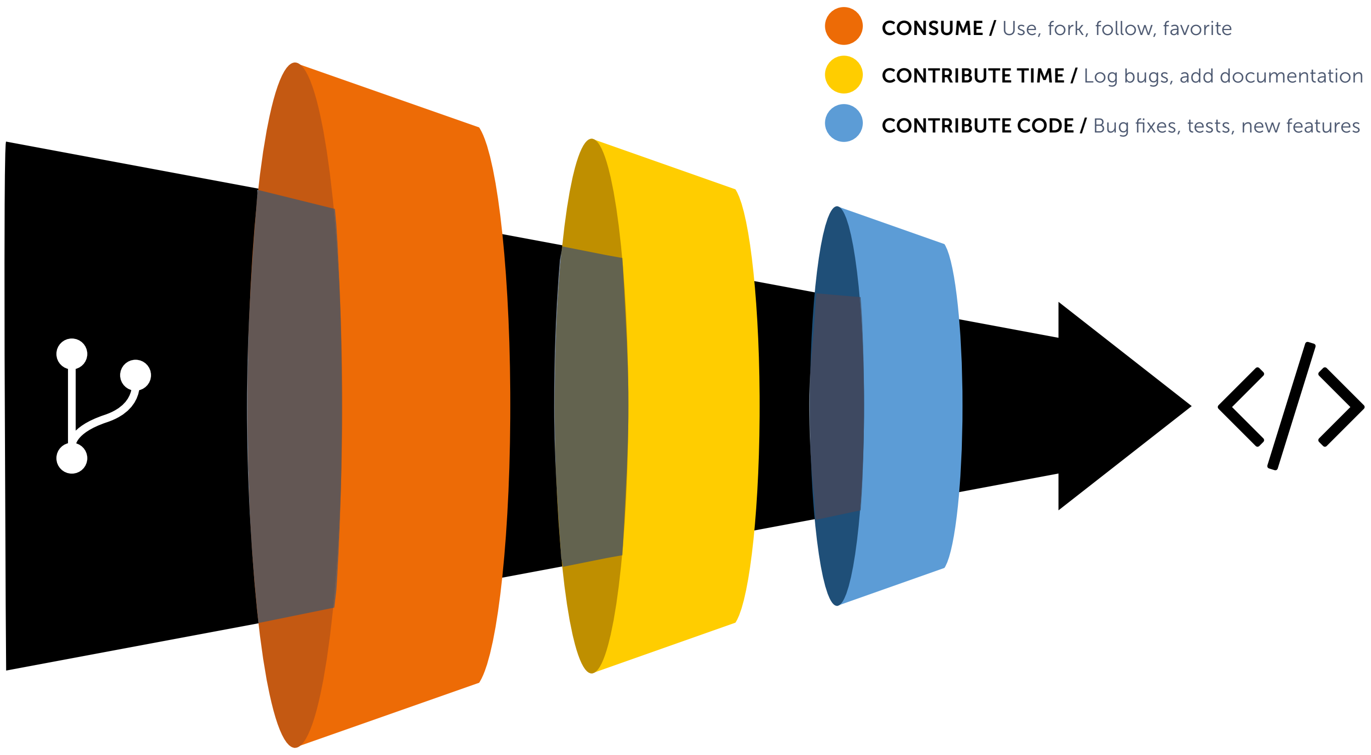
It turns out that inside our companies, we also have many teams that are geographically distributed. They are working on different project and often have different deadlines. There are many more similarities and lessons we can learn from working in the open that we can apply to cross-company collaboration with inner source.

It's not just setting up version control to allow sharing, and indeed to make it so that sharing by default becomes the norm inside the company. You want to look at the contribution funnel to understand how to inner source and open source projects work, see Figure 1: The Contribution Funnel.

With inner source and open source projects, the vast majority of people will just consume a shared component. This is perfectly natural and to be encouraged, however you want to maximise the number of people coming into the top of the contribution funnel. Therefore discoverability is important – how do people find shared components inside your organization? Again, GitHub has tooling built in to help this but if you are implementing inner source using other tooling then you need to pay close attention to discoverability. Even if you are using GitHub, are you providing a clear ReadMe file that helps people identify what the project is about and help them find and consume the code easily? Are they able to navigate the code and scripts to find what they want to re-use?

A small percentage of the consumers contribute back time. That might be telling you about issues / bugs, helping document areas or even telling their colleagues about your component and advocating for it. Therefore you need to think about how you help maximise the number of people coming down into that funnel. How do people find out how to contribute time in a useful way to you, what information do you need to capture in a bug / issue etc, where do you most need help? You also want to be welcoming to these people as they come in so that they feel part invested in the project and the internal community around it.

A small percentage of the people contributing time will contribute code. Again – how can you maximise that funnel? You want to make sure that you are responsive to pull requests, you want to make sure any coding guide-lines are documented so folks know how you would like code contributed. You want to make sure you have an easily repeatable build environment with clear and easy to reproduce dependencies. Ideally you should also have an automated CI build set up that runs on every change but also runs every time someone send over a



● CONSUME / Use, fork, follow, favorite
● CONTRIBUTE TIME / Log bugs, add documentation
● CONTRIBUTE CODE / Bug fixes, tests, new features

Figure 1: The Contribution Funnel

At Xpirit we are extremely proud to announce we achieved the first official github Verified Partner status in Europe.

Don't hesitate to contact us if you like to know more about how we can help you transform your organization towards DevOps!

GitHub Verified Partner

pull request with a suggested code submission – that way the person contributing code gets instant feedback if the code compiles, passes your tests, meets your quality bar and is ready for human review.

And finally, of the people contributing code, only a small percentage will help maintain that code going forwards. It may be that you want to strictly control access to the list of maintainers to people on your team – and that is fine. But understand that if you choose to do that then you are forcing the other group to permanently work on a fork of your codebase which means you will end up with diverging codebases over time. So you should ask yourselves what is in the best interest of your organization and your shareholders and take the decisions about joint ownership based on that. Remember – it's not your team that owns the code. You are paid to maintain it on behalf of the company and its shareholders who are the people that own it.

Having all the tooling in place to support inner source is great, but that is no good unless you have the culture in place to change how people work. As we mentioned, developers naturally love to share cool things so you have that working in your favour. However there are emotional barriers in place that often hinder sharing. People feel a strong sense of 'owner-ship' of source code. Often times they are worried about the judgement of colleagues across the organization about the quality of that code which might make them hesitate to share it. They might worry about their ability to share knowledge with-in an organization without getting intro trouble. They might also not want to pay the teamwork tax of communicating with others or taking dependencies on other teams that outside their managers direct reporting line and sphere of influence.

Therefore, as an organization you need to strive to build an economy of sharing with-in the company. You need to positively encourage individuals and teams that share and highlight their achievements. You should also look at your incentives for your engineering teams. When rewarding them, do you look at what impact they have had alone or do you also look for evidence about what impact they have had on other teams and what work they have done that has built on the work of others? By making those questions part of your core incentive model you not only encourage teams to find opportunities to work with each other you also encourage them to highlight the fact they have worked with other teams rather than trying to take all the credit. This ensures that all the people involved feel that they have been recognised and are getting rewarded.

By encouraging a culture in your organization that rewards collaboration, that allows developers and ops teams to learn from each other and share best practices you also build a culture that has a growth mindset and that is always looking to learn, to get better and to improve. Not only is that exactly the sort of place that I want to work, you'll find it's the kind of company culture that will attract and retain lots of high performing talent. More importantly, it definitely makes work a lot more fun and rewarding. I highly encourage you give inner source a try. </>

Martin Woodward
Director of Developer Relations, GitHub

"If you want to put a bit of sparkle into your @GitHub Actions."

Running 30 year old software as a cloud native SaaS solution with Docker and Kubernetes on Azure

For over four decades, students aspiring to become seafarers on one of the world's many ships, either as a navigator on the ship's bridge, or chief in the engine room, have studied and honed their skills on Kongsberg's simulators. Believing that knowledge is instrumental to safe and cost-efficient operation, Kongsberg has strived to remove the limitations inherent in the students operating environment to enable their customers, the instructors, and educators, to create any training scenario. With the help of the simulators, the instructor can create the most vivid and challenging experience for the students, such as groundings, collisions, communication blackouts, system failures, or a hundred-year storm.

AuthorS Gullik Anthon Jensen (Technology Director @ Kongsberg Digital), Roy Cornelissen (Consultant @ Xpirit, working with Kongsberg since 2017) & Sander Aernouts (Consultant @ Xpirit, working with Kongsberg since 2017)

The maritime industry is transforming, just like the transformation from sail to steam, or from the compass to GPS, the future of the maritime industry is autonomous ships, green shift, and remote operations. The educators are not only faced with this challenge of transformation but also the problem of digitalization. Students now take instant access to digital services for granted. To bridge this gap, Kongsberg is deploying a new platform using cloud technology to deliver traditional simulation training in a new and different way. By combining proven and loved simulators customers are confident and comfortable with and new cloud-native technology, simulators become even more accessible anytime and anywhere for students. Students can now keep building their competency outside the classroom and be as prepared as possible for the challenges that lie ahead.

We call this platform K-Sim Connect, a journey that started in 2017 by moving Kongsberg's engine room simulator (ERS) to the cloud. In this article, we will share the challenges we faced, the solutions that we have chosen, and the lessons we learned.

Moving a 30+-year-old software to the cloud

Kongsberg created the engine room simulator over 30 years ago and up until this point had only been installed on-site at costumers. We aimed to bring the engine room simulator to the cloud without having to change it too much to enable both on-site and simulation as a service delivery models.

Kongsberg's engine room simulator simulates the engine room of a specific ship model. It thus allows students to learn, for example, how to perform a cold start or emergency shutdown without physically being in the engine room of the ship.

Students perform a specific exercise, such as an emergency shutdown, and the simulator tracks their performance as part of an assessment. Instructors use this assessment as a pass/fail criteria for classes and specific certifications.

[1] https://azure.microsoft.com/en-us/support/legal/sla/api-management/v1_1/

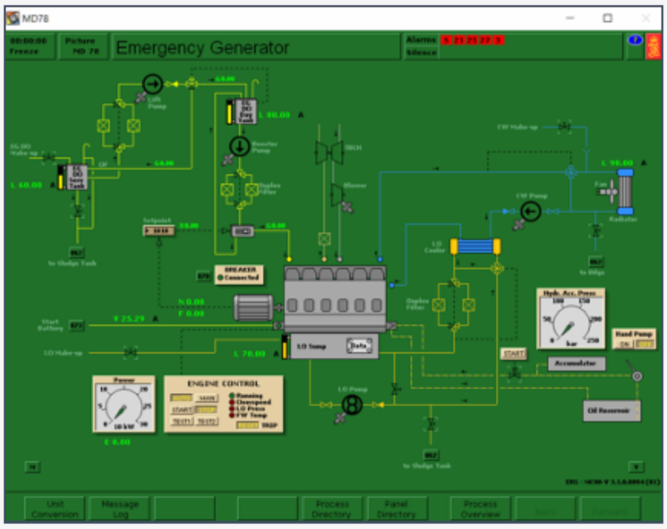


Figure 1: engine room simulator screen

Figure 1 shows a typical screen of the client application. The engine room simulators consist of many of these screens. Each screen represents specific controls that are in the actual engine room of the simulated ship model. On a customer site, these digital screens can be replaced by a physical replica of the ship's engine room to allow for an even more immersive learning experience.

To understand what it takes to bring this simulator to the cloud, you must understand the basics of how the engine room simulator works.

The engine room simulator is a client/server application where the simulation of the state of the ship's engine room runs on a server application (called simulator in figure 2). The students connect to this server through a client application (see Figure 1). First, the client and server perform a handshake process to determine the range of ports used for further communication. Next, the client and server exchange several messages over this range of ports.

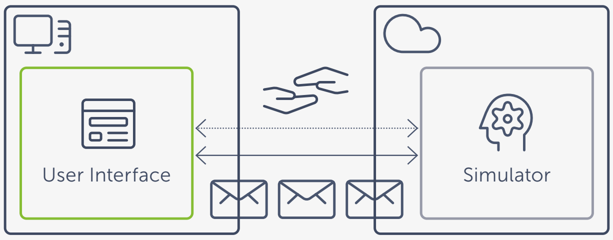


Figure 2: ERS topology (simplified)

Figure 2 assumes both applications run on a single computer. Still, it is also possible for multiple clients to connect to a single server in the same network as part of a collaborative exercise.

Since the engine room simulator has a client-server style architecture and it already supports running a distributed setup. So the most straight forward way of bringing the engine room simulator to the cloud was only to move the server part (simulator in Figure 3).

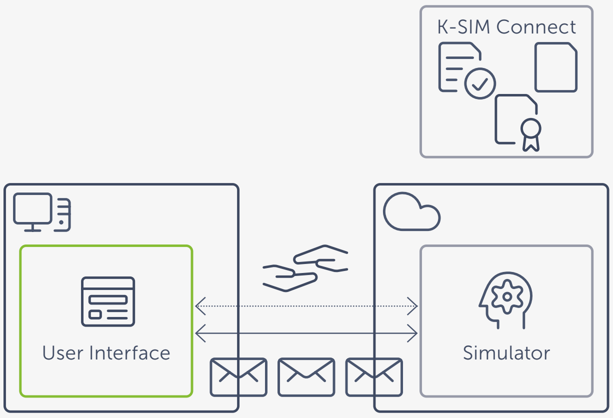


Figure 3: Moving the ERS to the cloud

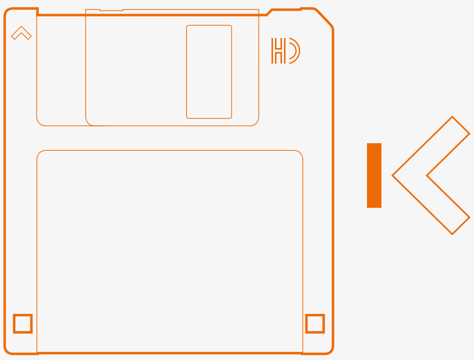
The rest of this article will cover the three main challenges we ran into bringing the 30+-year-old engine room simulator to the cloud.

The 1st challenge: containerization

The engine room simulator was built over 30 years ago, well before the age of containers. Our challenge was to run it on-demand, and in the cloud, so we decided to put the simulator in a Docker container. Containerization did, however, pose some challenges, for example, the code used low-level Win 32 API calls with C++, it uses arcane constructs such as "/etc/services", and relies on Windows Registry settings. These Windows-specific constructs meant we had to use Windows Server Core containers, which are some of the largest Docker images that exist. Also, when we started in 2017, the Windows container community was small (it still is), and official support in Docker related open source projects was simply not there. But containers did fit our needs perfectly, so we decided to try and use Windows containers to bring Kongsberg's engine room simulator into the cloud era.

The 2nd challenge: the internet

After successfully running the engine room simulator in a Windows container, we faced another challenge.
Since Kongsberg built the engine room simulator well before "the cloud" even existed, its distributed installation option assumed that the clients and server were at least on the same local area network (LAN), meaning that there are no firewalls in the middle. This assumption posed a challenge because the engine room simulator uses a proprietary communication protocol that dynamically allocates ~200 ports as part of the initial handshake process.



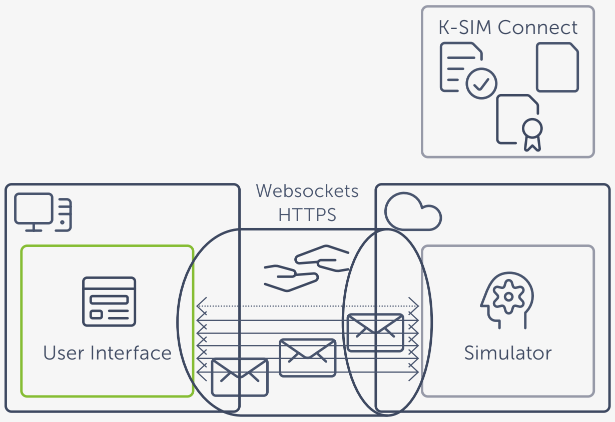


Figure 4: Tunneling over HTTPS Websockets

To overcome this challenge, we needed the help of the vendor, who made this communication protocol. They made a specialized tunnel for us that tunneled all messages over HTTPS using a single WebSocket connection. This tunnel allowed us to connect the client and server application over the internet.

The 3rd challenge: on-demand simulation

We were now technically able to run the engine room simulator in the cloud and to connect to it over the internet. But to run simulations as a service, we still needed a way for students to start simulation anytime and anywhere using the Azure cloud. There were several container orchestrators available, but in 2017 already, Kubernetes had the largest community and was getting adopted by the major cloud vendors. However, when we started, Windows containers in Kubernetes was still in beta. Windows containers in Kubernetes became generally available in 1.15.0 (June 19th, 2019).

Initially, we chose AKS engine to provision our Kubernetes cluster in Azure. AKS engine is the tool that Microsoft uses under the hood to provision AKS clusters, and since we knew Microsoft was working on supporting clusters with Windows nodes, we felt this was the best approach available to us. AKS engine generates the required templates and script to provision a cluster, but you end up with VM's that you have to manage yourself.

Recently Microsoft started officially supporting multiple node pools in AKS, which means that you can also have Windows nodes, although the Windows part of this feature is still in preview at the time of writing.

Figure 5 shows a simplified topology of how we utilize Kubernetes to run simulations as a service in the cloud. There are three main components involved in providing our simulation as a service solution to the students.

We have a portal where the students can, amongst other things, select an engine room model they want to train on and choose an exercise they want to run. We have a WPF application that runs on the student's computer, starts the simulator client, and configures it to connect to the simulator running in the cloud. And we have a scheduler component that creates the required Kubernetes resources to run the simulator in the cloud and makes it accessible to the simulator client running on the student's computer. All three components use a single SignalR Hub to communicate.

To start a new simulation in the cloud, a student will select an exercise in the portal and request to run it. Doing this s ends a message to our scheduler, which will then create all required Kubernetes resources. The scheduler will then publish a message to the WPF application on the student's computer, which starts the simulator client application and configures it to connect to the simulator running in our Kubernetes cluster.

There is a lot more going on behind the scenes to run these simulations in the cloud, but explaining all of that would be an article on its own. Instead, we will share what we learned from this journey.

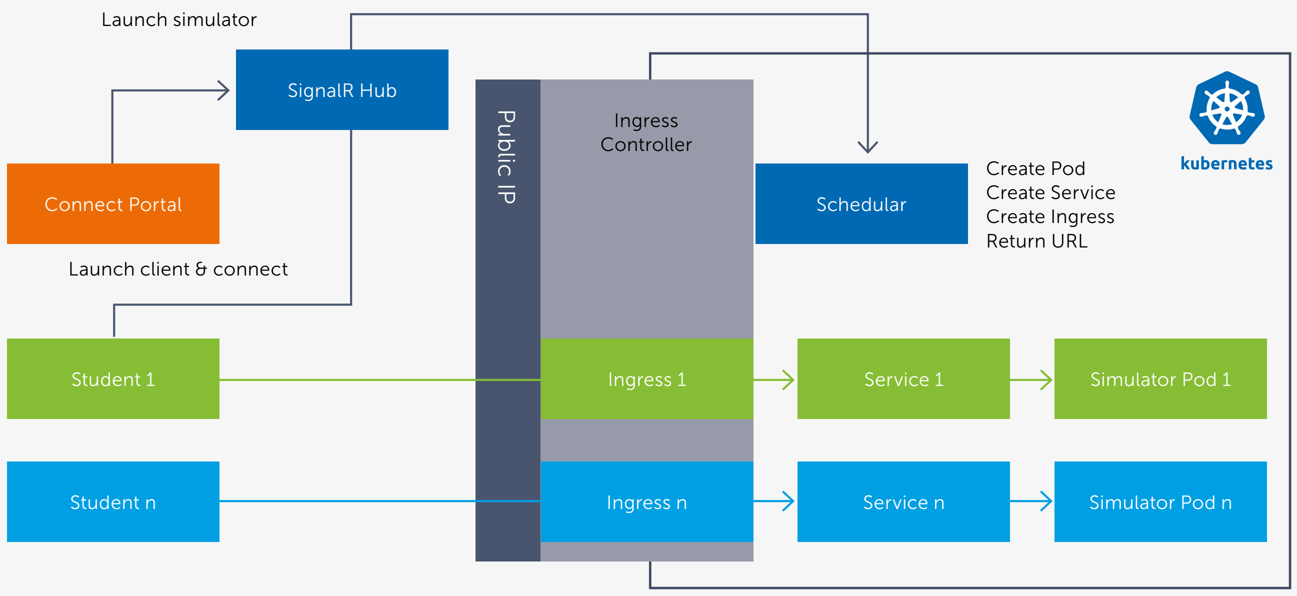


Figure 5: Kubernetes topology (simplified)

What we learned

Besides the technical challenges we had to overcome, the engine room simulator was surprisingly well suited to run in the cloud. Its client-server architecture allowed us to move the server to our Kubernetes cluster and move the client to the student's computer.

Windows containers are different from Linux containers. Windows containers are simply a lot larger in size, especially if you need the full Windows Server image like us. There are also challenges with the Windows Server version you run on your Kubernetes node and the Windows Server version of the docker image, and these have to align to some level.

Using beta/preview versions of Kubernetes because we needed Windows container support meant we did face some technical difficulties, but Kubernetes itself works well with Windows containers, especially after version 1.15.0. And the concept of Kubernetes, scheduling container workloads on-demand, is a perfect fit for the problem we had to solve: running simulations on demand. So, we choose to work with the restrictions and challenges that come with using beta/preview features over building an orchestrator. We firmly believed that official Windows support was on its way. So not having to develop an orchestrator certainly paid off, especially now that Kubernetes is officially supporting Windows nodes, and Azure Kubernetes Service (AKS) is close to officially supporting windows node pools as well.

It is possible to bring old software to the cloud and even change the way you offer it to your customers without doing a complete rewrite. Containerization with Windows containers helped this transition even with software that is 30+ years old and still allows Kongsberg to offer both models, on-demand simulation, and on-site installation.

"Windows containers are different from Linux containers."

What the future holds

The global market already embraces the K-Sim Connect platform and is in the early phase of adopting simulation beyond the training centers and schools. What seemed like a threat to the experienced instructors only a few years ago has turned into enthusiasm and discussions about the opportunities. Today the engine room simulator is our first cloud simulator in operation, but we will not stop there. We are already working on bringing navigation simulators and applications like radar training, navigation, and maneuvering to the cloud. Kongsberg is transforming the industry of maritime simulation and training and is leading the way to the future. Together with customers, students, instructors, legislators, and even competitors, we will continue this journey in confidence to unfold our collective future. </>

Roy Cornelissen
Distributed architecture, mobile development, creative

"Software development is the creative art of problem solving. Trying to squeeze more 'productivity' out of developers by over formalizing and industrializing processes is like asking Picasso to paint-by-number because it yields more masterpieces faster."

https://xpirit.com/xpiriter/roy-cornelissen/

Sander Aernouts
Microsoft application lifecycle management (ALM)

"Start by doing what's necessary; then do what's possible; and suddenly you are doing the impossible."

https://xpirit.com/xpiriter/sander-aernouts/

Gullik Anthon Jensen
Lead digital transformation Maritime Simulation, Kongsberg Digital

"Today we are transforming the maritime training industry by empowering every seafarer to acquire new skills and competency, so they can build the future of maritime industry."

Flight 1 to the cloud is now ready for boarding

Migrating your company IT into the Azure cloud is a complicated and time-consuming process. You need to think about many things such as: team collaboration, cost control, connectivity to on-premise systems, governance and compliance, education on how to use the cloud, and still facilitate efficient onboarding of DevOps teams. In this article, you will read about my experiences in helping an airliner transform from using a classic datacenter into Azure.

Author Loek Duys

The case

About a year ago, the management team of the fictional airliner, "Not Invented Aire" (NI-Aire), decided it was time to move their IT systems into Azure. They came to this decision because of ever-increasing friction between development teams and the datacenter operator. A few years ago, they accepted that a dedicated operations team made all changes to IT infrastructure and that it was acceptable for this to take a few weeks to complete. Today, they are adopting ever more DevOps practices. DevOps Teams must be enabled to deploy both software and infrastructure changes whenever needed, and sometimes they need changes multiple times per day. Their datacenter operator is unable to comply with this need. Fortunately, we can accomplish this by using Azure. So, all teams are now moving their workloads into the cloud. Before the migration started, NI-Aire workers had some questions:

> How do we connect with remaining on-premise systems?
> How do we deal with security and compliance?
> How can we share cloud knowledge and best-practices?
> How can we get insights into our Azure consumption?

How do we connect with remaining on-premise systems?

Microsoft recommends using a Hub & Spoke architecture to provide connectivity from cloud services to on-premise systems. A Hub & Spoke architecture looks like the diagram shown in Figure 1.

Instead of having to create a VPN connection from every team's subscription to the on-premise network, the Hub & Spoke architecture uses a single connection from a shared network, the Hub network. It uses Azure ExpressRoute[1] to create a link from the Hub network to the on premise datacenter. The teams can then create their own (Spoke) networks and connect them to the shared Hub network by peering[2] them.

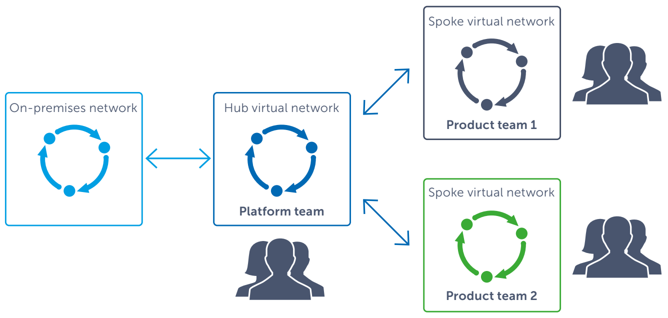


Figure 1: Hub and Spoke architecture

In this example, two teams have peered a Virtual Network with the Hub network. Each team uses dedicated Azure subscriptions.

✔ Connectivity with on-premise systems

We solved the connectivity problem, but unfortunately, we also introduced two new ones. One of the issues that arise from adopting this architecture is the management of the shared Hub network and the ExpressRoute connection; which team owns these resources?

[1] https://azure.microsoft.com/en-us/services/expressroute/
[2] https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview

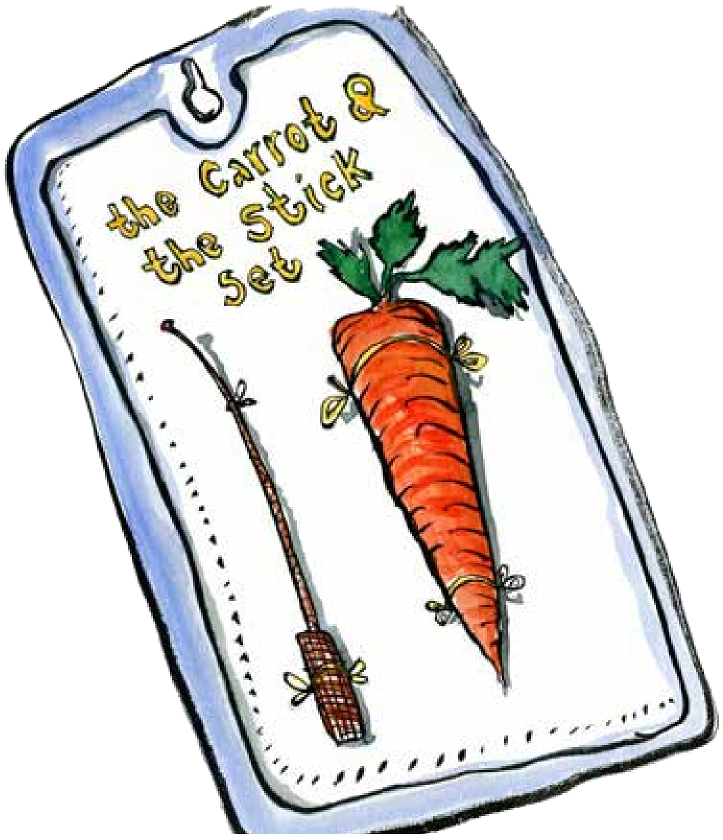
Another problem is the use of IP address ranges assigned to the virtual networks. We need to make sure that teams don't use overlapping ranges, as this would create traffic routing issues. Dealing with IP address ranges and subnets can be complicated. So, ideally, we would like to help teams by provisioning this bit of infrastructure for them. This way, they don't need to worry about this complexity.

Managing shared resources

To manage shared resources, NI-Aire chose to create a separate team; named the Platform team. This team consists of Product team members (part-time) and some dedicated people. It is dedicated to facilitating the other teams (Product teams) in their cloud migrations, not just by provisioning shared infrastructure, but also by offering hands-on assistance, sharing best-practices, and education in cloud concepts. Costs for shared resources are divided across all teams equally. Product teams can make changes to shared infrastructure by making pull requests on the Infrastructure as Code definitions, so the Platform team does not become a bottleneck and ownership is shared between stakeholders.

Provisioning infrastructure for teams

The Platform team chose to use Azure Blueprints[3] for required infrastructure deployments. With Blueprints, we can use Azure Blueprints to deploy resources, like virtual networks and ExpressRoute, that adhere to the company's standards and requirements. The main difference between using Blueprints and using regular ARM templates for deployments is that Blueprints allow us to set up a specific environment.
For example, it enables us to create resources and resource groups inside particular subscriptions.



This Photo by Unknown Author and is licensed under CC BY-SA-N

[3] https://docs.microsoft.com/en-us/azure/governance/blueprints/overview

How do we deal with security and compliance?

Coming from a situation in which the datacenter operator managed virtual machines, DevOps teams needed some guidance on operating the machines they used in the cloud. For instance, they needed help with the automated installs of anti-malware software and security patches. The Platform team applies the 'carrot & stick' analogy.

The carrot

The 'carrot' entices Product teams to use tools provided by the Platform team. For example, all Product teams are free to choose how they deploy their infrastructure. But to help them, the Platform team provides a set of validated ARM templates they can use. For instance, they provide a template to deploy a virtual machine preconfigured with an anti-malware extension. The Platform team should provide hands-on assistance to the Product teams.

The stick

When a Product team decides they would rather use PowerShell to deploy their virtual machine, it should still have anti-malware software. But what if they forget to add it?
This is where the 'stick' comes in. We use Azure Policies to check whether all Azure resources comply with company standards. For instance, we created a Policy that checks whether anti-malware is deployed to all Windows virtual machines. We combined the Policy with a remediation that automatically deploys an anti-malware solution to a VM if needed. A Policy can flag incompatible resources on the dashboard, remediate incorrectly configured resources, or prevent them from being deployed at all. We also use Policies to enforce resource naming guidelines, to deny deployments to unwanted Azure regions, and to apply role-based access control policies (RBAC). The Platform team does not only assign Azure resources, but they also assign Policies to Product team subscriptions. They do this by using Azure Blueprints, another nice feature of the Blueprints service.

✔ Dealing with security and compliance

How can we share cloud knowledge and best practices?

If we're not careful, the Platform team can quickly become a bottleneck when all Product teams depend on them to share knowledge of Azure services and deliver shared templates.
In reality, Product teamswill soon become familiar with the Azure resources they use, so logically, it should be them sharing their knowledge. This way, the Platform team only acts as a knowledge broker, not as the 'single source of knowledge'.
To give an example, at NI-Aire, Product team 1 wanted to use Azure API Management, and Product team 2 was already using it. The Platform team was aware of this, and connected the two groups and urged them to cooperate. In the end, Team 2 produced an ARM template, and Team 1 was able to use this for their deployments.

Optimized use of resources

Having every team deploying resources can introduce another problem: The software architecture requires API's to run inside a virtual network to allow communication with on-premise systems. For security reasons, these back-end API's cannot be exposed to the internet directly. Currently, only one SKU of API Management enables virtual network integration: Premium. When compared to other Azure resources, the Premium SKU of Azure API Management is relatively expensive. At this time, it costs around €2300 per month. In terms of workload capacity, the Premium tier is oversized. Therefore, if every Product team would run an instance, NI-Aire would pay a lot of money for many under-utilized resources.

Internal open-source

To mitigate the problem of under-utilized resources, the Platform team deployed API Management into the same subscription that runs the Hub virtual network (see Figure 1), named the Platform Azure subscription (see Figure 2).

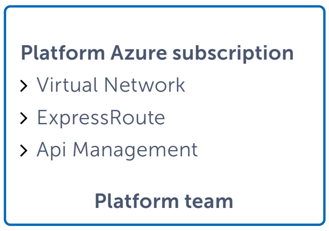


Figure 2: Platform subscription

To deploy API Management, they used the ARM template that Product team 2 created earlier, combined with additions required by Product team 1. The problem of under-utilized resources is now mitigated by sharing the resource amongst Product teams. Again, this created another problem. Who owns this resource? Will the Platform team be responsible for fixing issues and dealing with outages? This doesn't fit well in the DevOps paradigm of 'You build it, you run it'. We need a way to enable Product teams to operate API Management and other shared resources inside the Platform Azure subscription while keeping the Platform team in control over their Azure subscription.

At NI-Aire, we solved this issue by creating an 'internal open-source' Azure DevOps repository to store ARM templates. This repo contains ARM templates that are 'pre-approved for deployment' by the Platform team. They comply with security guidelines and other best practices. Any Product team can change these templates by doing a pull-request. A member from another team can then review the change and approve it. Platform team members monitor changes periodically as well. After the change is approved, a new version of the ARM template will be made available for use by publishing it to a package feed. Having Product team members as part of the Platform team enables quick approval for changes. The Platform team chose to use the Azure DevOps universal package feed[4] as the source to post and download ARM templates, as it comes with built-in version support.

This process, which is shown in Figure 3, is designed to enable teams to collaborate on Azure ARM templates, without introducing a bottleneck.

Template repository

Over time, the shared ARM template repo grew to contain lots of templates; for example, to deploy virtual networks and ExpressRoute. This way, they can be used both by regular deployments and Blueprint assignments. Over time, teams added many useful templates:

- Virtual networks that provide connectivity between services
- Virtual machine with anti-malware software and automated updates
- Log Analytics, with solutions that support VM updates
- Azure Key Vault, for application and deployment secrets
- Managed Identity, to securely connect between Azure resources
- And, of course, API Management, to securely consolidate and expose APIs.

The shared repository became the first place where Product teams looked when they needed to deploy a new resource into Azure. If the template wasn't there yet, they would contribute a new one to the repository.

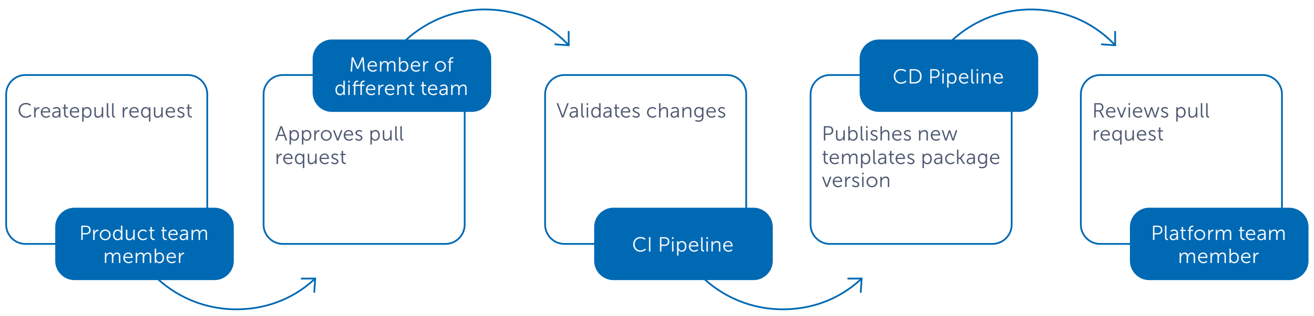


Figure 3: Collaboration on ARM templates

[4] https://docs.microsoft.com/en-us/azure/devops/artifacts/quickstarts/universal-packages

Education
Some of the NI-Aire team members had no prior experience in working with the Azure cloud and needed technical training. Again, the Platform team cannot become a bottleneck, so they wrote a curriculum of online training they could take. For example, Microsoft offers a free online introduction to Azure[5]. Pluralsight also offers some excellent training for various levels of expertise.

Of course, it is also essential for teams to learn from each other. Product team members regularly held 'Lunch and Learn' sessions, during which they would share something they learned with the rest of the teams.

The Platform team created videos in which they explain the ideas behind the tools. For example: 'How network connectivity works' and 'Why do we have an 'Internal open-source' template repository?'. This way, how to get started and quickly take off into the cloud can be explained to new team members .

✔ Share cloud knowledge and best-practices

Conclusion
I believe that the key to a successful cloud migration is to enable Product teams with a self-service platform. They need to be able to autonomously provision well-configured infrastructure in whatever way they see fit, and to be able to efficiently collaborate with other teams. With this article, I provided you with some insights into how we solved these challenges at NI-Aire. If you need help migrating your organization into the cloud, feel free to reach out. We'll help you board, so you can quickly take off into the cloud. </>

Loek Duys
Cloud software architecture

"Ten years from now, there will be no ten year old software."

https://xpirit.com/xpiriter/loek-duys/

[5] https://docs.microsoft.com/en-us/learn/paths/azure-fundamentals/

The Xpirit Learning Experience – Beyond individual learning

In this constantly changing world full of new technology and tools, training is an important investment for companies and individuals. To stay up to date, and thus relevant, following training is essential. But which training is the right choice? Because there is only so much budget, and training is expensive. In many cases, video training is a great alternative. Platforms such as Pluralsight, LinkedIn Learning or EdX provide many courses at very reasonable prices.
This training is great for gaining knowledge on a certain topic. However, it is rarely targeted at you, or the challenges you and your team face on a day-to-day basis. Furthermore, these types of training are targeted at the individual. Many companies are moving towards a DevOps Way of working and want to train their teams as a whole. Usually the way to try to accomplish this is to let all members of the team follow the same video training. While this improves the individual skills, it does not improve the team skills. We found we had to cover some ground on this level as well, and came up with the Xpirit Learning Experience.

Authors René van Osnabrugge & Rob Bos

In this article we will explain what we have learned as well as the journey towards this Learning Experience. How it started with the organization of a worldwide event that led to local spinoffs and eventually to a learning experience.

Global DevOps Bootcamp as an ignition for the Xpirit Learning Experience

In 2017 we, Xpirit (together with Solidify), organized the first Global DevOps Bootcamp. The Global DevOps Bootcamp (GDBC) is a global event, targeted at people who want to learn about DevOps. People joined a local venue at 25 locations worldwide to participate in the bootcamp. Because we wanted to ensure everybody had the same learning experience, we created an event out of the box. This means we provided keynotes on video, development machines provisioned in the cloud, all the necessary infrastructure, lunch and content in the form of challenges.

Instead of letting people do these challenges alone, the first thing we asked participants to do was to create teams. Together they could compete against their local competitors (other teams) and their global competitors (people in other venues), by providing some simple game mechanics and a scoreboard.

This was a huge success, and 3 years later, in 2019, we organized the third edition of GDBC, with 10,000 participants and 95 local venues. A lot has changed in terms of size and organization, but the concept still remains. We provide an event out of the box, where teams compete to complete challenges.

One of the great feedback items we received in 2019 was the cooperation between people with different backgrounds and the low threshold to get started. Since they had to work together, the team could be made up of different skills and maturity levels.

This meant that we had a mix of developers, engineers, people who had never used the Azure Portal, never seen Azure DevOps or who came from a different cloud provider to see how things work in a DevOps setting. We even had scrum masters joining us to help out the various groups!

Lessons learned

With GDBC we learned a number of things. We used that information to make the edition of the following year even better and applied these lessons learned in the Xpirit Learning Experience.

People do not work together automatically

When you put five different people in a team and give them each a user account, they start working alone. To overcome this, we provided the team one user account, so they are more inclined to work together. They still have an option to research things on their own or in pairs (highly encouraged), but to act with that information they need to use the shared account. This enables the team members to work as an actual team, where they need to communicate about the directions to the solutions they are taking.

Without a why, people do not get it

In the first year we made the basic mistake of not providing a good storyline. The challenges the teams needed to complete were great, but the rationale was missing.
So the logical question arose, why is this relevant? Why should I learn this? In the latest edition we changed this. We used a virtual company, PartsUnlimited, and built a storyline.
This virtual company wants to move to the cloud to fight the continuously increasing competition. Of course, this comes with a set of common, as well as not so common challenges. These challenges are based on what we see in the field at our customers. These real-life examples and proposed solutions made great material for the story of PartsUnlimited.
By creating a storyline, supported by videos with some sketches, people gained a better understanding of the rationale. By steering the choices towards a technology, without providing all the answers, teams were more inclined to research and find solutions.

Step-by-step Instructions have a reverse effect

Of course, it is great to follow a step-by-step instruction to complete a certain task or challenge, but the question is: Do people learn enough by following detailed instructions? Our experience is that true learning does not happen by following an instruction. You only learn by finding the solution yourself. In GDBC we provide the teams with guidance and links, so they can find the solution. Only after trying themselves, the proctor can decide to give them a step-by-step solution. For GDBC we created videos that show a recording of these step-by-step approaches. We saw that this made a huge difference in the behavior of the teams.

Don’t punish requests for help

Everybody likes a bit of competition. Because of the nature of the Global DevOps Bootcamp, we created a Global scoreboard where teams could see their ranking compared to other teams all over the world. This brought out the best in some teams, but we also saw the opposite. Teams were afraid of asking for help, because that would cost them valuable points. Since the event is about learning and not about being the best, we made sure teams were not punished for requesting help, but instead were motivated to do so.
By changing the mechanics of the scoreboard we made this happen. For example, instead of deducting points when asking for help, we gave points when a challenge was successfully completed.

The Platform

While the first edition of the Global DevOps Bootcamp was still very manual, the last edition was a complete self-service experience. People received a user account, a custom-made website with the challenges and they were guided through the storyline. The platform behind this experience takes care of starting and stopping challenges, causing disruptions or rolling out fixes and doing validation. This platform became the foundation of the Xpirit Learning Experience.

The Xpirit Learning Experience

After the last edition of GDBC we evaluated all our learning experiences and feedback, and found that this way of learning, learning by experience, is a great way to quickly get up to speed with new technology. We started to refine our back-end platform a bit more and created the Local DevOps Bootcamp. This event is a replica of the Global DevOps Bootcamp.
The only difference is that it is not a globally but a locally organized event. For example, to train some teams at one of our customers. Again, this worked out really well, and it was time to take it even one step further.

When one of our customers, Maersk, asked us to train their development teams for their cloud transition, this was the perfect opportunity to take this next step. Many of the people already followed some individual training, but, as we saw earlier, this training did not cover all Maersk-specific topics. The teams needed to start working in a Maersk-specific context, using the knowledge they gained through the individual training courses. This is how the Xpirit Learning Experience came to life.

The Xpirit Learning Experience follows the same principles as the Global DevOps Bootcamp
› Learn as a team, and not as an individual
› Learn by doing, without the hassle of setting up all pre-requisites
› Guide people through a storyline that explains the why
› Make it a full self-service experience
› Make it fun and engaging by using different formats throughout the day.

”Learning by experience, is a great way to quickly get up to speed with new technology.”

The Experience

To make it a bit more tangible, let's use the Maersk implementation as an example and show what the Xpirit Learning Experience can offer, regardless of the content.

Agenda

Let's start by explaining the agenda:

> Global Keynotes

To set the context and inspire participants, we start the day with a recorded Global Keynote. We have many great speakers within Xpirit and have splendid connections throughout the field. To cover the relevant topics for the learning experience, we make sure to select a speaker that is a subject matter expert on these topics. A recorded keynote makes it easier to arrange schedules and logistics, but also allows the event to be fully virtual.

> Local Keynote

In the local keynote, the organizational context can be further expanded. Why are people in the room, what is the vision of the company and what does the company expect to get out of this? It also is an introduction to the challenges ahead.

> Divide in Teams

After the keynotes it is time to divide into different groups. People can be physically together in a room, or use a virtual channel to work together. In the case of Maersk we used Microsoft Teams to put people together in a Teams channel.

> Work on challenges

When teams start working on the challenges, they should be able to do so in a self-service mode. The proctor at hand provides them with the URL of the challenges website and the credentials to log in. This enables the people to do everything themselves. They can look at the video content that guides them through the storyline. They can start challenges, they can automatically validate challenges and they can even fix challenges by pressing a button. Each challenge has a step-by-step instruction video attached, which they can use when they get stuck. The Learning Experience platform takes care of this by guiding them through the different challenges and allowing them to follow the storyline.

> Learning Review

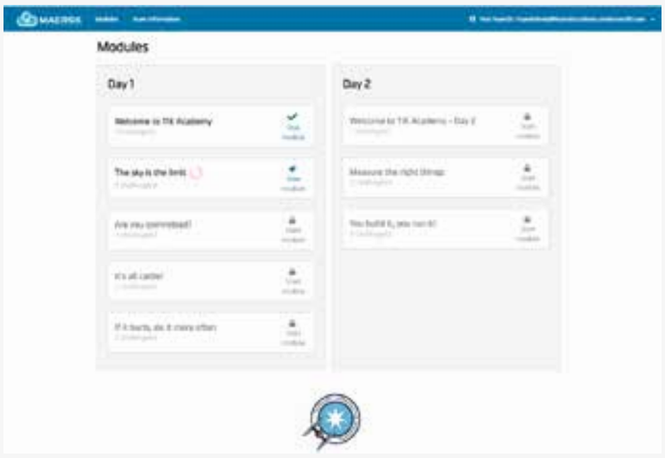
When a team completes a challenge, this is always followed by a Learning Review. This will help the team to reflect and it prepares them for the next challenges. In many cases we ask the team to fill in a short form that answers a few simple questions. What have you learned, what would you do differently, what would you do if this would happen?

> Share experience

After a set amount of time, we bring back the people and let them share their experiences by using the learning review. After that, they can start working on the next challenge.

Completing Challenges

Now that we have seen what concepts of the Learning Experience, let's walk through a number of screens to give an impression of how this looks.

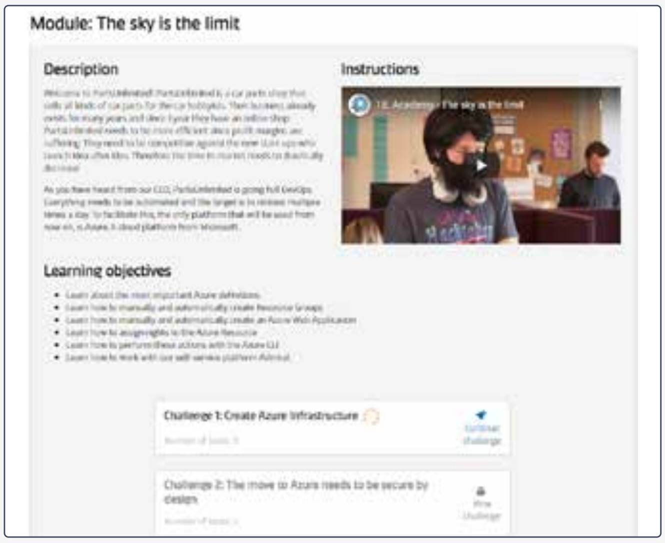


Modules

Day 1
- Welcome to TfE Academy
- The sky is the limit
- Are you connected?
- It's all cattle!
- If it hurts, do it more often

Day 2
- Welcome to TfE Academy - Day 2
- Measure the right things
- You build it, you run it!

After a team has received the URL and their credentials, they can log in to a custom-made challenges portal. From here they can go through the various modules. A module is a chapter of our story in which we cover a specific topic, for instance Cloud, Infrastructure as Code or Git. The Module describes the rationale, the storyline, and always starts with a funny sketch that sets the scene.



Module: The sky is the limit

Description

Welcome to PartsUnlimited! PartsUnlimited is a car parts shop that sells all kinds of car parts for the car hobbyists. Their business already exists for many years and [illegible] they have an online shop. PartsUnlimited needs to be more efficient since profit margins are suffering. They need to be competitive against the new start-ups who launch idea after idea. Therefore the time to market needs to drastically decrease.

As you have heard from our CEO, PartsUnlimited is going full DevOps. Everything needs to be automated and the target is to release multiple times a day. To facilitate this, the only platform that will be used from now on, is Azure. A cloud platform from Microsoft.

Instructions

TfE Academy - The sky is the limit

Learning objectives

- Learn about the most important Azure definitions
- Learn how to manually and automatically create Resource Groups
- Learn how to manually and automatically create an Azure Web Application
- Learn how to assign rights to the Azure Resource
- Learn how to perform these actions with the Azure CLI
- Learn how to work with our self-service platform [illegible]

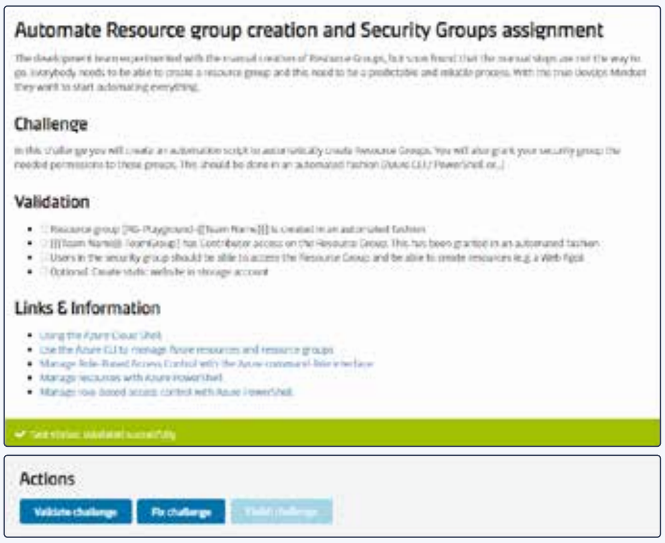
Challenge 1: Create Azure Infrastructure

Continue challenge

Challenge 2: The move to Azure needs to be secure by design

View challenge

Each module consists of multiple challenges. The challenges serve as different chapters in a module. Challenges are created in such a way that the team accumulates knowledge from the start. Instead of asking people to create a solution for the end state, we let them experience why the end state is needed. Let's explain this by means of an example. Many companies use a self-service portal to provision cloud resources. A logical choice is to educate people on using the self-service portal. However, that will not explain them WHY the self-service portal is needed. By letting them walk the line, they come to realize the rationale. In this case, people do not have rights to execute scripts on production. So, a portal that automates this for them is useful.



Automate Resource group creation and Security Groups assignment

The development team experimented with the manual creation of Resource Groups, but soon found that the manual steps are not the way to go. Everybody needs to be able to create a resource group and this need to be a predictable and reliable process. With the true DevOps Mindset they want to start automating everything.

Challenge

In this challenge you will create an automation script to automatically create Resource Groups. You will also grant your security group the needed permissions to these groups. This should be done in an automated fashion (Azure CLI / PowerShell or...)

Validation

- Resource group [RG-Playground-{[Team Name]}] is created in an automated fashion
- [[Team Name]@ TeamGroup] has Contributor access on the Resource Group. This has been granted in an automated fashion
- Users in the security group should be able to access the Resource Group and be able to create resources (e.g. a Web App)
- Optional: Create static website in storage account

Links & Information

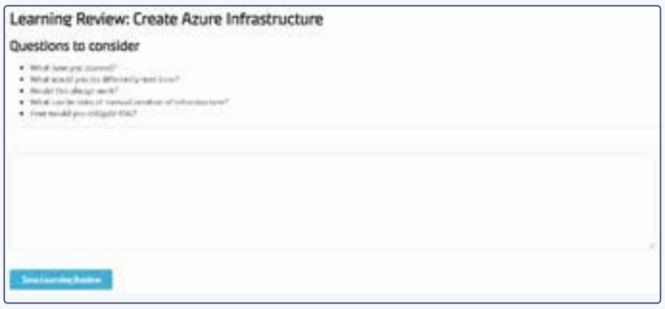
- Using the Azure Cloud Shell
- Use the Azure CLI to manage Azure resources and resource groups
- Manage Role-Based Access Control with the Azure command-line interface
- Manage resources with Azure PowerShell
- Manage role-based access control with Azure PowerShell

Task status: validated successfully

Actions

Within a challenge, the team needs to complete certain tasks to successfully finish that challenge. The tasks describe a specific problem that the team needs to overcome. The "Links & Information" section contains all necessary resources the team needs to fix the challenge. When all tasks have been completed, the team can hit the [Validate Challenge] button. This is where the platform kicks in again. The backend validates whether all tasks have been completed successfully and gives direct feedback to the team.

When a team is not able to complete the challenge, they can watch the step-by-step video that guides them through the process. When this is also not sufficient, the proctor can enable the [Fix Challenge] button. This rolls out fixes to their environment, so they look at a solution.



Learning Review: Create Azure Infrastructure

Questions to consider

- What have you learned?
- What would you do differently next time?
- Would this design work?
- What can be risks of manual creation of infrastructure?
- How would you mitigate this?

After that the team can fill in the Learning Review and prepare for the next challenge.

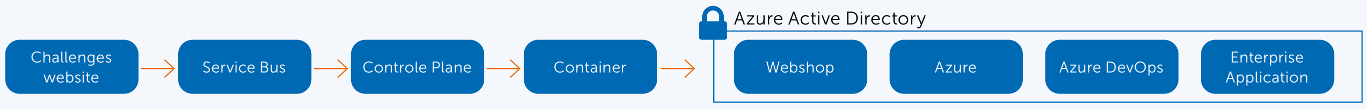
Available content

The Learning Experience can be set up with any custom modules and challenges needed for your organization.
We currently have modules available for:

- Azure Fundamentals
- Azure Automation with ARM Templates and Terraform
- Git Fundamentals
- Observability and Monitoring
- Azure DevOps with repositories, work items, pipelines and artefacts
- Site Reliability Engineering

Virtual DevOps Bootcamp

We also offer our Bootcamp as a virtual event. We have great experiences by providing virtual training and Bootcamps. While many people are still reluctant to try this, several customers already told us their virtual experience has been fantastic.



Every module starts with the fundamentals and builds up to more advance topics. Of course, it is all set up to learn things by doing them.

The platform

To make all this work, the platform that we created is quite extensive as it can accommodate a global event like GDBC with 10,000 people. To give you a brief insight, we will go over the main elements of the platform.

On the right you can see the items we provision for each team: a fully functioning DevOps environment that produces a working web shop running on Azure, with an App Service Front End and an Azure SQL database for storing the data. In Azure DevOps a team project will be generated that has the source code for the application as well as the CI/CD pipeline. They even get their own Azure Artifact Feed and service connections to Azure and Snyk Security Tooling, so they have the same environment as they would have in a real enterprise environment.

The left-hand part of the diagram shows the elements that provide the platform that is used. The attendees can log in to the Challenges Website and see their teams information, read the backstory on why they need to do something, and all the documentation and video training they need. They can start, stop, validate a challenge and we even give the proctors the option to fix their environment in case they get stuck. For example: when they trigger a challenge to start, the challenges website puts a message on the service bus. The control plane will pick up this and schedule a pod with the necessary docker containers and settings it needs to act inside the team's environment.

Using Docker gives us the flexibility to use any development stack to act in the environment and makes sure we can operate independently from other challenges and teams for example.

Conclusion

The Xpirit Learning Experience is targeted at teams that want to learn as a team. The content can be fully customized and targeted at you and your organization. Because it is fully self-service and uses video to guide teams through a storyline, the training can be followed by small and large groups, and can easily be run multiple times. This decreases the training costs per person dramatically. People who followed GDBC or one of the other Learning Experiences are very enthusiastic about it. It teaches them something by doing it, without the hassle of setting things up on a real live platform, with real life scenarios. </>

René van Osnabrugge
ALM, DevOps, Continuous Delivery, Initiator and Inspirator

"To truly innovate, do not optimize what you do, but rethink what you should do."

https://xpirit.com/xpiriter/rene-van-osnabrugge/

Rob Bos

"Where do you want to be?"

https://xpirit.com/xpiriter/rob/

Making Microservices easier with Dapr

In today's day and age, building software is all about how fast you can bring value to the market. Microservices and autonomous teams that build and run these services are an excellent fit for this goal because these teams have no dependencies on others for building and releasing their software. Microservices can be hard to build, though. You may be taking away some complexity of working together on the same solution with several teams, but they also add complexity in several ways. One of these things is the fact that if your teams want to be autonomous, they will need a lot of skills spanning a wide range of technologies.

Authors Chris van Sluijsveld & Geert van der Cruijsen

A few years ago being a full stack developer meant that you were able to do some frontend and backend development. However, today a lot more is required than programming frontend and/or backend solutions. Nowadays you also need to know your fair share about cloud infrastructures, network communications, security, CICD tooling and other tools. Having to spend a lot of time in all these domains distracts you from the reason why we went to autonomous teams: delivering business value faster. Dapr (Distributed APplication Runtime) is a new, open-source project created by Microsoft that tries to come up with an answer to these problems.

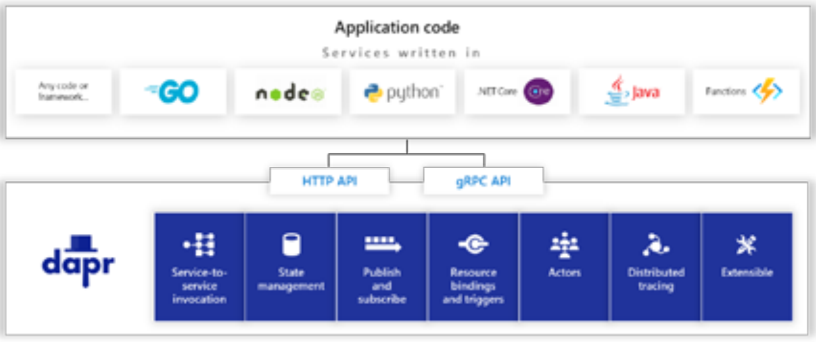
Dapr focuses on providing developers with tools that work on the cloud as well as on edge, and make it easier to build resilient microservices. It does this by handling a number of things for developers such as state management, publish/subscribe, secret management, service invocation, and it even has a built-in actor model. Dapr wants to create this in a way that works with greenfield microservice landscapes, but it can also be used in existing services to remove external dependencies. In addition, Dapr also works with any programming language or developer platform.

How does Dapr work?

Dapr is an open-source framework for building resilient microservices. It achieves this by providing developers with a set of building blocks that can be accessed over HTTP or gRPC. Because it only depends on these transport protocols, developers are free to choose from any language to develop their microservices.

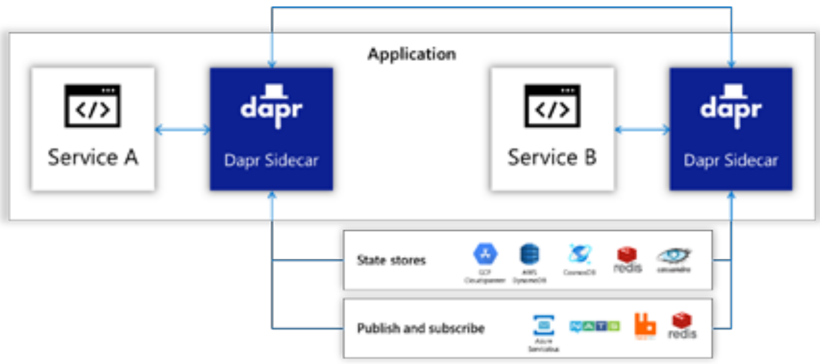
These building blocks support the fundamental features required by developers to build microservices, for instance service invocation, state management, and publish/subscribe messaging.

Dapr abstracts these building blocks behind standard HTTP or gRPC calls, as mentioned before. It does this by providing your service with a sidecar that is accessible over HTTP or gRPC. A sidecar is a utility container in the Pod, and its purpose is to support the main container. Generally, the sidecar container is reusable and can be paired with numerous types of primary



containers. Because of this abstraction behind a sidecar, you as a developer don't need to be bothered with implementing plumbing logic, for example, publishing an event on a message bus. Your microservice just makes a call to a local API exposed by the Dapr sidecar to post an event on a message bus. The actual implementation of the message bus is not relevant to the microservice. It can even be swapped over time. The Dapr runtime/sidecar is configured with information about which logical implementation you want to use behind the Dapr APIs.

This abstraction means that publishing an event is as simple as making a call to http://localhost:51987/v1.0/publish/MyArticleEventTopic. Another example is the use of persisted data. If you need to store state inside your microservice, you just make a call passing the payload to the state-store running at http://localhost:51987/v1.0/state.



Using Dapr

There are two ways to use Dapr.
The first way is to run it locally on your machine. A prerequisite for this is the availability of Docker. After installing Docker, you install the Dapr CLI, and after it has been initialized, it makes sure that the Dapr runtime is available for you to use.

The second way to use Dapr is to install it in Kubernetes. You can use the Dapr CLI installed locally to install Dapr into your Kubernetes cluster. However, it is advisable to install Dapr into your cluster using the available Helm chart.

Programming with Dapr

While Dapr provides a nice abstraction around the functionality implemented under its API, there are also SDK's available to program. For example, to implement the actor model in a microservice you can use the provided SDK.

What features does Dapr offer?

State Management

State Management is a core capability of almost every service. Dapr provides you with a very simple-to-use state management API that allows you to do either an Http GET or POST request to retrieve or store key/value pairs into storage. As a developer, you don't have to think about what type of storage is underneath. Currently, Dapr supports Redis, Azure CosmosDB, AWS DynamoDB, GCP Cloud spanner, and Cassandra.

You might think that storing data is not that hard to implement yourself. However, in a distributed application there are several challenges you will have to consider, for instance consistency and resiliency. In addition to removing these challenges, Dapr offers a number of built-in retry policies, including an exponential back-off pattern. Consistency is also a configuration option in which you can choose between eventual consistency and strong consistency. Eventual consistency is the default option.

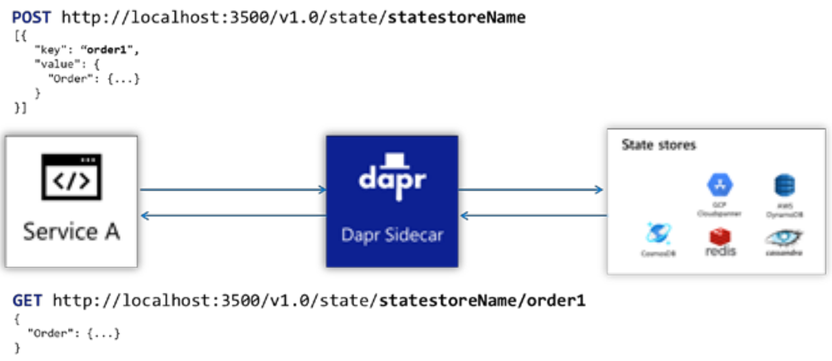
Secret Management

All applications and microservices need secrets. Dapr offers an easy way for developers to access and use secrets in their microservice. Secret management is performed in the same way as state management, without the developer knowing the underlying implementation of the secret management. It can be Azure Keyvault or Hashicorp Vault or another supported secret store.
The developer just calls the local secret API and Dapr takes care of the abstraction to the actual secret provider.

Service Invocation

Calling other services from a service is also a common scenario. Knowing where each service is running can become quite a burden. Dapr can also take this problem out of your hands. Dapr allows you to make calls to the Dapr runtime on localhost using HTTP such as GET http://localhost:<daprPort>/v1.0/invoke/<appId>/method/<method-name>. Dapr will route this request to the Dapr runtime of the other service and send back the response from the service in a similar way.

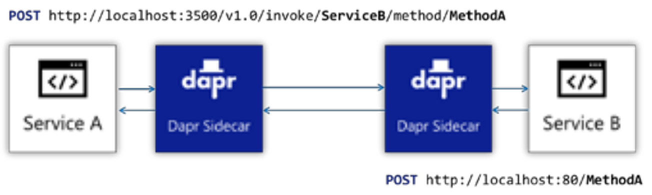
Like state management, Dapr adds several features for performing service invocations, which means that as a developer, you don't need to worry about cross-cutting concerns such as doing retries and enabling distributed tracing over multiple services.



```
POST http://localhost:3500/v1.0/state/statestoreName
[{
   "key": "order1",
   "value": {
     "Order": {...}
   }
}]
```

```
GET http://localhost:3500/v1.0/state/statestoreName/order1
{
  "Order": {...}
}
```

“Dapr is a portable, event-driven runtime that makes it easy for developers to build resilient, microservice stateless and stateful applications that run on the cloud and edge and embraces the diversity of languages and developer frameworks.”

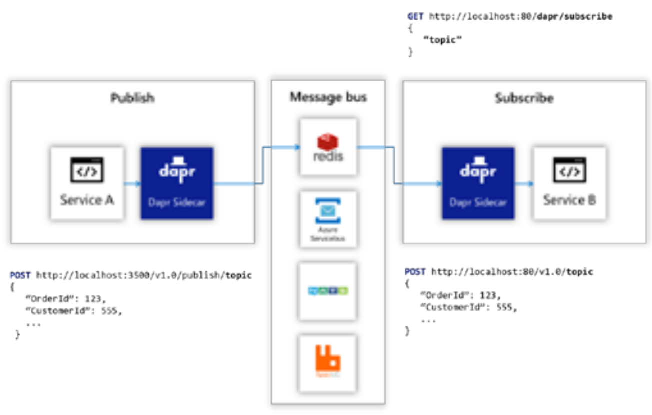


```
POST http://localhost:3500/v1.0/invoke/ServiceB/method/MethodA
```

```
POST http://localhost:80/MethodA
```

Publish / Subscribe
Publish / Subscribe is a common communication pattern in event-driven architectures and microservices architectures. The concept of publish/subscribe using Dapr is the same as with state management & service invocation. You can make an HTTP POST towards localhost to publish a message to a topic like this: http://localhost:<daprPort>/v1.0/publish/<topic>

To subscribe, you first have to do an HTTP GET to your Dapr runtime to let the Dapr runtime know you want to subscribe http://localhost:<appPort>/dapr/subscribe with the topic in the request body. After subscribing the Dapr runtime will do a POST towards your service to deliver the messages POST http://localhost:<appPort>/<topic>



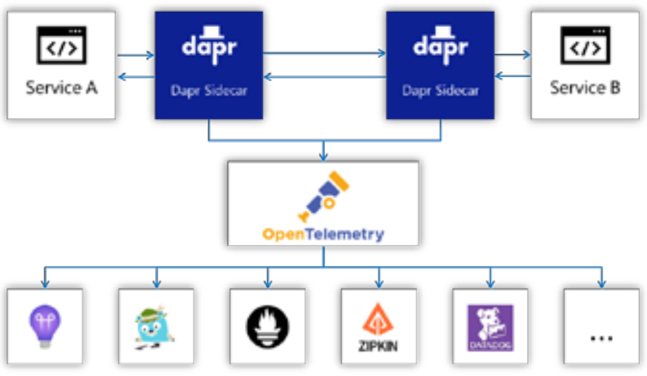
```
GET http://localhost:80/dapr/subscribe
{
   "topic"
}
```

```
POST http://localhost:3500/v1.0/publish/topic
{
   "OrderId": 123,
   "CustomerId": 555,
   ...
 }
```

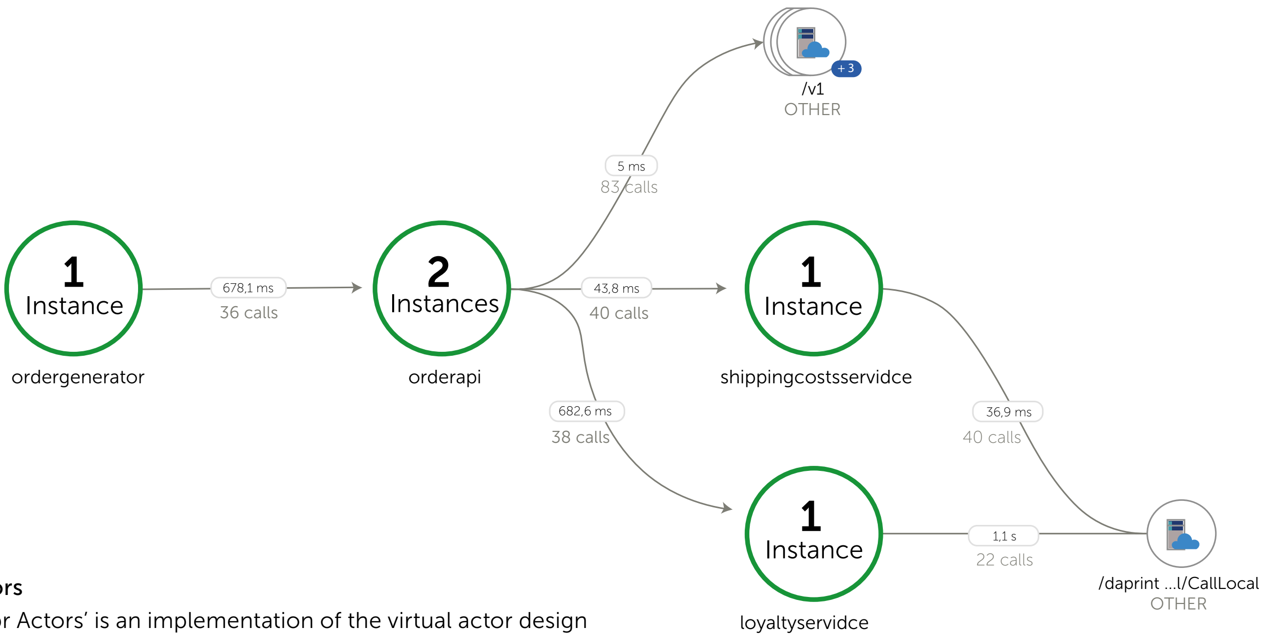
```
POST http://localhost:80/v1.0/topic
{
   "OrderId": 123,
   "CustomerId": 555,
   ...
}
```

Distributed Tracing
Finding problems in distributed applications is a difficult and tedious task. Debugging through a monolithic application is already hard, but adding multiple services in the mix makes the puzzle so much harder.

Because Dapr handles all communication through the Dapr sidecars, it is quite easy to collect all the telemetry data from this communication and gather it at a central location. Dapr can export all telemetry using the “Open Telemetry” standard, so various tracing tools such as Jaeger, Zipkin, and Application Insights can use it.



When you send this telemetry to one of these tracing tools, it makes finding problems in communication a lot easier. It also provides you with a good overview of how your app is communicating, as seen in the following picture. You can get these insights without writing any specific telemetry code yourself, which makes it very easy to obtain insight into the performance and logging of your microservices.



Actors
'Dapr Actors' is an implementation of the virtual actor design pattern. As with any software design pattern, the decision whether to use a specific pattern is made on the basis of the presence of a fitting software design problem.

Dapr actors are virtual, which means that their lifecycle is not tied to their in-memory representation. When a Dapr actor is first called with a message to its actorID, it is automatically activated and an actor object is created. These objects are removed, and garbage is collected after some period. When the actor is called again, a new object is created with the state of the previous actor object. Dapr uses the configured state management to store the Dapr actor state. Because the Dapr framework handles the state management and the (re)activation of the actors, as a developer you don't need to implement this yourself.

Future / Conclusion

At the time of writing this article, Dapr is available as a preview product. A lot of features are already available to try. Dapr decreases the complexity of building microservices. It provides a framework that abstracts the complexity delivered by a distributed microservices application. Because Dapr leverages containers and Kubernetes, it becomes portable across cloud and edge computing.

Dapr provides an extensive set of building blocks, ranging from service invocation and state management to pub/sub messaging between services. It also provides a virtual actor SDK, as well as distributed tracing between services. With the growing active community of people using and extending Dapr, more building blocks will become available in the future.

Starting with Dapr is quite simple. There is an extensive range of samples to get you going. Since everything works with HTTP, you can add it to almost any application written in any language. Give it a try and see how Dapr can help you build better microservice applications. </>

Chris van Sluijsveld
Digital disruptions using Microsoft Cloud Technology

"There is no room for complacency in the fast-moving digital world."

https://xpirit.com/xpiriter/chris-van-sluijsveld/

Geert van der Cruijsen
Digital Kickstarter, Enabler for companies to embrace DevOps, Cloud & improve their engineering culture

https://xpirit.com/xpiriter/geert-van-der-cruijsen/

Data Modeling and Partitioning in Azure Cosmos DB

Azure Cosmos DB is a massively scalable NoSQL database that works very differently than traditional relational database platforms. Rather than storing data as rows in a table with a defined schema, Cosmos DB stores data as JSON documents in a container, and there's no schema to define. There's much to learn, and for many newcomers to Cosmos DB, the learning process starts with data modeling and partitioning. How should you structure your model? When should you combine multiple entity types in a single container? Should you denormalize your entities? What's the best partition key for your data? In this article, I'll explain the key strategies for modeling and partitioning data effectively in Cosmos DB, so that you can achieve the best scale and performance for your database.

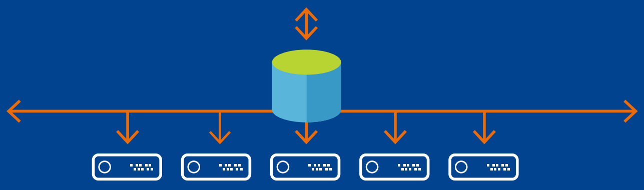
Author Lenni Lobel (Microsoft Data Platform MVP)

Many of us are familiar with relational databases like SQL Server and Oracle. But Cosmos DB is a NoSQL (non-relational) database – which is very different, and there are new ways to think about data modeling. To ease the learning curve, we'll use a real-world relational data model that you'll feel comfortable with, and then we'll refactor it as a non-relational data model for Cosmos DB.

First, there are many ways to describe Cosmos DB, but for our purposes, we can define it as having two primary characteristics: it is horizontally scalable, and it is non-relational.

Horizontally scalable

In Cosmos DB, you store data in a single logical container. But behind the container, Cosmos DB manages a cluster of servers, and distributes the workload across multiple physical machines. This is transparent to us – we never worry about these back-end servers – we just work with the one container. Cosmos DB, meanwhile, automatically maintains the cluster, and dynamically adds more and more servers as needed, to accommodate your growth. And this is the essence of horizontal scale.

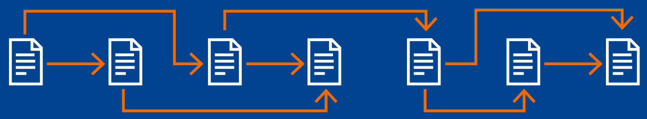


Each server has its own disk storage and CPU, just like any machine. And that means that – effectively – you get unlimited storage and unlimited throughput. There's no practical limit to the number of servers in the cluster, meaning no limit on disk space or processing power for your Cosmos DB container.

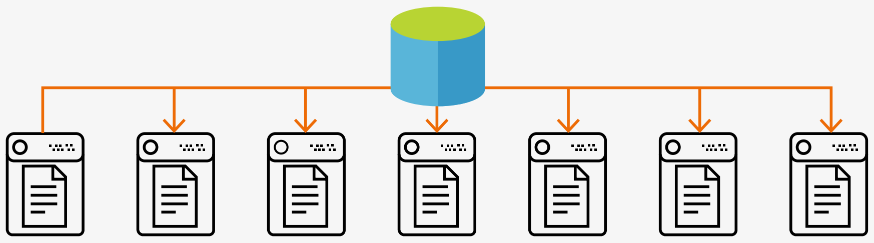
Non-relational

Think about how things work in the relational world, where we store data in rows. We focus on how those rows get joined along primary and foreign keys, and we rely on the database to enforce constraints on those keys.

But in the non-relational world, we store data in documents – that is JSON documents. Now there's certainly nothing you can store in a row that you can't store in a JSON document, so you could absolutely design a highly normalized data model with documents that reference other documents on some key, like so:



Unfortunately, this results in a very inefficient design for Cosmos DB. Why? Because again, Cosmos DB is horizontally scalable, where documents that you write to a container are very likely to be stored across multiple physical servers behind the scenes:



Although it is technically possible to enforce relational constraints across a cluster of servers, doing so would have an enormous negative impact on performance. And well, "speed and performance" is the name of the game in Cosmos DB, with comprehensive SLAs on availability, throughput, latency, and consistency. Reads and writes have extremely low, single-digit millisecond latency – meaning that they complete within 9 milliseconds or less in most cases. So, in order to deliver on these performance guarantees, Cosmos DB simply doesn't support the concept of joins, and can't enforce relational constraints across documents.

Suitable for relational workloads?

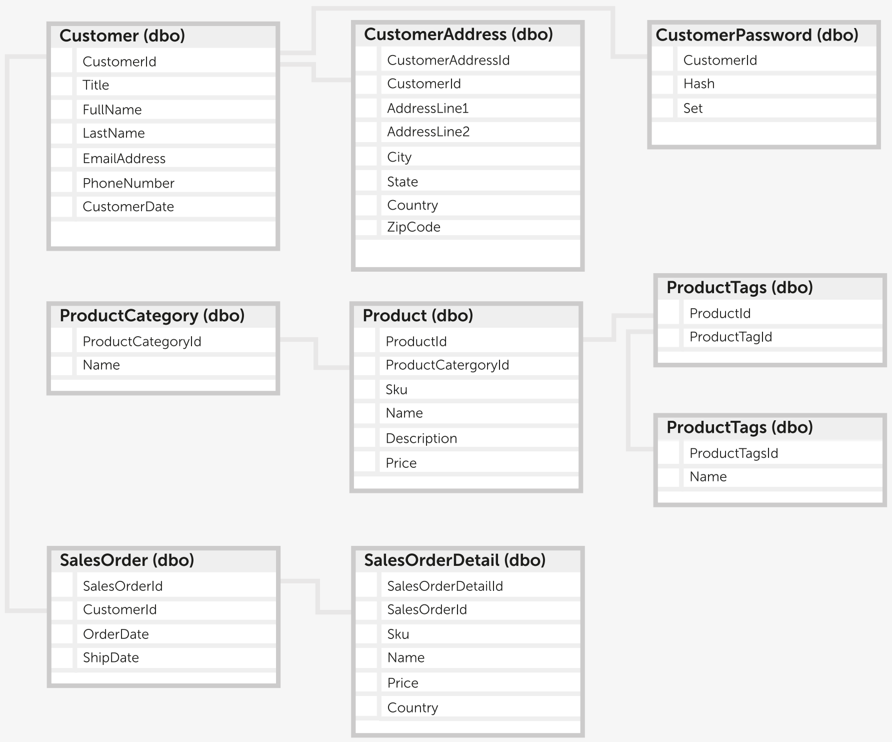
With no joins and no relational constraints, the obvious question becomes: "Is Cosmos DB suitable for relational workloads?" And the answer is, yes, of course it is. Otherwise, the article would end right here.

And when you think about it, most real-world use cases are relational. But because Cosmos DB is horizontally scalable and non-relational, we need to use different techniques to materialize relationships between entities. And this means a whole new approach to designing your data model. In some cases, the new methods are radically different, and run contrary to best practices that some of us have been living by for decades.

Fortunately, and I hope by the end of this article you'll agree, while it is very different, it's not really very difficult. It's just that, again, you need to think about things differently when designing your data model for Cosmos DB.

WebStore relational model

Our sample database is for an e-commerce web site that we're calling WebStore. Here is the relational data model for WebStore in SQL Server:



This data model is relatively small, but still representative of a typical production model. It has one-to-many relationships, like the ones from Customer to CustomerAddress and SalesOrder. There is also a one-to-one relationship from Customer to CustomerPassword, and the Product-Tags table implements a many-to-many relationship between Product and ProductTag.

Container per table?

It's very natural at first to think of a container in Cosmos DB like a table. Your first instinct may be to say, OK, we have nine tables in our data model, let's create nine containers in Cosmos DB.

Now you can certainly do this, but again, this would be the worst possible design, being horizontally scalable and non-relational. Cosmos DB will expose no way to join documents, or to enforce relational constraints between them. Therefore, this approach will not only perform poorly, but it will be very difficult to maintain and program against.

What's the answer? Let's get there one step at a time.

Embed vs. reference

Let's start with customers and their related entities. JSON is hierarchical, so we don't need separate documents for every type. So now think about the distinction between one-to-many and one-to-few. It's reasonable to impose an upper limit on the number of addresses a customer can have, and there's only one password per customer, so we could combine all of those into a single document.

This has immediately solved the problem of joining between customers and their addresses, and passwords, because that's all "pre-joined" by embedding the one-to-few relationship for addresses as a nested array, and the one-to-one relationship for the password as an embedded object. Simply by embedding, we've reduced three relational tables to a single customer document.

```
{
  "id" :  " ...",
  "title" :  " ...",
  "firstName" :  " ...",
  "lastName" :  " ...",
  "emialAddress" :  " ...",
  "phoneNumber" :  " ...",
  "creationDate" :  " ...",
  "addresses" :  }
    {
      "addressLine1" :  " ...",
      "addressLine2" :  " ...",
      "city" :  " ...",
      "state" :  " ...",
      "country" :  " ...",
      "zipCode" :  " ..."
    }
  ],
  "password" :  {
    "hash" :  " ...",
    "salt" :  " ...",
  }
}
```

On the other hand, we would certainly not want an upper limit on the number of sales orders per customer – ideally, that is unbounded (while the maximum document size is 2 MB). The orders will be stored in separate documents, referenced by customer ID.

The rules for when to embed and when to reference are simple, as we're demonstrating. One-to-few and one-to-one relationships often benefit from embedding, while one-to-many (particularly unbounded) and many-to-many relationships require that you reference. Embedding is also useful when all the entities are typically queried and/or updated together (for example, a customer profile with addresses and password), while it's usually better to separate entities that are most often queried and updated separately (such as individual sales orders).

The next – and arguably most important – step is to choose a partition key for the customer document. Making the right choice requires that you understand how partitioning works.

Understanding partitioning

When you create a container, you supply a partition key. This is some value in your documents that Cosmos DB groups documents together by in logical partitions. Each server in the cluster is a physical partition that can host any number of logical partitions, each of which in turn stores any number of documents with the same partition key value. Again, we don't think about the physical partitions, we're concerned primarily with the logical partitions that are based on the partition key that we select.

All the documents in a logical partition will always be stored on the same physical partition; a logical partition will never be spread across multiple servers in the cluster. Ideally, therefore, you want to choose a partition key whose value will be known for most of your typical queries. When the partition key is known, then Cosmos DB can route your query directly to the physical partition where it knows all the documents that can possibly satisfy the query are stored. This is called a single-partition query.

If the partition key is now known, then it's a cross-partition query (also often called a fan-out query). In this case, Cosmos DB needs to visit every physical partition and aggregate their results into a single resultset for the query. This is fine for occasional queries, but adds unacceptable overhead for common queries in a heavy workload.

You also want a partition key that results in a uniform distribution of both storage and throughput. A logical partition can't exceed 20 GB, but regardless, you don't want some logical partitions to be huge and others very small. And from a throughput perspective, you don't want some logical partitions to be heavily accessed for reads and writes, and not others. These "hot partition" situations should always be avoided.

Choosing a partition key

With this understanding, we can select a partition key for the customer documents that we'll store in a customer container. The question is always the same: "What's the most common query?" For customers, we most often want to query a customer by its ID, like so:

```
SELECT * FROM c WHERE c.id =
'<custId>'
```

In this case, we want to choose the id property itself as the partition key. This means you'll get only one document in every logical partition, which is fine. It's desirable to use a partition key that yields a large spectrum of distinct values. You may have thousands of logical partitions for thousands of customers, but with only one document each, you will achieve a highly uniform distribution across the physical partitions.

We'll take a very different approach for product categories. Users visiting the website will typically want to view the complete list of product categories. Then, they'll want to query for all the product that belong to a category that interests them, which is essentially a query on the product category container with no WHERE clause. The problem though, is that would be a cross-partition query, and we want to get all our category documents using a single-partition query.

The trick here is to add another property called type to each product category document, and set its value to "category" in every document. Then we can partition the product category container on the type property. This would store all the category documents in a single logical partition, and the following query could retrieve them as a single-partition query:

```
SELECT * FROM c WHERE c.type =
'category'
```

This same is true of tags; users will typically want a full list of tags and then drill to view the products associated with interesting tags.

This is a typical pattern for short lookup lists that are often retrieved all at once. So that would be another container for product tags, partitioned on a type property where all the documents have the same value "tag" in that property, and then queried with:

```
SELECT * FROM c WHERE c.type =
'tag'
```

Many-to-many relationships

Now for the many-to-many relationship between products and tags.
This can be modeled by embedding an array of IDs on one side or the other; we could either store a list of tag IDs in each product, or a list of product IDs in each tag. Since there will be fewer products per tag than tags per product, we'll store tag IDs in each product document, like so:

```
{
  "id" :  "...",
  "categoryId" :  "...",
  "sku" :  "...",
  "name" :  "...",
  "description" :  "...",
  "price" :  "...",
  "tagIds" :  [
    "...",
    "...",
  ]
}
```

Once the user chooses a category, the next typical query would be to retrieve all the products in a given category by category ID, like so:

```
SELECT * FROM c WHERE
c.categoryId = '<catId>'
```

To make this a single-partition query, we want to partition the product contain on the product category ID, and that will store all the products for the same category in the same logical partition.

Introduction denormalization

Now we have a new challenge, because product documents hold just the category ID and an array of tag IDs – it doesn't have the category and tag names themselves. And we already know that Cosmos DB won't join related documents together for us. So, if we want to display the category and tag names on the web page – which we do – then we need to run additional queries to get that information.

First, we need to query the product category container to get the category name, and then – for each product in the category – we need to query the product tag container to get all the tag names.

We definitely need to avoid this, and we'll solve the problem using denormalization. And that just means that – unlike in a normalized data model – we will duplicate information as necessary in order to make it more readily available for queries that need it. That means that we'll store a copy of the category name, and copies of the tag names, in each related product document:

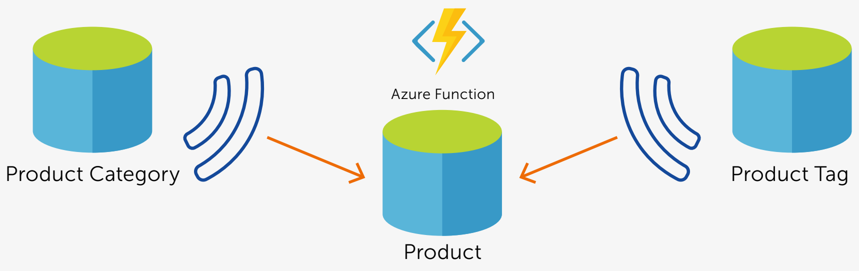
```
{
  "id" :  "...",
  "categoryId" :  "...",
  "categoryName" :  "...",
  "sku" :  "...",
  "name" :  "...",
  "description" :  "...",
  "price" :  "...",
  "tags" :  [
    {
      "id" :  "...",
      "name'" :  "..."
    },
    {
      "id" :  "...",
      "name" :  "..."
    }
  ]
}
```

Now we have everything we need to display about a product self-contained inside a single product document.
And that will work great, until of course, there's a category name or tag name is changed. Because now we need a way to cascade that name change to all the related copies in order to keep our data consistent.

Denormalizing with the Change Feed

This is a perfect situation for the Cosmos DB Change Feed, which is a persistent log of all changes made to any container. By subscribing to the change feed on the category and tag containers, we can respond to updates and then propagate the change out to all related product documents so that everything remains in sync.

This can be achieved with a minimal amount of code, and implemented out-of-band with the main application by deploying the change feed code to run as an Azure function:



Any change to a category or tag name triggers and Azure function to update all related product documents.
This lets us maintain a denormalized model that's optimized to retrieve all relevant information about a product with one single-partition query.

Combining different types

The last part of our schema are the customer orders and order details.
First, we'll embed the details into each order as a single document for the sales order container, because that's another one-to-few relationship between entities that are typically retrieved and updated together.

It will be very common for customers to retrieve their orders using the following query:

```
SELECT * FROM c WHERE
c.customerId = '<custId>'
```

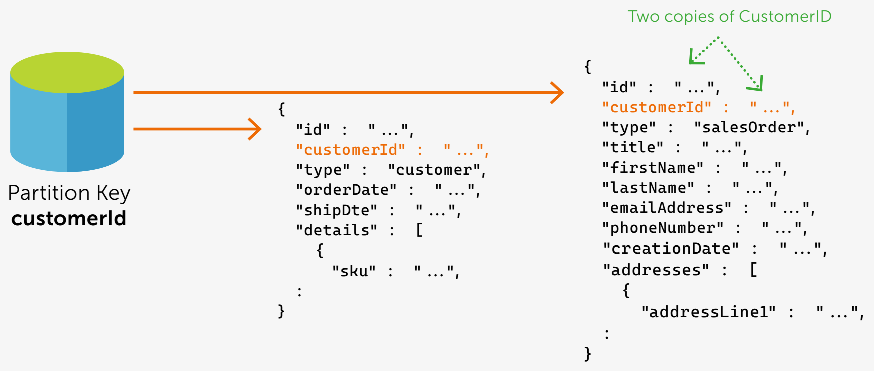
That makes the customer ID the best choice for the partition key. But before we jump to create another container for sales orders, remember that we're also partitioning customers on the customer

ID in the customer container. And unlike a relational database where tables have defined schemas, Cosmos DB lets you mix different types of documents in the same container. And it makes sense to do that when those different types share the same partition key.

We'll combine customer and sales order documents in the same customer container, which will require just a minor tweak to the customer document. We'll need to add a customerId property to hold a copy of the customer ID in the id property. Then we can partition on customerId which will be present in both document types:

Notice that we've also added a type property to distinguish between the two types of documents. So now, there is still only one customer document per logical partition, but each logical partition also includes the related orders for that customer. And this kind of gets us our joins back, because now we can retrieve a customer and all their related orders with the following single-partition query:

```
SELECT * FROM c WHERE c.id =
'<custId>'
```



```
{
  "id" :  " ...",
  "customerId" :  " ...",
  "type" :  "customer",
  "orderDate" :  " ...",
  "shipDte" :  " ...",
  "details" :  [
    {
      "sku" :  " ...",
  :
}
```

```
{
  "id" :  " ...",
  "customerId" :  " ...",
  "type" :  "salesOrder",
  "title" :  " ...",
  "firstName" :  " ...",
  "lastName" :  " ...",
  "emailAddress" :  " ...",
  "phoneNumber" :  " ...",
  "creationDate" :  " ...",
  "addresses" :  [
    {
      "addressLine1" :  " ...",
  :
}
```

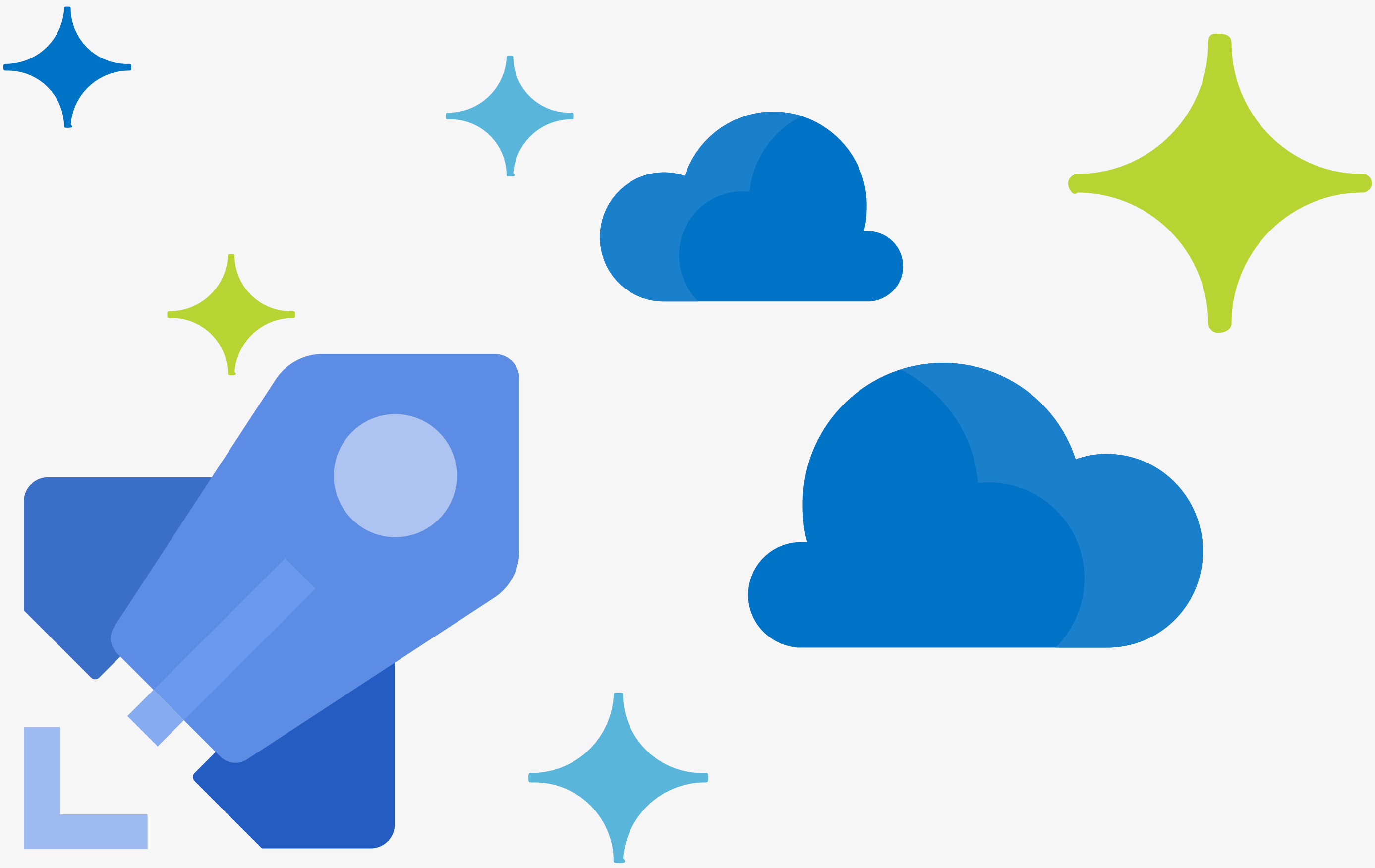
Denormalizing with a Stored Procedure

Let's wrap up with one more query to retrieve our top customers; essentially, a list of customers sorted descending by order count. In the relational world, we would just run a SELECT COUNT(*) on the Order table with a GROUP BY on the customer ID, and then sort descending on that count.

But in Cosmos DB, the answer is once again to denormalize. We'll just add a salesOrderCount property to each customer document. Then our query becomes as simple as:

```
SELECT * FROM c WHERE
c.type = 'customer'' ORDER BY
c.salesOrderCount DESC
```

Of course, we need to keep that salesOrderCount property in sync; each time we create a new sales order document, we also need to increment the salesOrderCount property in the related customer document. We could use the change feed like before, but stored procedures are a better choice when your updates are contained to a single logical partition.



In this case, the new sales order document is being written to the same logical partition as the related customer document. We can write a stored procedure in JavaScript that runs within the Cosmos DB service which creates the new sales order document and updates the customer document with the incremented sales order count.

The big advantage here is that stored procedures in Cosmos DB run as a transaction that succeeds or fails as a whole. Both write operations will need to complete successfully or they both roll back. This guarantees consistency between the salesOrderCount property in the customer document and the true number of related sales order documents in the same logical partition.

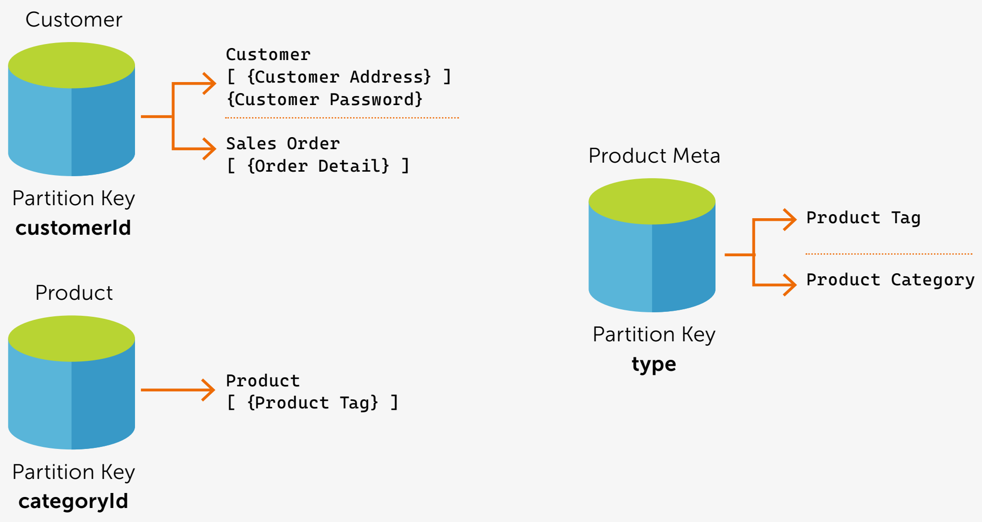
One last thing to mention is that this is a cross-partition query, unlike of our previous examples, which were all single-partition queries. Remember again that cross-partition queries aren't necessarily evil, as long as they aren't very common. In our case, this last query won't run routinely on the website; it's more like a "back office" query that an executive runs every now and again to find the top customers.

Summary

This article has walked you through the steps to refactor a relational data model as non-relational for Cosmos DB. We collapsed multiple entities by embedding, and we support denormalization through the use of the Change Feed and stored procedures.

We also combined customer and sales order documents in the same container, because they are both partitioned on the same value (customer ID). To wrap up the design, we can also combine the product category and product tag documents in a single "product meta-data" container, since they are both partitioned on the same type property. That brings us to our final design shown below.

Using a combination of non-relational modeling techniques, we've reduced nine tables into just three containers, where majority of queries run by the application are all scoped to a single logical partition. This article has given you the information you need to succeed in designing the optimal non-relational data model for your next Cosmos DB application. </>



Lenni Lobel
Microsoft Data Platform MVP

"Specialized in Microsoft-based solutions, with experience that spans a variety of business domains."

Feature toggles in favor of continuous deployment

Why you want feature toggles and what type of toggle to use? Feature toggles or feature flags are techniques for hiding feature implementations from your customer until you want your customer to experience your new feature. This technique helps to overcome all kinds of challenges during your software development. There are many different types of feature toggles and different ways to implement them. Depending on whether you build your own toggles or use a framework, a number of toggles will be provided out of the box. Many frameworks also support your custom implementation of a feature toggle. However, before using and building your own toggles, it is essential to understand a number of possible scenarios and when to use a particular toggle. This article will provide insight into a variety of toggles and how to choose the most suitable toggle for your requirements.

Author Erick Segaar

Feature toggle categories

Feature toggles come in different shapes and sizes, and the implementation can be anything you want it to be. You can categorize your toggle in one of the following categories, each of which has its distinct purpose and criteria;

- Release, this toggle can change per release and usually has a life span of days to weeks. Once the code is deployed, you don't want to flip it because it might activate unwished code or unexpected behavior.
- Operations, this toggle can be anywhere from short- to long-lived, often managed by an operations group in order to have a way to control software in a tested and controlled manner.
- Experiment, this toggle is short-lived, and due to the nature of experiments, you don't want it to be activated too long because it will distort the results of your experiment with other code changes.
- Permission, you're probably already using this toggle to provide access to a cohort or targeted user to use a closed-down part of your software. The life span can be anywhere from short- to long-lived.

It is important to understand what you want to do with your toggle and what category it belongs to. This will help you to understand what you are implementing and why you are implementing a specific piece of code.

The following overview shows the various categories with examples of some of the most frequently used implementation types.

Feature toggle categories

Release	Operations	Experiment	Permission
• Deploy Incomplete code • Release time	• Performance related • Manual circuit breakers	• Testing scenarios • Dynamic (User or request based)	• Targeted users • Defining cohorts • Dynamic (User or request based)
Short lived	Short to long-lived	Short lived	Medium to long-lived
• On • Off • Percentage	• On/Off • Kill-switch	• Precentage • Time Window	• Claims • Cookies

Separating deployment from release

Although you are in full control of everything required to develop your application in an ideal situation, the reality is that quite often you're dependent on other teams, products, or schedules. The way we manage this traditionally is to have meetings and agreements with all teams and products involved. We agree upon a plan to deliver a product to our customers when everything is finished, tested, and 100% ok. Despite the fact that this looks great on paper with some helpful "Gantt charts", it could not be further from the truth. Not only are we humans bad in planning complex work, but our environment is also changing continuously, and we might have time to develop this new feature now, but in two weeks' time something else is likely to be more important. All this time, the constraints you have with other teams, products, and schedules need to be managed, and this time cannot be used to develop more new and exciting features.

By separating the deployment of features from exposure to your customers, you can mitigate the dependencies between the technical part and the business part; i.e. between the technical phase of deploying and shipping the product, and the business phase of exposing and releasing new features. By separating these phases, you enable yourself to keep developing and deploy functionality independently from other teams or products because it is hidden from your end-users.

The functionality of an ON/OFF toggle is a typical operation toggle that gives you the ability to enable or disable the feature, and would be a simple way to implement. While developing your functionality, you can use the ALWAYS-OFF toggle from the release category because you don't want your feature to be activated by accident. But when your part of the work is finished, use a more dynamic toggle to activate it when the business is ready for it.

Code branch management

When you work on features, there can be quite a time gap between start and finish. During the development phase we are often tempted to create a separate branch that allows us to develop in isolation. Branching protects you from changes by others or other teams because they 'won't reflect into your branch until you update it. However, the biggest pitfall is that you are also delaying yourself from receiving feedback on your changes. You especially want constant feedback on big and complex changes that take a lot of time. And you certainly don't want to wait until the end, when you finish your significant changes and realize the product has changed too much to easily integrate it back into the master version.

Using feature flags allows you to wrap the new application functionality that is under development. Such functionality is "hidden" by default. You can safely ship the feature, even with the unfinished work, because it will stay dormant during production. By using this approach, called "dark launch," you can release all your code at the end of each development cycle. You no longer need to maintain any code branch across multiple cycles because of the feature taking more than one cycle to be completed. Just keep working on the master following trunk-based development together with your team.

A complex branching strategy is often over-engineered with only a few people that truly understand it. Prevent merging issues and follow the continuous integration principle by continuing to work on the master branch instead of hiding your work in a separate branch. Use an ALWAYS ON/OFF toggle until you are finished so you can keep in sync with your colleagues and enable fast peer feedback by exposing your changes.

Testing in production

When you develop a new feature, you want to shorten the feedback cycle from your customer as fast as possible. Knowing that you are building the right thing right is an advantage you get with short development cycles and fast customer feedback. You can start to experiment and validate the hypothesis you think your customers want.

Feature flags allow you to grant early access to new functionality in production. For example, you can limit the access to only development team members or some internal beta testers. This technique is called “Ring-based deployment,” and provides users with the full-fidelity production experience instead of a simulated or partial one in a test environment. It gives you the much needed and fast feedback without the need for a production-like environment. After all, this is always a lot of trouble to set up and maintain, especially when you are handling personal data and need to be GDPR-compliant, which is a struggle many organizations are facing these days.

Use a PERMISSION TOGGLE implementation to combine the newly developed feature with your claims system to grant specific groups access to the functionality being developed quickly. First set it for your development team and later extend it to early adapters.

Flighting

After weeks of developing, testing, and validating our test environments, the time finally arrives to release the world-changing feature to customers. To obtain full exposure, your marketing team sent an email to your customers so they can all try out your new feature. This scenario is not uncommon in the industry. However, all too often the result consists of unresponsive webpages or customers facing long queuing or no experience at all. Correct estimations of the number of customers hitting your new feature and the required underlying resources are hard to make. And we ’don’t want to show up on the news with negative publicity.

By using a flighting mechanism, you can incrementally roll out new functionality to your end-customers. Start by targeting a small percentage of your user population and gradually increase that percentage over time, after you have gained more confidence in the implementation and the use of your feature. When something goes wrong, only a small part of your customer base would be affected, and you can monitor the capacity of your resources closely while ramping up the number of customers.

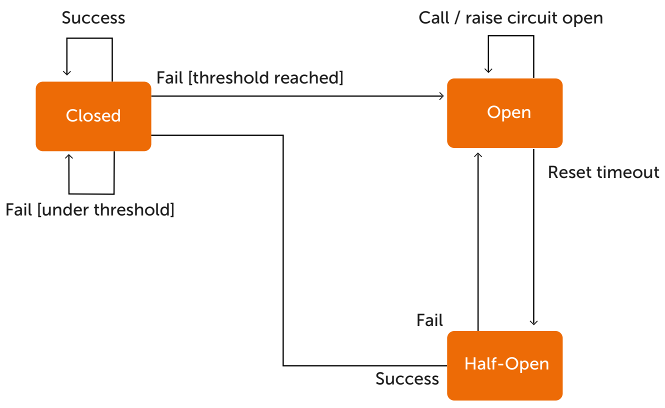
A typical flighting lifecycle starts as an ALWAYS OFF toggle. When the feature is complete and ready to be exposed to your customers, you can either use a PERMISSION toggle, if you want to control who gains access, or a PERCENTAGE toggle which you can ramp from 5% to 100% in as many steps as you feel comfortable.

Instant kill switch

Your feature is available to your customers, and suddenly an unexpected behavior arises that is costing the enterprise money every second it is enabled. The error could come from a part you control or a dependency upon another service. For example, a payment service that you are requesting for different payment options from a specific bank is returning status 500 and giving your customers no option to pay as a result.

When you are dependent upon externally controlled services, it is advisable to think about how you want your application to react when that service is down or behaving unexpected-ly. After all, they are just another product like your own, and mistakes and unwanted responses will occur. But even when your application is entirely under your control, you want to think about how you want your application to react in case of errors. For example, when the authentication service of Netflix is down, they grant access to everyone instead of blocking everyone. From their perspective, they made a mistake, and the customer should not have to pay for it, which could also lose them business and reputation.

Although the toggle implementation is just a simple ON/OFF toggle, the mindset and duration of that toggle are different compared to the “Separate Deployment from Release.” This toggle is an operational toggle intended to be controlled manually when production needs it. A potent addition to this toggle is to combine it with a circuit breaker strategy.



”A BIG BANG can turn out painful! Continuous Deployments ease the transition.”

Whenever a dependency returns an unexpected response within a timeframe, the circuit breaker will trigger, i.e. toggle the feature toggle. Doing this will prevent your application from being affected by errors from the external service, and your customers will enjoy your service even though the service will probably show a certain level of degraded performance or lesser functionality, but to prevent it from being offline. In addition, you can give the remote service some breathing space to recover or start-up additional instances without being hammered constantly by your application. Every X-amount of time, the circuit breaker will try one or two requests to check whether the external service is recovered. When the result is positive, the circuit breaker will close, meaning the toggle will flip again, and all traffic will flow to the external service, once again enabling full functionality for your customers.

Selective activation

Think of a scenario in which you are developing a new feature for every web browser. A custom implementation is needed and your feature is dependent upon that implementation to work correctly. You prioritized Chrome to be the most important and want to expose it to be enabled, but only for customers with the correct version of Chrome.

The focus on one implementation allows you to receive production feedback fast and quickly for the part of your feature that is finished. Moreover, you can expand the list of supported browsers in the future. You could also use this to inform your customers of the browsers that you support with additional functionality.

Implementation-wise this toggle will validate whether the 'visitor's web browser version occurs in a list of values stored in the feature toggle.

Life-span

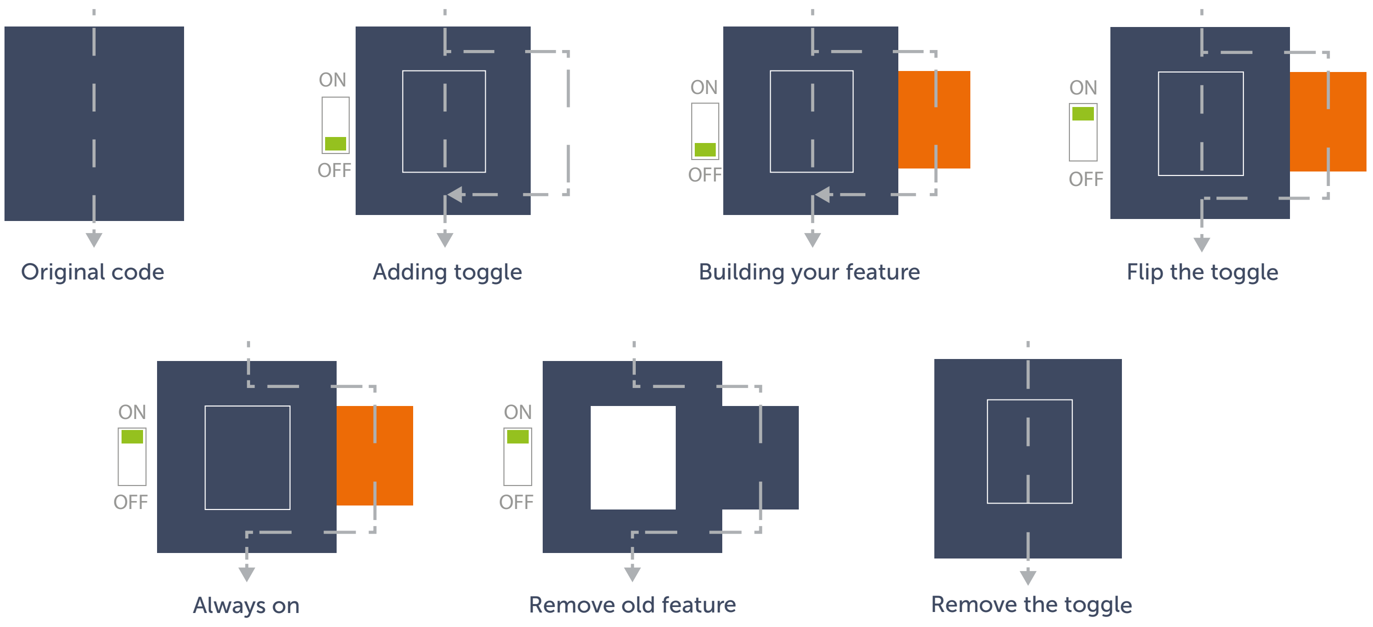
While there are all kinds of toggles, you can use it in different scenarios. The life-span can vary from short-lived to very long-lived. One thing that all toggles have in common is the way you implement the toggle and that you eventually need to remove the toggle.

"Savvy teams view their Feature Toggles as inventory, which comes with a cost, and work to keep their inventory as low as possible."
Martin Fowler

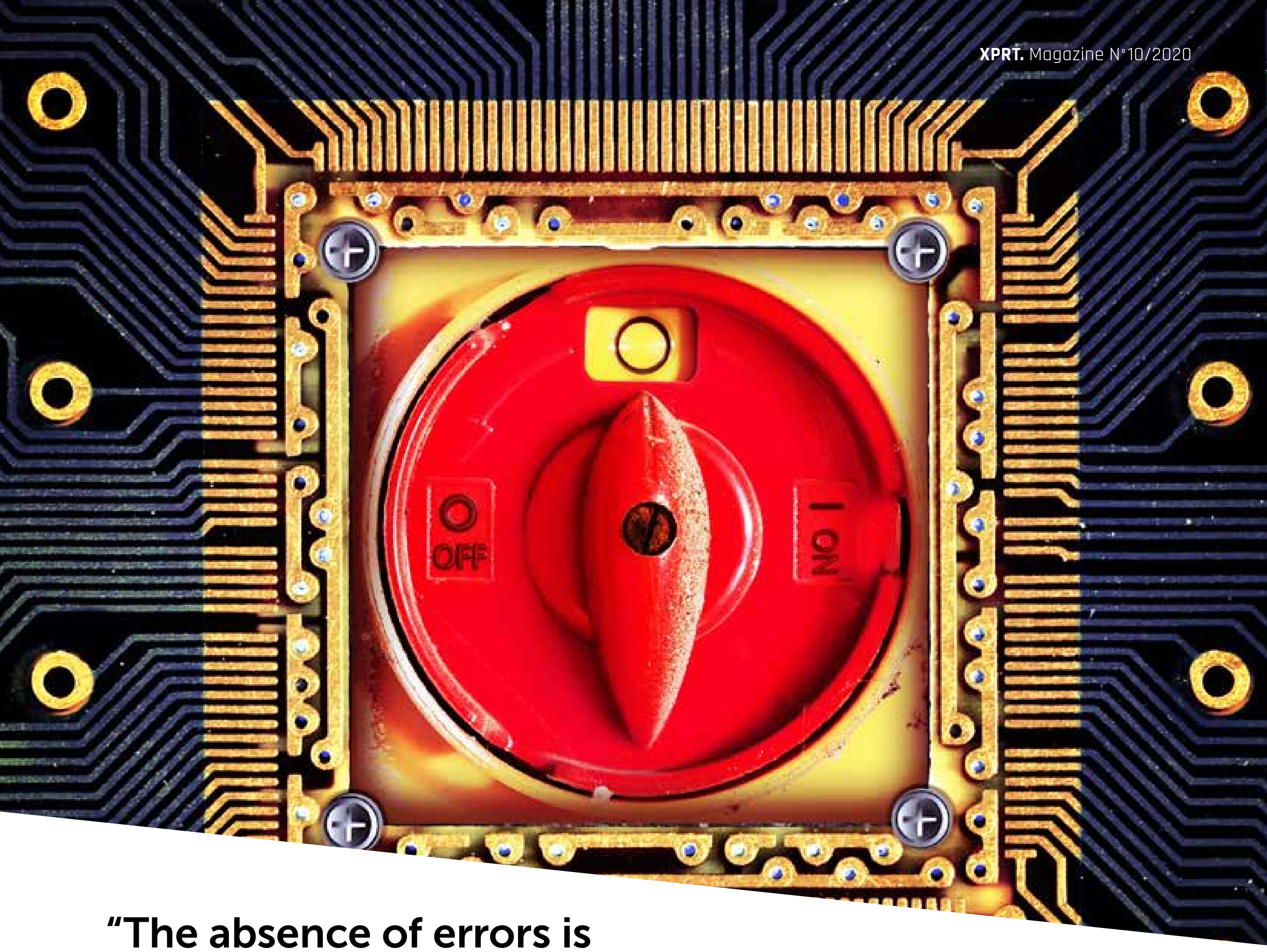
We start with our original code, and we decide that we need a feature toggle. Next, we create the toggle, we think about where to place it on the correct level and implement it as an ALWAYS OFF toggle. Now, everything is set up to hide the development of your new feature, and you can start building it. Eventually, you can change your ALWAYS OFF toggle to become an ON/OFF toggle and flip the switch so people can experience your new feature. When your feature is running the way it should run, and it has not been turned off recently, you have your ALWAYS ON toggle, and now you can start removing the old feature resulting in the removal of your feature toggle.

Monitoring

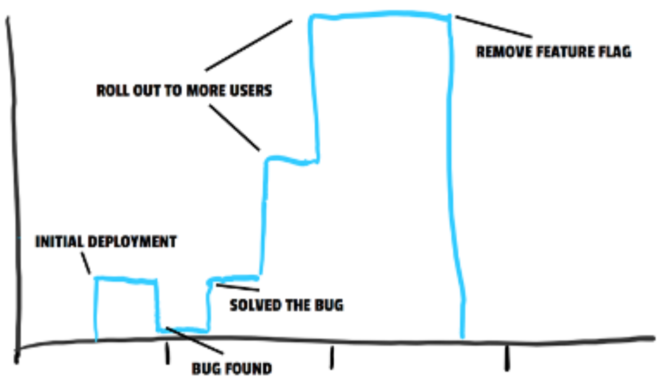
In addition to the technical implementation, you want to monitor your application and the usage of your feature toggle. Every time you enable a feature, you should treat it as a deployment. Have an increased awareness of exceptions and watch your monitoring closely to identify unwanted behavior. You want to know whether people are using your new feature and how many people are using it. If nobody is using your feature, it will not send any exceptions either.



Overview feature toggle implementation



"The absence of errors is not good enough."



Number of users over time that use the toggle

In the diagram above, you can see that the initial deployment enabled our new feature, and we see an increase in the number of users, limited to a particular control group. Upon finding a bug, we close the feature down to solve it. We fix the bug and re-deploy it and see the same control group as an increasing activity. When we feel more confident, we extend the feature to be available to more customers. Eventually, the toggle is running in production for a time, and you remove the feature.

Smells and pitfalls

The opportunities provided by feature toggles make you think that this must be a silver bullet! Well, not exactly. Feature toggles are a means to an end and not a goal in itself. By not understanding the use and complexity of feature toggles, you could do serious harm to your product. The following section contains an unordered set of smells and pitfalls that can help you recognize and understand the risky situations we experienced during our years of development.

Too many feature toggles

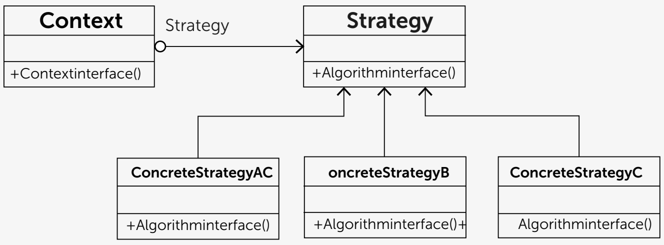
It is difficult to explain exactly what number is too many, but this is a sign that something is not right in your understanding or implementation. The amount can differ per product, team, and experience. Keep track of all your toggles and keep the number limited.

Fine-grained feature toggles

Fine-grained toggles give you a lot of control but also give you many toggle combinations to test alone and in combination. Keep toggles on entire features and keep them simple. The fewer combinations you need to validate, the less prone you are to making errors.

Toggle on technical capabilities instead of business process
Feature toggles should be easy to translate to business processes and capabilities. When you are implementing technical toggles, they should end up in the hands of your operator, or they should be very short-lived.

Same toggle used in multiple places
Toggles that are used in multiple places and that don't have having a single place of entry could indicate that you need to place your toggle on a higher level, or maybe you need a strategy pattern[1] to inject behavior. This helps you to keep a clear overview of where a toggle is used and reduces the number of code paths.



Forgetting to describe what your toggle does
A description like: "This toggle routes traffic to the new score calculator engine, when the toggle is off, the old legacy one will be used and can cause latency bugs to appear." It will work so much better than a description like ft-calc-engine.

"A good convention is to enable existing or legacy behavior when a Feature Flag is Off and new or future behavior when it's On."
Martin Fowler

Toggles are technical debt
All toggles are by nature technical debt and should be treated as such. They should be removed when they are no longer needed. The removal of toggles is a continuous part of refactoring your code and the cost you are paying for using feature toggles.

Launching blindly
Launching blindly is nothing more than throwing your application over the fence. Even when you deploy, turning a toggle on or off without some kind of monitoring is irresponsible, and you won't know whether the deployment succeeded.

Unseparated Control
When you manage your toggles from the same product you are controlling, toggle management might be unresponsive when a toggle enables and is harmful to memory or CPU. However, you cannot turn it off now.

Long-lived toggles
By nature, long-lived toggles present technical debt and should be removed. The longer a toggle is in your system, the higher the risk of combining multiple toggles, adding complexity to your code path.

Re-using a feature toggle
A feature toggle 'shouldn't be re-used, it's a one-time implementation with history and auditing. Once it is refactored out, you should not re-use the name because this would only create confusion.

Conclusion
In this article, I have tried to provide an overview of a variety of scenarios for applying feature toggling. There are many more scenarios and they often involve chaos engineering (i.e. the discipline of experimenting on a software system in production in order to build confidence in the system's capability to withstand turbulent and unexpected conditions). Although it is almost a textbook explanation of how to use feature toggles, I often see this technique misused and thus it undermines the confidence of the team and product owner. As a result, they maintain the status quo of releasing once a month or even less frequently. I deliberately stayed away from the technical implementation; this depends heavily on the framework you choose to work with or build your own. If you want to have a starting point, you can take a look at "Azure Application Configuration Feature management2." This gives you a number of out-of-the-box toggles, local use of feature toggles, as well as a good cloud platform to run it for you". The take-away of this article is that feature toggles are business-driven, allowing you to separate the use of functionality from its technical deployment. Using the mantra, "A BIG BANG can turn out painful! Continuous Deployments ease the transition " Use this to your advantage and use the 'smells and pitfalls' to help you recognize a hard to maintain set-up or an error-prone method. </>

Erick Segaar
Coaching, analytical Scrum, CI/CD, ALM, People-first Mindset

"I have never tried that before, so I think I should definitely be able to do that."

https://xpirit.com/xpiriter/erick-segaar/

[1] https://en.wikipedia.org/wiki/Strategy_pattern

Treat your VM like a Container

In the current days of serverless and k8s, you may forget about the dark corners in the cloud that are filled with legacy and cloud-not-so-native applications. Automatic deployments are the norm nowadays, and Infrastructure as Code (IaC) has been on the rise. What if we told you that it is possible to do the same for your age-old applications and get almost the same benefits?

Authors Manuel Riezebosch & Arjan van Bekkum

Machine Images

The beauty of machine images is that they enable you to create Virtual Machines in a repeatable manner and add new instances in a minimum of time. Both Azure and AWS have the so-called notion of images: Managed Image on Azure and Amazon Machine Image (or: AMI) on AWS. Deploying new VMs (or EC2 Instances) using images is already possible with the given infrastructure. You can even take this a step further by deploying Virtual Machine Scale Set or use AWS Auto Scaling to create and destroy instances on demand.

Build Your Own

For speeding up and stabilizing deployment it is best to create your own hand-rolled images with
customizations that you need in every instance. The normal procedure for creating such an image is:

1. Spin up a fresh VM
2. Perform your customizations, either remote or from login
3. Generalize the machine (sysprep for Windows)
4. Shutdown the VM
5. Capture the disk.

Packer helps you by automating this process. By doing so, it enables you to set up a CI/CD pipeline which ensures that the image is build, stored, and deployed on the cloud provider of your choice.

Packer uses packer templates, which contain all the configuration and instructions to build an image.
Building images like this is like adding layers to a docker image, except for the storage part. You can even reuse your custom image to be used as the input of another packer build.

Builders

The template contains the builders with the configuration for the target platform(s) on which you want to create an image.

This article will show you how to do this for both Azure and AWS. On Azure you use the 'azure-arm' builder and on AWS the 'AMI builder'. In short: the builder is the configuration for your target platform for the intermediary VM that is used, and the target location of the image that is the build output.

Provisioners

The other part of the template is about the provisioners. Provisioners are the things you use to interact with the VM. For example, it can execute some script running inside the VM to install some software. Another provisioner will copy some content to the VM or download it from the VM. In short, provisioners are about customizing the intermediate VM from which the final image is built.

Installing software

The first thing you want to do when building a Windows Server image is to install the package manager, for instance chocolatey or winget. Having a package manager at your fingertips will greatly reduce the time and effort spent on hand-rolling installation scripts.

For now we will use chocolatey, but the idea is the same for winget. To install chocolatey you just follow the regular installation instructions from the chocolatey website https://chocolatey.org/install. Put that in a script and invoke it from a provisioner.

```
iex ((New-Object System.Net.WebClient)
  .DownloadString('https://chocolatey.org/install.ps1'))
```
Install-Chocolatey.ps1

```
"provisioners": [{
    "type": "powershell",
    "scripts": [
      "{{ template_dir }}/Install-Chocolatey.ps1"
    ]
}]
```
packer-template.json

After that, most installation scripts will look like this:

```
choco install awscli -y
```

It is tempting to put all these installations in an inline script in the packer template. However, we have found it valuable to put these small snippets into self-contained files. Here you can include additional validation for the expected application or configuration to be present (apart from the exit-code), or you can also include these in separate files.

For example, when you need sqlcmd on your image and you install the following package:

```
choco install sqlserver-cmdlineutils -y
```

Install-SqlCmd.ps1

You then validate that the command is actually available on the VM:

```
if (!(Get-Command sqlcmd)) {
   exit 1;
}
```

Validate-SqlCmd.ps1

These are the unit tests of your packer build!

Generalize

To deploy a Windows image to different PCs, you first need to generalize the image to remove computer-specific information such as installed drivers and the computer security identifier (SID).

Sysprep (System Preparation) prepares a Windows installation (Windows client and Windows Server) for imaging, allowing you to capture a customized installation. Sysprep removes PC-specific information from a Windows installation, "generalizing" the installation so it can be installed on different PCs.

The following provisioner is often used as the last step in the Packer template to generalize the Azure VM before the image is captured:

```
{
  "type": "powershell",
  "inline": [
    "if( Test-Path $Env:SystemRoot\\windows\\system32\\
    Sysprep\\unattend.xml ){ rm $Env:SystemRoot\\
    windows\\system32\\Sysprep\\unattend.xml -Force}",
               "& $env:SystemRoot\\System32\\Sysprep\\
               Sysprep.exe /oobe /generalize /quiet /
               quit",
               "while($true) { $imageState = Get-
               ItemProperty
HKLM:\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\
Setup\\State | Select ImageState; if($imageState.
ImageState -ne 'IMAGE_STATE_GENERALIZE_RESEAL_TO_OOBE')
{ Write-Output $imageState.ImageState; Start-Sleep -s
10  } else { break } }"
  ]
}
```

Full script: https://bit.ly/xprt-packer

Amazon provides the following scripts in the EC2 instances:

```
{
      "type": "powershell",
      "inline": [
        "C:/ProgramData/Amazon/EC2-Windows/Launch/
        Scripts/InitializeInstance.ps1 -Schedule",
        "C:/ProgramData/Amazon/EC2-Windows/Launch/
        Scripts/SysprepInstance.ps1 -NoShutdown"
      ]
 }
```

Initialize and generalize an EC2 instance

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/sysprep--system-preparation--overview

Deploy

After the image has been created, Packer stores the image in the Shared Image Gallery on Azure, or in the EC2 Console in AWS. From here on the work for Packer is done. Now the image is available in the cloud provider you use, and you can create new VMs from it!

An example script for creating a VM from either a Shared Image Gallery or Managed Image directly on Azure would be:

```
az vm create \
 -n MyVm \
 -g MyResourceGroup \
 --image /subscriptions/xxx/resourceGroups/xxx/
providers/Microsoft.Compute/galleries/xxx/images/xxx

az vm create \
 -n MyVm \
 -g MyResourceGroup \
 --image
 /subscriptions/xxx/resourceGroups/xxx/providers/
 Microsoft.Compute/images/xxx
```

On AWS you use a Cloudformation yaml file to deploy the created AMI file as an EC2 Instance. In the file, set the id of the created AMI as the imageid option.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html

CI/CD

Since we now have a fully automated build process for creating machine images, it's quite easy to set up a build pipeline like we're used to doing for software development.

Azure Pipelines

The most convenient way to build from an Azure DevOps pipeline is to install and use the packer extension. First of all there is a task to download a specific or, when not specified, latest version of Packer and put it on the path.

```
- task: riezebosch.Packer.PackerTool.PackerTool@0
  displayName: Download packer
```

Then there is the second task of executing packer commands and using a service connection to provide Packer with the credentials for the selected cloud provider.

```
- task: riezebosch.Packer.Packer.Packer@1
  displayName: Packer build
  inputs:
    azureSubscription: $(serviceConnection)
    templatePath: packer.json
    force: true
    variables: |
      resource_group=$(resource_group)
```

AWS CodeBuild

You can use Linux containers to build source from a git repository on AWS CodeBuild. You can use a .yaml file to set up a build definition just like in a git repository in Azure DevOps. A build definition on AWS consists of several phases, for example 'install', 'pre-build' and 'build'. Before we can build the packer json file, we need to install Packer on the Linux container.

pre_build:

```
pre_build:
    commands:
      - curl -sS -o packer.zip
https://releases.hashicorp.com/packer/1.5.1/
packer_1.5.1_linux_amd64.zip
      - unzip packer.zip
      - mkdir -p /usr/local/bin
      - mv packer /usr/local/bin
```

After installing Packer we can use the packer command line to build the Amazon Machine Image (AMI). A packer EC2 instance will start, based on the packer.json file provided in the build definition. This EC2 instance will execute all the scripts and files in the provider section of the packer file. To build the packer file, add the following lines to the yaml file.

```
  build:
    commands:
      - packer validate packer.json
      - packer build packer.json
```

That's it

If you treat the infrastructure like code, then you can set up a fully functioning deployment pipeline to create managed images and deploy virtual machines from this. By doing so, you can put all sorts of programming practices in place, for instance pull request validation and automated smoke testing. Ultimately, you will end up with an environment in which all servers are immutable and disposable. When you can recreate or update VM's with the push of a button, no one should really care about specific instances. Each VM will be the same on each environment, and deployments will be repeatable. </>

Manuel Riezebosch
ALM/Cloud/Docker/Git/TDD

"Automate all the things!"

https://xpirit.com/xpiriter/manuel-riezebosch/

Arjan van Bekkum

"To really make a difference you have got to stand out in the crowd."

https://xpirit.com/xpiriter/arjan-van-bekkum/

We are crossing the border

Xpirit Belgium was founded in November 2018, but the initial idea of starting a new consultancy company was already coined many months before. My partner in crime Gill Cleeren was still working as an employee when we made plans to join forces to build something new. For many years I had already been running a one-man show as a freelance Application Lifecycle Management consultant and was longing for a collaboration to realize something bigger, together with a strong team of people.

Authors Pieter Gheysens & Gill Cleeren

Our previous experiences in the consultancy business had given us a clear understanding what we didn't want to do and which core values were crucial to get started. Being both heavily involved as Microsoft Most Valuable Professionals (MVPs) in the developer community, we came into contact with many consultants from Xpirit in The Netherlands. It was obvious that Xpirit shared the same core values when doing business with customers, and more importantly, they had the same vision of how to grow a culture of innovation, knowledge sharing, and collaboration. The most important assets in every consultancy business are the people, eager to learn and ready to rise above themselves.

It still took a while to officially launch our business in Belgium and to define the rules of engagement. But once we started, we never looked back and we're very happy to have chosen the path of extending the Xpirit brand and the cooperation across the border. Being able to rely on an existing business partner has proven its value.

Over the nearly two years that we have been in business now, we have grown into a local team of seven consultants without having a specific plan for growth. The people who joined in the beginning were all from within our own network, people whom we could trust and who showed the technical skills that we instantly needed for our customers. This journey to add people to our team and to keep everyone happy has been the biggest challenge, and we have learned a lot about the different personalities in our team. Technical expertise is one important asset to make a difference, but we noticed that it's even more important to have the right mindset and to fit into the team as a team player. In the end, we want to build a long-lasting relationship with everyone who joins Xpirit Belgium.

The last couple of months have been quite different with the ongoing Covid-19 situation and we miss the in-person events to keep up-to-date with each other but we feel that we are ready to take the next step and welcome extra people to our team. And so, for the first time, Xpirit Belgium is officially hiring and we are looking for new talent!

Xpirit won the Microsoft Global DevOps Partner Award of the Year in 2018 and still employs the most Microsoft MVPs within one single company worldwide, in addition to two Microsoft Regional Directors. A few months ago, Xpirit also achieved the GitHub Verified status which enables our customers to apply DevOps practices in all areas. </>

Microsoft® Most Valuable Professional

https://xpirit.com/xpiriter/pieter-gheysens/

https://xpirit.com/xpiriter/gill-cleeren/

We Are Hiring

Do you want to become part of a team that is constantly learning, challenging the status quo, sharing experiences, driving change and having fun?

We are looking for highly motivated people to strengthen our team of Azure Cloud and DevOps consultants.

Get into contact with us and let's grab a coffee to discuss the next step in your career! Our consultants in Belgium and The Netherlands are the best you can find in the industry and they would love to have a conversation on how you can start your journey to be part of the best of the best in the Microsoft technology space. Are you ready to make the jump and move up to the next level?

xpirit.com/careers

www.xpirit.com

Think ahead.
Act now.

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