XPRT.

Magazine N°7/2018

TECHORAMA SPECIAL EDITION

Future Tech, Test Adventures, Cloud Strategy and DevOps

Hal9000, Skynet and the Samaritan

Property-based testing

Why Containers Will Take Over the World

Back to the future of the value stream

Xpirit

PROUDLY PART OF XEBIA GROUP

Think ahead. Act now.

We are crossing the border.

xpirit.com

Microsoft Partner | 2018 Partner of the Year Winner DevOps Award

Xpirit

Colofon

XPRT. Magazine N° 7/2018

Editorial Office
Xpirit Netherlands BV

This magazine was made by
Vivian Andringa, Pascal Naber, René van Osnabrugge, Martijn van der Sijde, Loek Duys, Alex Thissen, Kees Verhaar, Geert van der Cruijsen, Chris van Sluijsveld, Marco Mansi, Marcel de Vries, Pascal Greuter, Alex de Groot, Roy Cornelissen, Jesse Houwing, Marc Duiker, Sander Aernouts, Jasper Gilhuis, Peter Groenewegen, Rob Bos, Immanuel Kranendonk, Michiel van Oudheusden, Wendy van Veenendaal, Manuel Riezebosch, Marc Bruins, Reinier van Maanen, Maira Camu,

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

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

Translations
TechText

Gold
Microsoft Partner
Microsoft

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

In this issue of **XPRT.** Magazine our experts share their knowledge about Future Tech, Test Adventures, Cloud Strategy and DevOps.

How a passion for sharing knowledge and community work led to the Techorama conference

Welcome to the special Techorama edition of our XPRT magazine! How did this conference begin, and how did they find their amazing (international) speaker line-up for a first conference year in the Netherlands? It all happened because of an amazing group of people that have a passion for community work and sharing knowledge!

Author Marcel de Vries / CTO Xpirit

The roots of Techorama lie in Belgium, where Gill Cleeren, Pieter Gheysens and Kevin DeRudder started this conference five years ago after Techdays had ceased to exist. This team of community leaders thought it would be a cool idea to organize an internationally recognized conference, to replace Techdays. The conference was organized, and Microsoft helped spread the word that Techorama would replace Techdays in Belgium. The team built up a great network of speakers that loved to speak at the conference. Techorama has always been about the experience of both the attendees as well as speakers and this seems to resonate very well given the success.

HORAMA
do epic shit

In its first year in Belgium Techorama attracted 600 attendees and in subsequent years 800, 1000, 1500 and last year even 1600 people attended! Techorama is a conference that is run mainly by the community. The crew of volunteers has become an all-round team that runs the event smoothly.

As you probably know, we at Xpirit have always supported knowledge sharing in many ways and have been involved in supporting Techdays in the Netherlands since the beginning. Last year it became clear that Techdays in the Netherlands would also cease to exist. After over 15 years of existence, Techdays would leave a ‘conference legacy’ behind and there would be no really good conferences to take its place. The major changes in the Microsoft organization, including a reorganization of their team of developer evangelists, left a vacuum, and it became apparent to us that we would not have a great conference anymore.

Together with the help of the Dutch community we contacted the Techorama Belgium team to see if we could do the same thing as in Belgium. After a short conversation it was apparent that we could try to make this happen in exactly the same way, and this was the birth of Techorama in the Netherlands. To pull it off we needed to move fast, since organizing a conference of this size is not a simple task.

We started with finding a location that would create the same experience that is the signature of Techorama: a cinema location. Nothing is more impressive as a presentation that is the size of a full-size movie. For speakers it is also an amazing experience when you don’t have to scale up your fonts to make them readable during your demos. We found Pathé in Ede to be the perfect place for the first Dutch edition.
Together with communities such as SDN, Stichting DotNed, .NET Zuid, .NET Noord, .NET Oost and meetup organizers in the Microsoft community, we crafted the complete plan.
We sent out a Call for Papers and received an overwhelming 800 session submissions! Well-known international speakers submitted awesome sessions and we selected over 100 talks to be delivered over the course of two days.

So, we had a great mix of international and local established speakers and thought leaders to educate our attendees on the latest and greatest in technology.
We also sought a really awesome keynote speaker and this made us reach out to Scott Guthrie, Executive Vice President of the Cloud and Enterprise group in Microsoft. Scott is a well-respected speaker and one of the few executives that will still code on stage. We asked him if he would like to help us in our efforts to create a new conference to train the Dutch community on the latest Microsoft technology. You can imagine how excited we were when we received the news that he was willing to do so.

Early April we kicked off the sales of the conference tickets, and at first things did not look that promising. We were confronted with the fact that most people decide very late to buy a ticket. Fortunately, we received a lot of support and encouragement from many companies. We were contacted by many partners who helped us establish the conference and spread the word on what we were trying to achieve.

And now here we are. Techorama Netherlands is a fact.
We have an amazing speaker lineup, we have a big group of wonderful supporting partners and best of all we have over 1000 attendees the first year!

We are looking forward to a great first edition and we hope our effort to build this new learning experience is to your liking. We are thrilled to welcome you and hope you have an awesome experience!

I want to give a special thanks to the following amazing community leaders who made Techorama possible:
Gill Cleeren, Pieter Gheysens, Kevin DeRudder,
Dennis Vroegop, Sjoukje Zaal, Marcel Meijer, Roland Guijt,
Andre Obelink, Barry Luijbregts, Gerald Versluis,
Andre Kamman and Albert Alberts. </>

Marcel de Vries,
Chief Technical Officer &
Pascal Greuter,
Managing Director

Hal9000, Skynet and the Samaritan

If pop culture has taught us anything, it would be that we can't trust Artificial Intelligence (AI) applications, because they will turn against us eventually. I think pop culture is wrong.
We are just training the machines wrong and feeding them biased material. Whether we want to or not, we are all biased to some extent, which means that even if we try not to be and train the machines, we are still adding a little bit of bias.

Author Jessica Engström

Artificial Intelligence, cognitive computing or machine intelligence involve computers that can perform tasks previously thought to require human intelligence (e.g. problem solving and learning). AI is often used as an umbrella term for artificial intelligence, machine learning and deep learning.

Before we look ahead, let's take a step back and see where these concepts came from.

AI may seem like some something new, but in reality the concept is quite old.

Artificial intelligence in the 50s

Artificial Intelligence was created as a topic in computer science during a two-month study[1] that John McCarthy pushed for in the summer of 1956. Many of the attendees were to become world leaders in AI for decades.
The goal of this study was to develop a thinking machine.

The study also included other topics, e.g. computers, natural language processing, neural networks, and other subjects that today are still highly relevant to the field of AI.

They never got as far as a thinking computer, but their brainstorming laid the foundation for AI.

Machine learning in the 50s

Machine learning is a field in computer science that gives the computer the ability to learn without being explicitly programmed.

The term was coined in the late 50s by Arthur Samuel, a pioneer in the areas of AI and computer games. It was developed from a number of other methods, including pattern recognition.

Machine learning has given us a lot of cool applications, and some of them are my favorites which I use on a daily basis, for instance more effective web searches, personal recommendations given by Netflix, and spam filters for email.

Deep learning in the 80s

In the mid-80s Rina Dechter introduced the term Deep Learning, It's the part of machine learning where a model learns to classify things directly from the data, e.g. images, text or sound.
Deep learning is usually implemented with the help of neural network architecture. Deep refers to the number of layers in the network.
The more layers, the deeper the network.

Traditional neural networks have a few layers while deep networks can have hundreds and even thousands of layers.

Searching with AI in the 2010s

A couple of years ago Google built a neural network consisting of sixteen thousand processors with one billion connections. The model was used to analyze ten million random videos over a period of three days, while no help was given with things like characteristics or what it was supposed to search for.

By the end of the three-day period, it had learnt how to recognize the most common things in videos, faces, body parts and cats, in that order. It had become so good that it found cats with 74.8% accuracy and faces with 81.7%.

[1] The Dartmouth Summer Research Project on Artificial Intelligence

One billion connections and sixteen thousand processors can sound rather unfeasible for us "regular" developers. Do I need to be a data scientist or an AI specialist to use AI in my apps and solutions?
Fortunately not.

There are a lot of companies that offer services to help us out, for instance Amazon, IBM, Google, Microsoft, Apple and more.

The cognitive or intelligent services that they offer are surprisingly easy to use. Most of them are just a rest-call away. They all offer services that handle vision, speech, text analytics, video, bots & personal assistants, and even software for deep learning, tools for prediction and classifications.

If you want a taste of what you can use these services for, check out Seeing AI.[2]. It's a smart camera app that is extremely valuable for people with visual impairment. It can read text out loud, scan barcodes on products to tell you what they are, recognize friends and people around you and describe them to you, including their emotions. It can also describe the surroundings and identify currency, all this thanks to cognitive services.

There are ways to do these things locally, for instance Microsoft's Windows ML. This is great for sensitive data, small devices, and much more.

It's all about the data

As I stated before, it all comes down to how we train the machines. We are full of bias, and this influences the way we train the machine, whether we like it or not.

Microsoft released an AI bot on twitter a couple of years back. It was called Tay, and it was designed to learn from the conversations it had.

In less than 24 hours Microsoft had to withdraw the bot because it had become racist and used words that were ... let's call them inappropriate.

However, we should not fear Machine learning. Instead, we should fear the data and the way humans use, and abuse the opportunities offered by Machine Learning.

Let's take exploits and bias out of the equation for now and look at the possibilities of AI and Machine Learning.

Focus on the right thing

Replacing humans should never be the goal of AI, the goal should be to remove humans from tedious and repetitive tasks such as analyzing spreadsheets, analyzing large amounts of data, or as a failsafe to catch human errors.

We should let the machines do what they are best at, and let us humans flourish in the areas in which we excel.

By adding AI to the mix, we can save time, help more customers, and create a better workplace for everyone.

Machines for our health

A machine[3] was trained to look for breast cancer in biological tissue samples on a slide using machine l earning, predictive analytics and pattern recognition. Going through thousands of hi-res images is of course very time consuming for a person.
On average, human pathologists have a 73% accuracy rate while the machine attained 89%. Of course the result is far from perfect, but we are well underway.

The same method has been tried with other types of cancer, and in some cases results were found two years before medical staff could find the-se results using traditional methods. The machine can, therefore, make the difference between life and death, and imagine how many lives this could save!

Why not utilize our connected lives to improve our health with the help of AI? We can already keep track of what we eat with the help of our phones today. What if we take this a step further and let AI keep track of everything we eat, how much we work out etc. Who knows the type of devices we will have in the future. Perhaps it's a pair of glasses with a camera and built-in AI, or perhaps our whole society will be connected to cameras for us to utilize.

What AI could do is analyze what nutrients you lack based on the information collected on your phone, and keep track of your pulse and blood pressure, weight and so on. It can even call the doctor for you, get a prescription for iron or blood pressure medication or whatever you are in need of before it becomes a serious problem.

As we speak, remarkable advances in research in the field of prosthetics are being made. Machine learning already enables us to control a robotic limb with our minds, but researchers are working on limbs that can feel with the help of wires that link the limb and brain, telling the sensory cortex when pressure is applied.

Some researchers are working on a hand that has a tiny camera built in. Using AI it can analyze the object in front of it and determine the action to be taken, for instance, grab a can and raise it to the mouth.

A recently published research paper described prosthetic limbs packed with electronics and sensors, which, combined with a new control algorithm, can help amputees to feel more realistic sensory feedback. This is the start of letting prosthetics feel like a part of the body, not just a tool.

AI for added safety

What about the notion of using self-flying planes in addition to self-driving cars? The technology of autopilot and landing assistant is nearly ready. And why not take it a step further, because machines can often communicate better than humans can, and they are definitely better at analyzing data, in larger quantities much faster, and they can adjust accordingly.

On the streets we can have retractable road barriers that only allow authorized vehicles to pass. And we could use facial recognition in office buildings, which would avoid having to mess about with entry cards, but still ensure that only

[2] www.seeingai.com
[3] According to mobile messaging report

authorized people enter the building. Of course, we can go full "Person of interest" or "Minority Report" and search for suspicious behavior or even emotional states, but this would entail that we do not train the machines with our human bias. The machine could use facial recognition to find known perpetrators or even suspect behavior common amongst thieves or shop-lifters. Or look for depressed or suicidal behavior and deploy help before it's too late.

Mass shootings like the one in Las Vegas in 2017 might actually have been prevented if we had had systems like that. The perpetrator stockpiled an arsenal of weapons and ammunition in at least 21 bags and a couple of smaller bags during six days running up to the shooting. If the rifles had been visible it would have been easy for a human to see, but detecting behavioral anomalies is not as easy, especially at a large and busy resort. Anyway, it's not regular behavior for a single man to bring that many bags.

Smart home then, now and in the future

I remember when remote controls for private homes came on the market, allowing you to control your lights with a remote control or even over the web.

This was in early 2000 and it didn't take long until we had integrated voice control, so we could control our lights by means of voice commands. Well, it didn't work as well as voice control does today, but this was 18 years ago!

So, we went from a button to a remote, remote to voice and voice to...? The next logical step to making your home smart, is to actually make it smart.

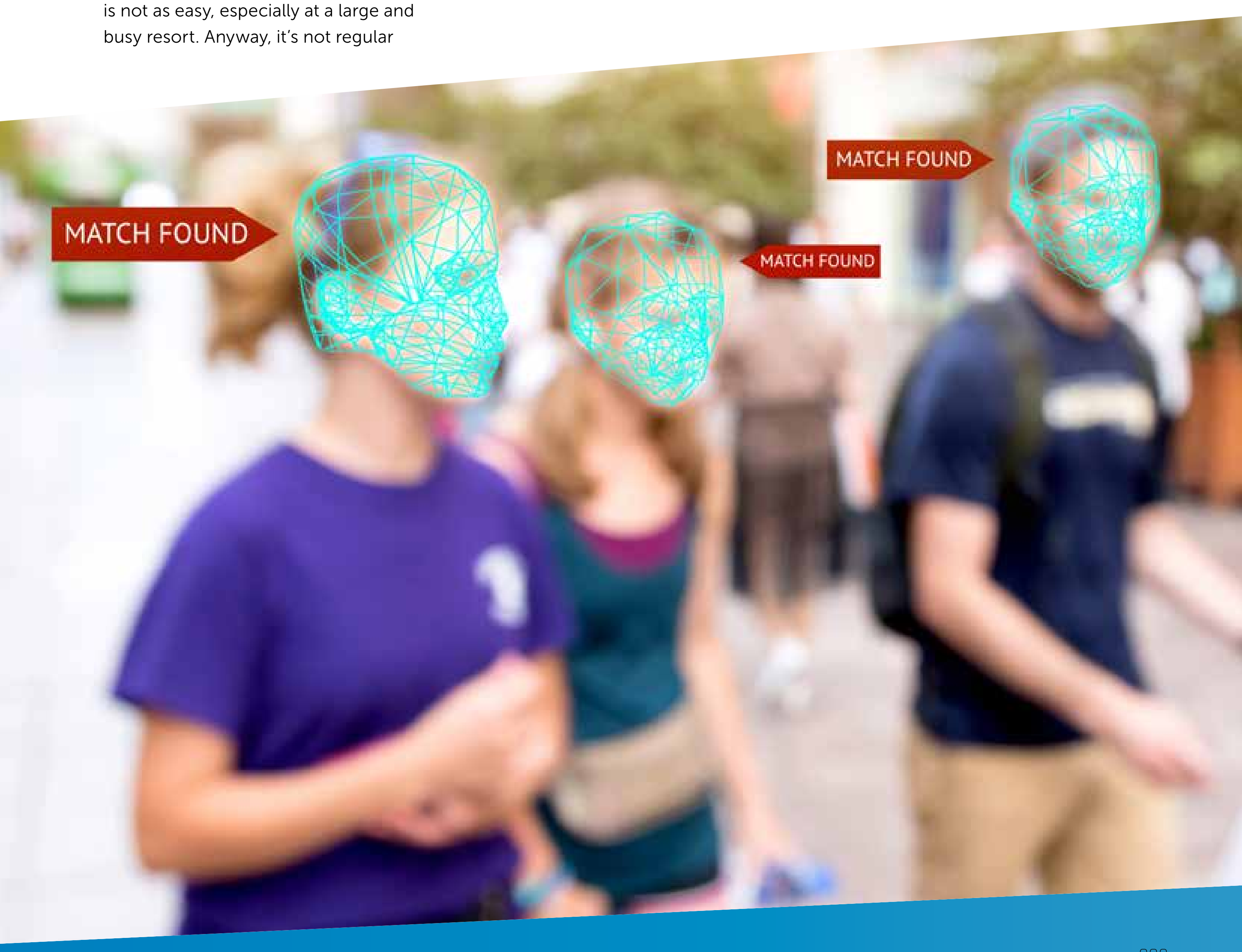
Most smartphones have digital assistants and it's getting more and more common for us to invite them into our homes with smart speakers and the likes.

Our house and assistants know basically everything about us, but they don't act on it yet.

I can manually set my washing machine to be ready when I get back from work, but if I come home early, it's not done, or even worse, if I get home very late, I will have to re-wash the laundry.

Since our house and assistants already know our schedule and they keep track of our whereabouts, why should I have to tell them when it should be done? They already know!

If we just add a little machine learning and combine all those things into one, it could look a bit different.



When the last person has left the house, the security alarm will automatically turn on, and the lights and appliances will be turned off. Our smart fridge will look up a recipe based on the ingredients we have in the house (or order groceries) so we don't have to nag about what's for dinner. It might even cook for us if we use a sous vide or crockpot.

The house would be vacuumed and mopped while we're out and when we return, it turns on the lights where needed, and music starts to play. Depending on who's home it plays different music, and if we're all home it will only play music genres we have in common.

We should not have to open an app or ask our assistants to turn the lights off at night or pull the blinds.

It's all about removing petty little tasks and letting humans focus on what's important. Giving us more time to spend with our families, giving us more time to be creative, giving us more time to relax and unwind. Something that is much needed in our society today, where we all live under constant stress and where even school kids get burn outs.

All these small things may help reduce stress and anxiety.

Many people suffer from telephobia or telephone apprehension or just don't want to talk on the phone. Google has revealed that their assistant[4] can take away all this and make it easier.
The assistant can make phone calls for you and make appointments and such. It does so in a very humanlike way with added filler words, a very natural way to speak.

Being able to use a very natural way of talking to a bot or assistant is very important, so we should try our hardest not to force language or syntax on people. It will not feel natural.

Jessica Engström

Being a geek shows in all parts of her life, whether it be organizing hackathons, running a user group and a podcast with her husband, game nights (retro or VR/MR) with friends, just catching the latest superhero movie or speaking internationally at conferences. Her favorite topics are UX/UI, Mixed reality and other futuristic tech. She's a Windows Development MVP. Together with her husband she runs a company called "AZM dev" which is focused on HoloLens, Windows development, UX and teaching the same.

The same goes for bots that are talking to us: if it feels computerized, it's not as natural.

Customer service run by bots

Bots are more popular than ever: 49,4% prefer a chatbot[5] over calling in. By adding a bot to your home page, you can let the bot answer the most common questions, and let it connect to a person when needed, making your helpdesk available the whole day, every day of the week. This will put the helpdesk in a much better position for answering questions than a situation in which its staff would have to handle all those regular questions. The waiting time would also be reduced, and if you are available 24/7, nobody will try to find another similar service just because they couldn't find that simple question on your page.

There are services available for feeding all your documents, FAQs and even recorded calls into a machine, extract all your questions and answers, and put them into an FAQ.

From that FAQ you can use a build-a-bot-from-your-FAQ service which means that you don't even have to write your own questions to have your bot up and running.

Conclusion

Don't be afraid of AI, don't let pop culture rule how we develop the future, and don't let our bias ruin what can be a glorious future.

A lot of people are talking about the risks of AI and everything that could go wrong, but I wanted to show you that AI can be a force for good. Machines won't use data for evil, whereas humans might.

Let us use AI and create the best future possible where we can focus on what matters most. So I encourage you to take advantage of all the services that do the hard work for you, and see how you can improve and optimize your apps and solutions to make a better future for us humans. </>

[4] https://siliconangle.com/blog/2017/03/05/google-uses-machine-learning-better-detect-breast-cancer-pathologists/
[5] https://youtu.be/ogfYd705cRs?t=2105

Scrum.org

The Home of Scrum

Professional Scrum Training and Certification

PSF Scrum.org

PSM Scrum.org

PSPO Scrum.org

PSD Scrum.org

SPS Scrum.org

PAL-E Scrum.org

PSK Scrum.org

www.scrum.org/assessments ● www.scrum.org/courses

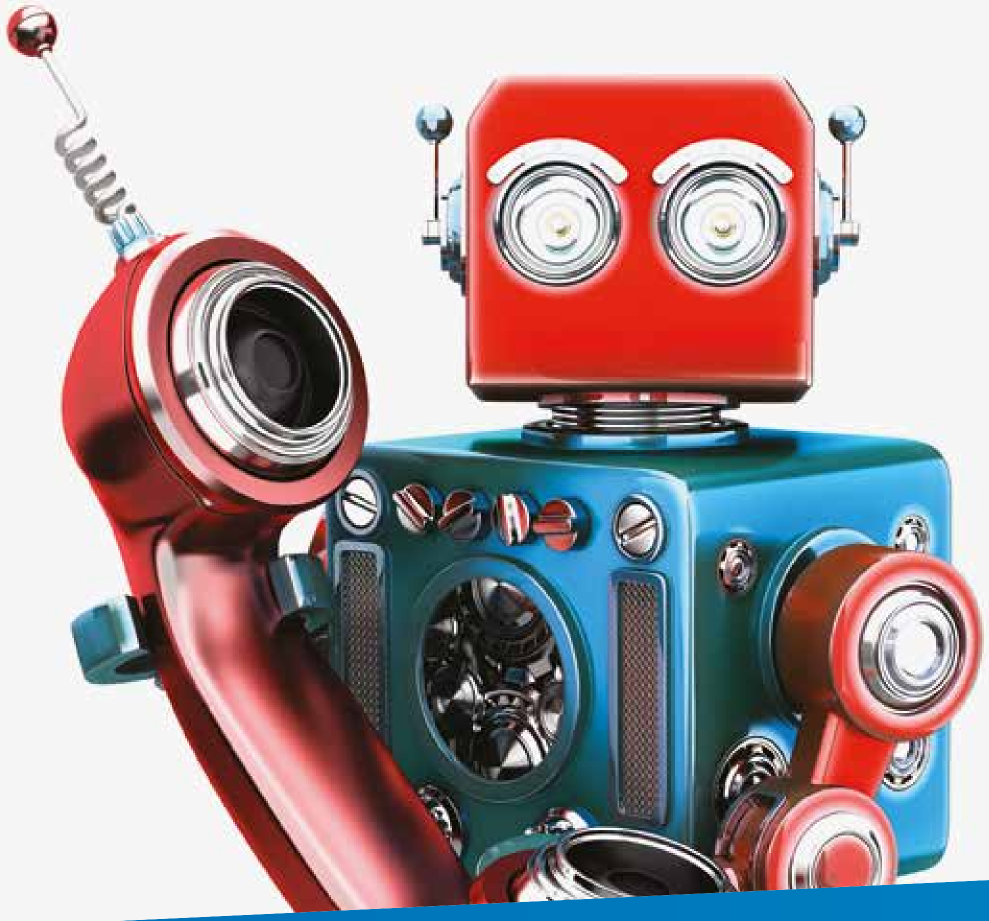
The Human Computer: The Rise of the Bots Part II

We hope you enjoyed Part 1 of this article, in which we discussed how to create a Bot that can understand and process natural language. Now, imagine using your Bot to order your family's favorite pizzas. But instead of you having to type everything, your Bot would listen to each of your family members and recognize them by their voice. It would then look up what their favorite pizza is and place the order. Sounds difficult? It isn't! You can build this using Azure Cognitive Services. In this article we'll show you how to get started.

Authors Marc Bruins & Loek Duys

Marc Bruins

Loek Duys



Azure Speaker Recognition API

Speaker verification

The Azure Speaker Recognition API is part of Azure's Cognitive Services. It is a cloud-based platform that helps you identify people by their voices. You can use it to authenticate users, instead of using usernames and passwords. Also, you can use it as a factor inside a multi-factor authentication (MFA) process. This feature is called 'speaker verification'.

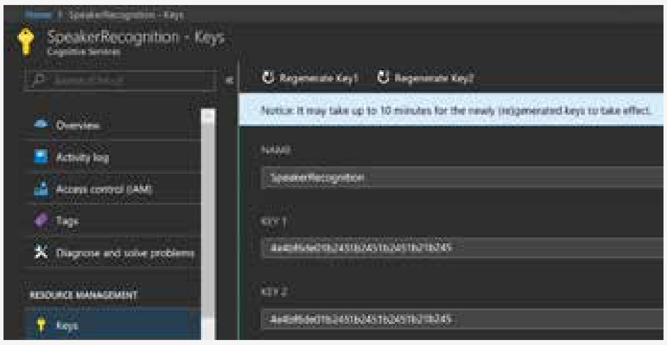
Speaker identification

The platform can also be used to identify the 'current speaker' during a conversation with multiple speakers, a feature that is useful if you need to display information about speakers during a conversation. For example, during a television interview, you could use this API to display differently colored subtitles based on who is talking. This feature is called 'speaker identification', which we'll discuss in this article.

Profiles, enrollment and recognition

The API works by creating speaker profiles and enrolling them. This means training the recognizer by uploading audio fragments containing a speaker's voice. Once enrolled, the speaker can be identified in a different audio fragment.
The identification process uses an asynchronous model; you upload an audio stream and start periodically querying for identification results. This way multiple speakers can be identified during a conversation.

Getting started



Notice: It may take up to 10 minutes for the newly (re)generated keys to take effect.

NAME: SpeakerRecognition

KEY 1: 4a4bf6de01b2451b2451b2451b21b245

KEY 2: 4a4bf6de01b2451b2451b2451b21b245

Figure 1: Speaker recognition resource

To help you get started building a Bot you'll need to prepare your development environment first. After installing the required tools, we'll show you how to set up your Bot project. Next, we'll introduce you to the various Cognitive Services and how to interact with them from the Bot.

First, you will need to get access to the Speaker Recognition API. Go to https://azure.microsoft.com/en-us/services/cognitive-services/speaker-recognition/ and sign up.
You can opt for a free seven-day trial period or use your Azure Subscription. Once the resource is deployed, navigate to the resource on the Azure Portal to grab one of your account keys, as displayed in Figure 1. You'll need this key to access the APIs later.

Bot Emulator

Also, please install the Bot Emulator v4, which can be found here: https://github.com/Microsoft/BotFramework-Emulator/releases. The Bot Emulator can be used to test your Bot during development. It will allow you to connect to a Bot and interact with it. The new 'V4 preview' version of the emulator (Figure 2) looks a little nicer and will eventually offer more functionality than the emulator we discussed in the previous article.

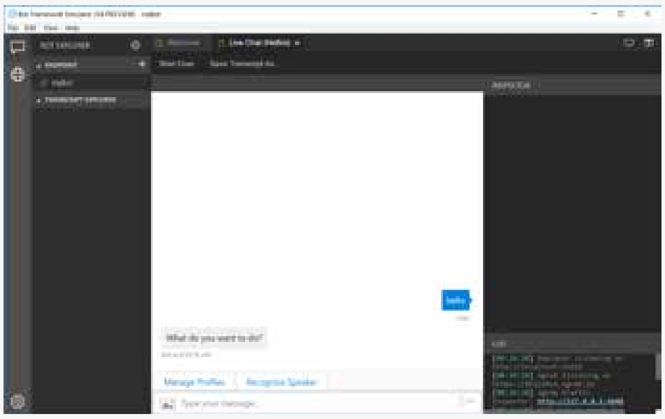


Figure 2: Bot emulator V4

Visual Studio 2017

The code used in this article was created by using the Microsoft Bot project template, which you can find here: https://marketplace.visualstudio.com/items?itemName=BotBuilder.botbuilderv4

The SDK template allows you to get started quickly by scaffolding a Visual Studio solution for you. After installing it, restart Visual Studio, and create a new project of the type 'Bot Builder Echo Bot'. You can use this project template as a quick way to get started. There is a great GitHub repository with loads of useful sample projects, explaining many Bot features. You can find it here: https://github.com/Microsoft/botbuilder-dotnet/tree/master/samples-final

At this time, both the SDK and the emulator are in preview, which means you should not deploy this version into a production environment yet.

To access the Speaker Recognition API in a simple manner, add a NuGet package named "Microsoft.ProjectOxford.SpeakerRecognition". The class 'SpeakerIdentificationServiceClient' from this package helps you interact with the Speaker Recognition speaker identification APIs in Azure. Note that there is a different class named 'SpeakerVerificationServiceClient' that can be used for speaker verification. It works in a very similar way.

Some of the code that uses the identification client was inspired by the sample code from Microsoft in this GitHub repository: https://github.com/Microsoft/Cognitive-Speaker-Recognition-Windows

Code

Now it's time to write some code. The dialogs we need to make the Bot interact with users are contained in a DialogSet.

Our Bot has the following elements:

1. Main menu dialog
This is the entry point of the conversation. The Bot will ask the user to choose between 'managing profiles' and 'recognizing speakers'.
2. Profile management dialog
This is the set of dialogs used to view, create, enrol or delete speaker profiles. These options will be displayed in the same way as the Main dialog, using buttons.
3. Speaker recognition dialog
In this dialog the user can upload an audio fragment to be analyzed for speaker identification by the speaker identification API.

Main menu

The main menu uses the code displayed in Figure 3. The menu options are passed as a list of string.

```
Add(Inputs.ManageOrRecognize, new ChoicePrompt(Culture.
English));
Add(Dialogs.MainDialogName, new WaterfallStep[]
{
   async (dc, args, next) =>
   {
      // Prompt for action.
      var mainOptions = new List<string>
      {
           MainMenu.ManageProfiles,
           MainMenu.RecognizeSpeaker
      };

        await dc.Prompt(Inputs.ManageOrRecognize,
        "What do you want to do?",
             new ChoicePromptOptions
               {
                Choices = ChoiceFactory.ToChoices
                (mainOptions),
                RetryPromptActivity = MessageFactory.
                SuggestedActions(
                     mainOptions, "Please select an option.")
                     as Activity
           });
 },
 async (dc, args, next) =>
 {
      var action = (FoundChoice) args["Value"];
      switch (action.Value)
      {
           case MainMenu.ManageProfiles:
           await dc.Replace(Dialogs.ManageProfileDialogName);
           break;

           case MainMenu.RecognizeSpeaker:
           await dc.Replace(Dialogs.RecognizeSpeakerDialogName);
           break;
      }
 }
});
```

Figure 3: Main Dialog

A dialog with a Bot usually involves multiple steps: display information, ask for input, and process input. This is modeled by adding an array of WaterfallStep[].Every delegate in this array represents a round-trip between the Bot and the user, e.g. requesting user input and processing the response.

The first step of the dialog uses the ChoicePrompt to display two buttons in the Bot Emulator. The user must choose one of the options. The call to dc.Prompt is used to display the information and buttons, and prompt for input. The second step in the dialog will be invoked after the user clicks one of the buttons. The user input is retrieved, and a new dialog is started. This is done by calling dc.Replace and passing in the new dialog's identifier.

The profile management dialog in our Bot has a very similar set-up.

Speaker Profiles

To recognize speakers, you must first create a speaker profile. After creating the profile, you can enrol the profile by uploading some audio fragments that contain the speaker's voice. The speaker recognition service will learn to recognize the speaker based on the audio fragments. In our Azure environment, we have created two speaker profiles. We have uploaded two audio fragments for each profile to enrol them, so we can test speaker recognition later. You can find these audio fragments in the 'samples' folder. *Note that in your environment, the speaker profile identifiers will be different.*

The speaker profile management dialog will use the Speaker-IdentificationServiceClient to call the speaker identification APIs. Once this dialog is in place, we can start using the enrolled profiles for speaker identification.

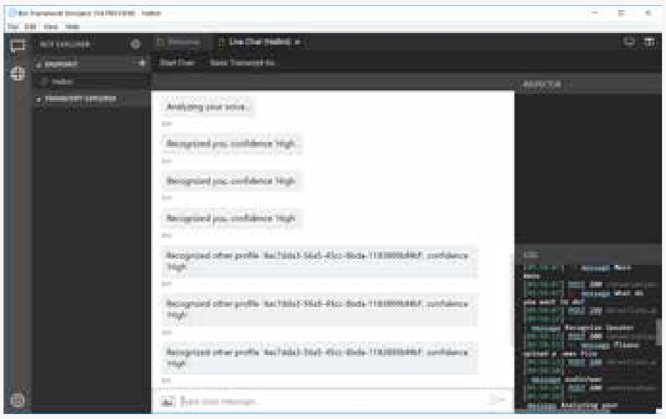
Speaker recognition

The speaker recognition dialog will request the user to upload an attachment. The simplified code is displayed in Figure 4. The user is requested to upload an audio file as an attachment. You do this by using the type AttachmentPrompt.

```
Add(Inputs.RecognizeThisPrompt, new AttachmentPrompt());
Add(Dialogs.RecognizeSpeakerDialogName, new WaterfallStep[]
{
 async (dc, args, next) =>
 {
      await dc.Prompt(Inputs.RecognizeThisPrompt,
      "Please upload a .wav file",
           new PromptOptions());
 },
 async (dc, args, next) =>
 {
      //Get attachment details
      var attachment = ((List<Attachment>) args["Attachments"])
       .FirstOrDefault();
      [..validate..]
      var state = dc.Context.GetConversationState<ProfileState>();
      string attachmentContentUrl = attachment.ContentUrl;
       [..load enrolled profiles..]
      await dc.Context.SendActivity("Analyzing your voice...");
      await AnalyzeWavFile(attachmentContentUrl, dc, state);
      await dc.Context.SendActivity("Analysis complete.");
      //we're done
      await dc.Replace(Dialogs.MainDialogName);
 }
});
```

Figure 4: Request audio

To identify speakers, we can use the Bot emulator. You can send an attachment containing a .wav file. The Bot will forward the audio file to the speaker identification API for analysis. During analysis, whenever a speaker is recognized, the Bot will notify the user. You can see an example of this in Figure 5.



Analyzing your voice...

Recognized you, confidence: High

Recognized you, confidence: High

Recognized you, confidence: High

Recognized other profile [illegible], confidence: High

Recognized other profile [illegible], confidence: High

Recognized other profile [illegible], confidence: High

An audio fragment used by the recognition API must conform to some strict rules in order to work.

It must be a .wav file, with a single audio channel using a 16KHz sample rate with 16 bits per sample.

A great way to convert almost any input audio format into a valid .wav file is by using FFMpeg, which can be run on Linux or in Bash on Windows.

For example, to convert a file 'both.m4a' use the following command line: ffmpeg -i both.m4a -acodec pcm_s16le -ac 1 -ar 16000 both.wav

Figure 5: Speaker recognition

After profile enrolment, your Bot can recognize you and your family members by their voice. Now let's bring it up one level and see how your Bot can detect what you are saying, to make an even better determination about what your favorite pizza is, based on your mood!

Making your Bot understand what's being said

Now that your bot recognizes your voice, the next step would be to let it understand what is being said. When combined with the speaker recognition, we can tell our bot to "Order my favorite pizza". Your bot will know who is speaking and what's being said! To convert speech to text (STT) we will use the Bing Speech recognition API.

Bing Speech recognition

The Bing speech recognition service allows us to convert audio to text. It is a cloud-based service that extracts text from an audio file. It works well, but there are few configuration options. If you would like more advanced features, such as custom speaker style recognition, then you should have a look at the "Custom Speech API" which is part of Cognitive Services. Note that this service is currently in preview. For our purposes, the Bing Speech recognition works fine though.

Rest API and streaming audio

The speech recognition API offers a choice between using a REST API and streaming the audio in real-time by using web sockets. Using the REST API enables us to send our .wav file to the service and get the result back in text format.

Recognition modes

When using the API, we have different recognition modes that we can use. Each of them serves a different use-case.

1. Interactive mode

This is a short request to a computer. The user expects the computer to do something as a result of what's being said.

2. Conversation mode

This mode allows you to recognize two or more people that are in conversation. It can translate all that's being said, and can also recognize slang and informal speech.

3. Dictation mode

A human citing to a computer. Typically, long utterances, like taking notes.

In our case we would like to order pizza. We know that we are talking to a bot, so our utterance will probably be something like "Hey Jarvis, can you order a pizza Tonno for me?". This is a perfect match with 'interactive mode', so we will go with that.

An educated guess

When the Speech API is not entirely sure what's being said on the audio that we are sending, it will make an educated guess and returns a list of "N-Best" values in different formats. When you think about this, it is not very different then what humans do. We also often make assumptions when we do not fully under-stand something. The most likely meaning will have the highest confidence level. You can see an example in Figure 6.

```
{
  "RecognitionStatus": "Success",
  "Offset": "1236645672289",
  "Duration": "1236645672289",
  "NBest": [
      {
        "Confidence" : "0.87",
        "Lexical" : "remind me to buy five pencils",
        "ITN" : "remind me to buy 5 pencils",
        "MaskedITN" : "remind me to buy 5 pencils",
        "Display" : "Remind me to buy 5 pencils.",
      },
      {
        "Confidence" : "0.54",
        "Lexical" : "rewind me to buy five pencils",
        "ITN" : "rewind me to buy 5 pencils",
        "MaskedITN" : "rewind me to buy 5 pencils",
        "Display" : "Rewind me to buy 5 pencils.",
      }
  ]
}
```

Figure 6: Speech recognition API result

Code

Before we start, make sure you register a new Bing Speech service on Azure, install the Microsoft Bing speech Nuget package (https://www.nuget.org/packages/Microsoft.Bing.Speech) and get the key as you did earlier in this article.
The steps that we now need to take are as follows:

1. Create a new STT dialog mode
2. Upload a .wav file
3. Send the file to the API and return the text

Now that we have explained the basic concepts and everything is in place, let's start coding! First let's add a new dialog to our bot.

Speech to text dialog

We can re-use the earlier steps and adjust it a bit since we don't need to know now who is speaking. We just want to know what the text is, and we want the selected .wav file to analyze (Figure 7).

```
Add(Inputs.RecognizeThisSTTPrompt, new AttachmentPrompt());
Add(Dialogs.SpeechToTextDialogName, new WaterfallStep[]
{
    async (dc, args, next) =>
    {
        await dc.Prompt(Inputs.RecognizeThisSTTPrompt,
        "Please upload a .wav file", new PromptOptions());
    },
    async (dc, args, next) =>
    {
        //Get attachment details
        var attachment = ((List<Attachment>) args
        ["Attachments"]).FirstOrDefault();
        string attachmentContentUrl = attachment.ContentUrl;
        //send attachment in chunks to be analyzed
        await dc.Context.SendActivity("Analyzing text...");
        await AnalyzeSpeechFile(attachmentContentUrl, dc);
        await dc.Context.SendActivity("Analysis complete.");
        //we're done
        await dc.Replace(Dialogs.MainDialogName);
    }
});
```

Figure 7: Bot dialog for STT

Sending the request

Now that we have the file we can analyze it and upload it to the Bing Speech API. We will use the 'SpeechClient' for this. The SpeechClient can optimize the result for the collector of the audio by using metadata. The code in Figure 8 shows you how to do this.

```
speechClient.SubscribeToRecognitionResult
(this.OnRecognitionResult);
var deviceMetadata = new DeviceMetadata(DeviceType.Near,
DeviceFamily.Desktop,
  NetworkType.Ethernet, OsName.Windows, "1607", "Dell", "T3600");
var applicationMetadata = new ApplicationMetadata
("PizzaBot", "1.0.0");
var requestMetadata = new RequestMetadata(Guid.NewGuid(),
deviceMetadata,
  applicationMetadata, "PizzaBotService");

await speechClient.RecognizeAsync(new SpeechInput(audio,
requestMetadata),
  this.cts.Token).ConfigureAwait(false);
```

Figure 8: Speech client metadata

Getting the response

Once the request is completed we will get a response in the method that we defined when setting up the response. Remember that we get an N-Best result list, sorted on confidence. For our bot it is enough to get the first result from the list and get the pretty "DisplayText", as you can see in 9.

```
public Task OnRecognitionResult(RecognitionResult response)
{
    var bestResult = response.Phrases?.First().DisplayText;
    //yay, a result!
    return Task.CompletedTask;
}
```

Figure 9: Speech client result

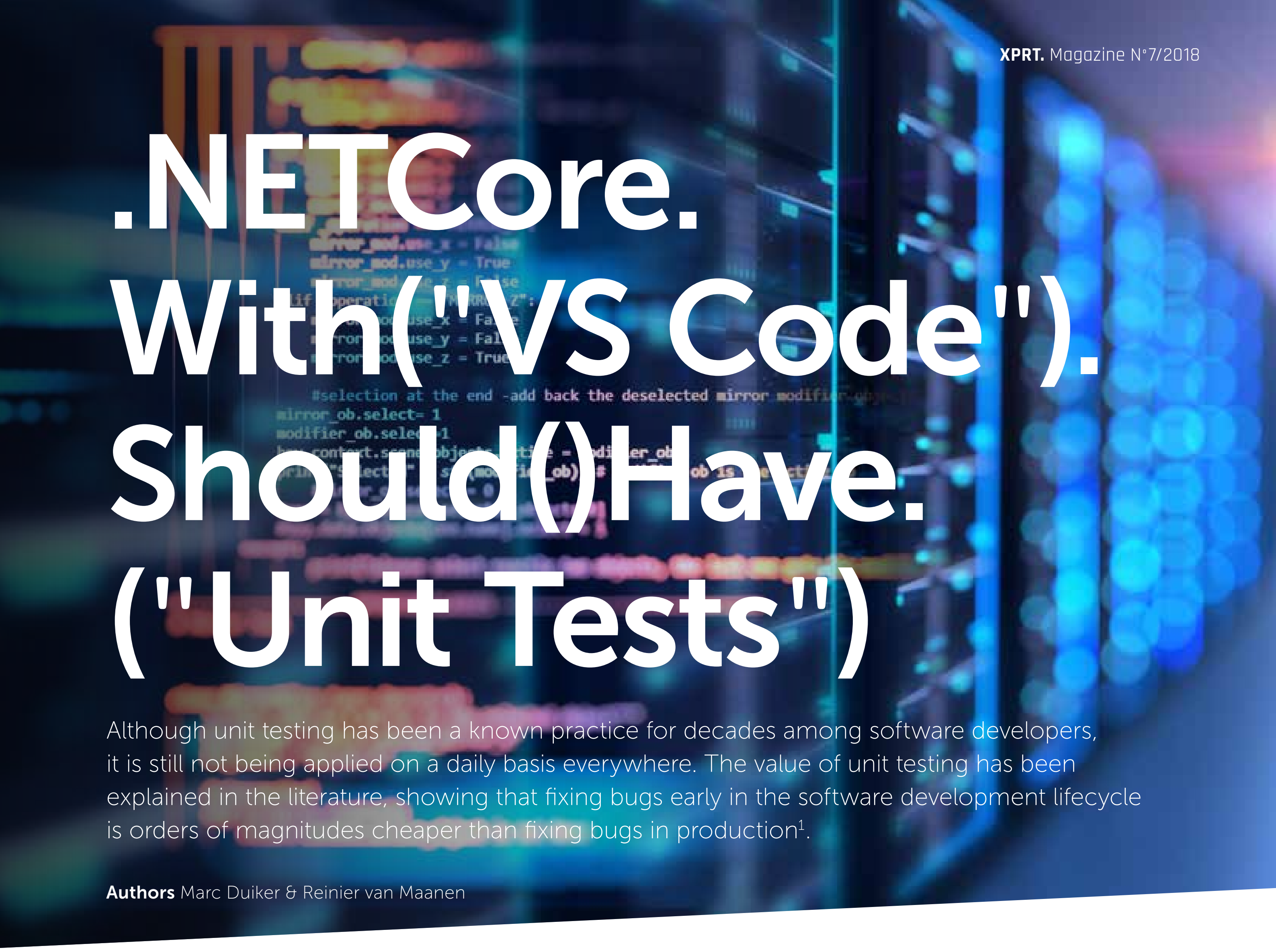
Adding sentiment analysis

Our bot is getting smarter and smarter! We now have a bot that recognizes your voice and we can convert the audio fragments to text. Obviously, we are not there yet. We don't understand the emotions of speech yet. Based on your mood, your choice of pizza may vary!

To add this, you can use yet another one of Azure's Cognitive services: the Text Analytics service. You can find out how it works here: https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics. This service can be used for a multitude of things, including sentiment analysis. The API takes text as input and returns either a score between zero and one as a result. Scores close to one mean positive sentiments (like happiness), scores close to zero indicate negative sentiments (like anger). Calling this API is easiest when using the Nuget package found here: https://www.nuget.org/packages/Microsoft.Azure.CognitiveServices.Language.TextAnalytics/2.1.0-preview. Note that at the time of writing, this package is in preview. You can use the method 'SentimentAsync' on class 'TextAnalyticsClient' to upload text for analysis.

Final thoughts

We hope this article inspired you to build your own bot and that it showed the power of the Azure services that we've used. The source code of the bot is available on GitHub: https://github.com/XpiritBV/HaBot so you can take a peek and build on what we have built. Feel free to submit a PR if you found an issue, or if you created functionality that extends the existing bot! </>



.NETCore. With("VS Code"). Should()Have. ("Unit Tests")

Although unit testing has been a known practice for decades among software developers, it is still not being applied on a daily basis everywhere. The value of unit testing has been explained in the literature, showing that fixing bugs early in the software development lifecycle is orders of magnitudes cheaper than fixing bugs in production[1].

Authors Marc Duiker & Reinier van Maanen

Benefits of unit testing

Bugs can never be prevented by just using unit tests alone, but by writing unit tests, or even better, applying Test Driven Development (TDD), you ensure that your application is designed in such a way that it is easily testable and maintainable. So when bugs are found (and they will be found) you can fix and refactor code with confidence and within time.

Another benefit of having unit tests (if they are well-written) is that they provide a clear explanation of how the application works. This helps you and future developers to understand the code base and to make changes to it much more easily than without unit tests.

So, what if you're a hip developer who prefers to hang around in VS Code instead of Visual Studio IDE and is just starting with .NET Core? How well are the unit testing frameworks and related tools supported? Let's find out...

.NET Core and VS Code

In this article we (Reinier van Maanen & Marc Duiker) will focus on how unit testing is done these days using the .NET Core framework and the VS Code[2] editor. We assume you have a basic understanding of unit testing and mocking in .NET full framework and Visual Studio IDE.

The .NET Core framework is very popular since it runs cross-platform and it performs better than the full .NET framework. VS Code has been adopted quickly by all kinds of developers thanks to the great number of extensions. VS Code is also a lightweight editor when compared to the Visual Studio IDE, so it's an ideal editor to write .NET Core applications. But it's not just C#! – support for Python, JavaScript, Java, Go, Ruby, PHP and other languages is also available. And did we mention it's free for everyone to use?

Installation

First things first, you're going to need VS Code and the .NET Core SDK[3]. Installation is straightforward and differs per platform, so we won't go into that, just follow the instructions! When you first start VS Code, it doesn't contain a lot of features, so you're going to need some extensions. Install these extensions from the marketplace[4] to improve the unit testing experience in VS Code:

> C#
> .NET Core Test Explorer
> Coverage Gutters.

Reload VS Code to activate the plugins and you're good to go.

[1] Code Complete, 2nd Edition, Steve McConnell Error Cost Escalation Through the Project Life Cycle, NASA
[2] https://code.visualstudio.com
[3] https://microsoft.com/net/download
[4] https://marketplace.visualstudio.com

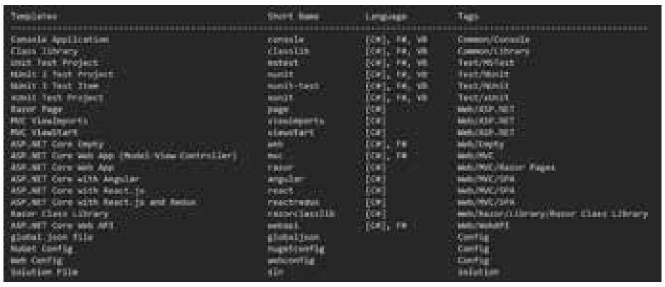
Frameworks

When you're going to write unit tests you to need to make some choices, such as which test framework and which mocking framework to use. They all have their pros and cons and it really doesn't matter much, as most popular frameworks are now supported in .NET Core.

In our example code, we'll be using xUnit[5] and NSubstitute[6], but this is merely our personal preference in this case. In our opinion, you should use the frameworks your team is the most familiar with. If you have little experience with test frameworks, stick with xUnit as it is well documented, very popular, extensible and supported by the dotnet CLI, as we'll see in a moment. For mocking, NSubstitute is a good start because it's easy to learn. If you feel like it, throw FluentAssertions[7] in the mix for more readable assertions, or AutoFixture[8] to run different scenarios against one test. We won't go into any of this in detail in this article.

Set up your projects

We use the dotnet CLI for creating projects with *dotnet new*. There are many templates available:



```
Templates                                         Short Name         Language          Tags
----------------------------------------------------------------------------------------------------------------------------
Console Application                               console            [C#], F#, VB      Common/Console
Class library                                     classlib           [C#], F#, VB      Common/Library
Unit Test Project                                 mstest             [C#], F#, VB      Test/MSTest
NUnit 3 Test Project                              nunit              [C#], F#, VB      Test/NUnit
NUnit 3 Test Item                                 nunit-test         [C#], F#, VB      Test/NUnit
xUnit Test Project                                xunit              [C#], F#, VB      Test/xUnit
Razor Page                                        page               [C#]              Web/ASP.NET
MVC ViewImports                                   viewimports        [C#]              Web/ASP.NET
MVC ViewStart                                     viewstart          [C#]              Web/ASP.NET
ASP.NET Core Empty                                web                [C#], F#          Web/Empty
ASP.NET Core Web App (Model-View-Controller)      mvc                [C#], F#          Web/MVC
ASP.NET Core Web App                              razor              [C#]              Web/MVC/Razor Pages
ASP.NET Core with Angular                         angular            [C#]              Web/MVC/SPA
ASP.NET Core with React.js                        react              [C#]              Web/MVC/SPA
ASP.NET Core with React.js and Redux              reactredux         [C#]              Web/MVC/SPA
Razor Class Library                               razorclasslib      [C#]              Web/Razor/Library/Razor Class Library
ASP.NET Core Web API                              webapi             [C#], F#          Web/WebAPI
global.json file                                  globaljson                           Config
NuGet Config                                      nugetconfig                          Config
Web Config                                        webconfig                            Config
Solution File                                     sln                                  Solution
```

Figure 1: File templates when using dotnet new

Here's how to set up a new project with a corresponding unit test project:

- .Launch VS Code if you haven't done so already.
- .Control-K + Control-O[9]: Open (or create) a new empty directory for your projects. VS Code doesn't work with solution files, so just open a directory and work from there.
- .Control-Shift-Tilde: Open a new terminal so we can use the dotnet CLI.
- .Execute the commands listed below:
  - .dotnet new console --name TicTacToe
  - .dotnet new xunit --name TicTacToe.Tests
  - .cd TicTacToe.Tests
  - .dotnet add reference ../TicTacToe/TicTacToe.csproj
  - .dotnet add package NSubstitute

You just created a new console application called *TicTacToe* and a new test project called *TicTacToe.Tests*. Then you added a reference between them and added the NSubstitute NuGet package to the test project.

[5] https://xunit.github.io/
[6] https://nsubstitute.github.io/
[7] https://fluentassertions.com/
[8] https://github.com/AutoFixture/AutoFixture
[9] If your shortcuts aren't behaving as expected, they might have been overwritten by global Windows 10 shortcuts. This seems to happen often with "control-tilde". As this can be done by any app, we can't provide any advice on how to fix this, other than removing the application that triggers when you press the shortcut.
[10] https://github.com/XpiritBV/unit testing-dotnetcore-vscode
[11] https://github.com/XpiritBV/unit testing-dotnetcore-vscode

As we assume basic knowledge of how to write unit tests, we won't go into that right now. A reference project containing the *TicTacToe* project including its unit tests can be found on GitHub[10]. The next section will describe that codebase.

.NET Core Projects Overview

We've created a repository[11] for you to clone that contains two .NET Core projects: *TicTacToe* and *TicTacToe.Tests*. The test project contains ten unit tests in two classes.

Test Project

The tests in the *EndGameStrategyTests* class verify the functionality of the *EndGameStrategy* class, i.e. when is a game of TicTacToe complete and who has won. The tests are quite straightforward, and nothing is .NET Core specific here. The tests are based on *xUnit* and use *FluentAssertions* in order to use the fluent `result.Should().Be()` syntax. The unit tests use various instances of the TicTacToe board which are created in the *BoardFactory* class.

The tests in the *AvailablePositionsTests* class verify the available number of positions on the TicTacToe board when moves are made. The system under test is the *GameEngine* class. It requires an instance of IEndGameStrategy which is injected into the constructor. Since we want to test the *Game-Engine* class in isolation, we create a fake implementation of IEndGameStrategy by using *NSubstitute*, see the *Create-GameEngine* method.

.vscode Folder

Note that the repository contains a folder called .vscode, as you might expect this contains some specific files that define how VS Code handles the projects. There are three files present in the .vscode folder:

- *launch.json:* Specifies what happens when you launch a debug session for the TicTacToe project.
- *settings.json:* Contains the VS Code workspace settings. This is the place for settings you want to share with everyone in your development team since they are in version control. In this case it contains a setting specific for the .NET Core Test Explorer extension (more on that later in this article).
- *tasks.json:* Contains a list of tasks which can be executed using the VS Code Command Palette or shortcuts. For .NET Core it always contains a build task (which can be selected with CTRL+SHIFT+B) and in this case it also contains a test task, so we can run the unit tests (more on that in the next section). The tasks that can be specified here are not limited to building or testing projects. Any tool that has a command line interface can be configured here.
- *extensions.json:* Contains a list of recommended extensions. You receive a nice suggestion to install these when opening the directory in VS Code.

Running Unit Tests

Unit tests in VS Code can be executed from the terminal, the Command palette, or from the .NET Core Test Explorer extension.

Let's have a look at each of these.

VS Code Terminal

The most basic way to execute unit tests is to use the dotnet CLI directly through the terminal as follows:

> .Open the VS Code Terminal (CTRL+`).
> .Navigate to the TicTacToe.Tests folder.
> .Type dotnet test.

The following response appears:

```
Build started, please wait...Build completed.
Test run for <LOCAL_REPO_PATH> \TicTacToe.Tests\bin\Debug\
netcoreapp2.1\TicTacToe.Tests.dll(.NETCoreApp,Version=v2
..csproj'.1)
Microsoft (R) Test Execution Command Line Tool Version 15.8.0
Copyright (c) Microsoft Corporation.  All rights reserved.
Starting test execution, please wait...

Total tests: 9. Passed: 9. Failed: 0. Skipped: 0.
Test Run Successful.
Test execution time: 1.7864 Seconds
```

Note that you only get the summary information for successful tests and nothing about individual tests that have been executed. To get more information, you can run the following:

```
dotnet test -v n
```

where `-v` is the verbosity switch and n is the normal verbosity level.

After execution of the verbose command you will see a list of the unit tests and the indication whether they have passed or failed.

VS Code Command Palette

Instead of typing dotnet test all the time and making sure you're in the right folder, you can also utilize the Run Test Task in the Command Palette. This task will run the dotnet test command for you, so you don't need to type it repeatedly. Before you can run the task, you will need to add it to the tasks.json file located in the .vscode folder.

Editing tasks.json

The following JSON snippet shows a task with the label test. This instructs VS Code to run the dotnet CLI command and passes test and the csproj file location as arguments.

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "build",
            "group": {
             "isDefault": true,
             "kind": "build"
            },
            "command": "dotnet",
            "type": "shell",
            "args": [
                "build",
                "${workspaceFolder}/TicTacToe.Tests/TicTacToe.
                Tests.csproj"
            ],
            "problemMatcher': '$msCompile"
        },
        {
            "label": "test",
            "command": "dotnet",
            "type": "shell",
            "group": {
                "isDefault": true,
                "kind": "test"
            },
            "args": [
                "test",
                "${workspaceFolder}/TicTacToe.Tests/TicTacToe.
                Tests.csproj"
            ],
            "presentation": {
                "reveal": 'always",
            },
            "problemMatcher": "$msCompile"
        }
    ]
}
```

Make sure you save the *tasks.json* file after editing.

Running the test task

Once you have defined the test task you can run it from the Command Palette as follows:

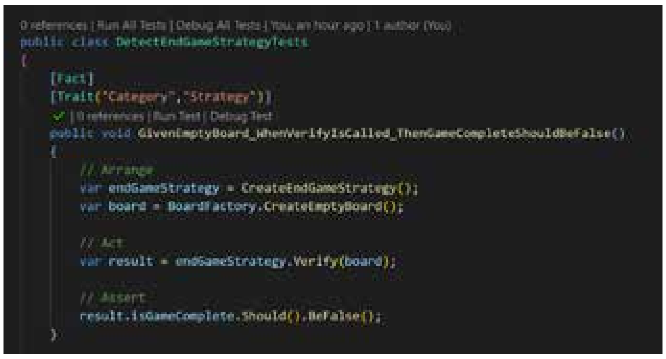
> Open the Command Palette (*CTRL+SHIFT+P*).
> Type test and select Tasks: *Run Test Task in the list of matches*.
> Hit Enter to run the task.

The output is the same as when you run *dotnet* test manually.

Creating a Shortcut for the Run Test Task

If even typing the task in the Command Palette is too much work for you, you can add a custom keyboard shortcut to select the Run Test Task:

- Open the Keyboard Shortcuts preferences (CTRL+K, CTRL+S).
> Type run test task in the Search keybindings field. Run Test Task appears, and it should not have any key bindings set yet.
> Edit the key binding (CTRL+K, CTRL+K or double-click) and enter a key binding which will activate the Run Test Task. You will be notified when you specify a key binding that is already in use. We've chosen CTRL+ALT+T.
> Save the key bindings file.
> Now type the new key binding (CTRL+ALT+T) and you'll see the Run Test Task appear in the Command Palette. Note that it doesn't execute the task, so you need to hit Enter to run it.


0 references | Run All Tests | Debug All Tests | You, an hour ago | 1 author (You)
✓ | 0 references | Run Test | Debug Test

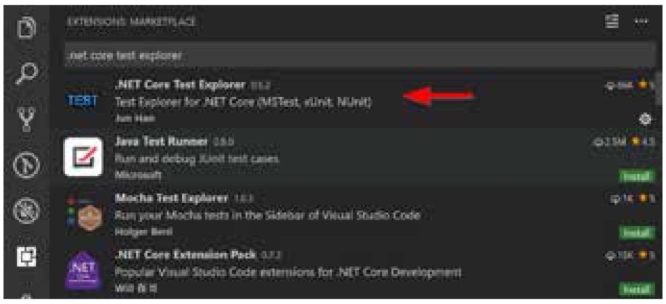
```
public class DetectEndGameStrategyTests
{
    [Fact]
    [Trait("Category","Strategy")]
    public void GivenEmptyBoard_WhenVerifyIsCalled_ThenGameCompleteShouldBeFalse()
    {
        // Arrange
        var endGameStrategy = CreateEndGameStrategy();
        var board = BoardFactory.CreateEmptyBoard();

        // Act
        var result = endGameStrategy.Verify(board);

        // Assert
        result.IsGameComplete.Should().BeFalse();
    }
```

Figure 2: Functionality, provided by the C# extension, to run and debug unit tests

All tests can be run by selecting the *Run All Tests* and *Debug All Tests* links above the class declaration. Individual tests can be run by selecting the *Run Test* and *Debug Test* links above the unit test method signatures (see Figure 2).

Test Explorer

The final method to run unit tests that we'll cover in this article uses a VS Code extension named .NET Core Test Explorer.



.NET Core Test Explorer
Test Explorer for .NET Core (MSTest, xUnit, NUnit)
Jun Han

Java Test Runner
Run and debug JUnit test cases
Microsoft

Mocha Test Explorer
Run your Mocha tests in the Sidebar of Visual Studio Code
Holger Benl

.NET Core Extension Pack
Popular Visual Studio Code extensions for .NET Core Development

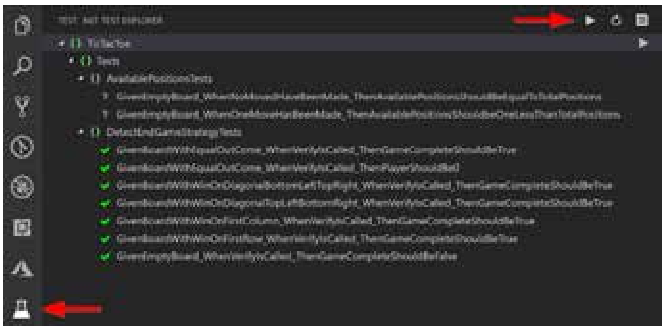
Figure 3: .NET Core Test Explorer Extension

As you can see in Figure 3 we have version 0.5.2 installed. There have been three releases in this month alone (August 2018) so there is a lot of active development going on. That is one of the reasons this extension is growing in popularity.
As with any 0.x release there are some minor shortcomings to this extension. One of these things is that the UI only updates if you start the tests from the extension. If you run dotnet test manually, or by pressing "Run Test" above your test method signature, it will still display a green check in the UI even if the test failed. Also, the detection of (new) unit tests isn't as fancy as in Visual Studio, you have to press the refresh button manually.

Unit test discovery

Once this extension is installed and you have opened the TicTacToe repository you need to build the test project for the tests to be discovered by the Test Explorer.

› Click the flask icon to open the Test Explorer (see Figure 4).
› Click the Refresh button in the top right of this window.
› If you don't see any tests yet, ensure that the dotnet-test-explorer.testProjectPath setting is set to the correct value to locathe outputte the unit test project.
› Go to Settings (CTRL+,) to verify this value. For the TicTacToe tests this workspace setting is defined as follows:
  `dotnet-test-explorer.testProjectPath": "**/*.Tests.csproj"`
› The unit tests should now be visible in the Test Explorer.



TicTacToe
Tests
AvailablePositionsTests
GivenEmptyBoard_WhenNoMovesHaveBeenMade_ThenAvailablePositionsShouldBeEqualToTotalPositions
GivenEmptyBoard_WhenOneMoveHasBeenMade_ThenAvailablePositionsShouldBeOneLessThanTotalPositions
DetectEndGameStrategyTests
GivenBoardWithEqualOutCome_WhenVerifyIsCalled_ThenGameCompleteShouldBeTrue
GivenBoardWithEqualOutCome_WhenVerifyIsCalled_ThenPlayerShouldBeE
GivenBoardWithWinOnDiagonalBottomLeftTopRight_WhenVerifyIsCalled_ThenGameCompleteShouldBeTrue
GivenBoardWithWinOnDiagonalTopLeftBottomRight_WhenVerifyIsCalled_ThenGameCompleteShouldBeTrue
GivenBoardWithWinOnFirstColumn_WhenVerifyIsCalled_ThenGameCompleteShouldBeTrue
GivenBoardWithWinOnFirstRow_WhenVerifyIsCalled_ThenGameCompleteShouldBeTrue
GivenEmptyBoard_WhenVerifyIsCalled_ThenGameCompleteShouldBeFalse

Figure 4: Test Explorer showing unit tests

Running the Tests

Click the play icon in the top right corner (see Figure 4 above) to run all unit tests, or use shortcut *ALT+R, ALT+A*.
If you need to run a selection of unit tests, based on trait values or project names, you can use the *dotnet-test-explorer.testArguments* setting. For instance, if you only want to run tests with a specific Trait, e.g. `[Trait("Category", "Strategy")]`, specify the following setting in either the user or workspace settings:

```
"dotnet-test-explorer.testArguments":
"--filter Category=Strategy"
```

Of course, this argument can also be passed to the CLI. For more information on how to use the --filter switch see the Microsoft documentation[12].

The output of the Test Explorer (visible under the .NET Test log in the output window) is similar to the plain dotnet test output, but it also contains logging to a test results file (trx), which the Test Explorer uses:

```
Executing dotnet test --logger
"trx;LogFileName=<USER_PATH>\AppData\Local\Temp\test-explorer\
0.trx" in <LOCAL_REPO_PATH>/TicTacToe.Tests
Build started, please wait...
Build completed.

Test run for <LOCAL_REPO_PATH>\TicTacToe.Tests\bin\Debug\netco-
reapp2.1\TicTacToe.Tests.dll(.NETCoreApp,Version=v2.1)
Microsoft (R) Test Execution Command Line Tool Version 15.8.0
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...
WARNING: Overwriting results file: <USER_PATH>\AppData\Local\
Temp\test-explorer\0.trx
Results File: <USER_PATH>\AppData\Local\Temp\test-explorer\0.trx

Total tests: 9. Passed: 9. Failed: 0. Skipped: 0.
Test Run Successful.
Test execution time: 1.8404 Seconds
```

Test Coverage

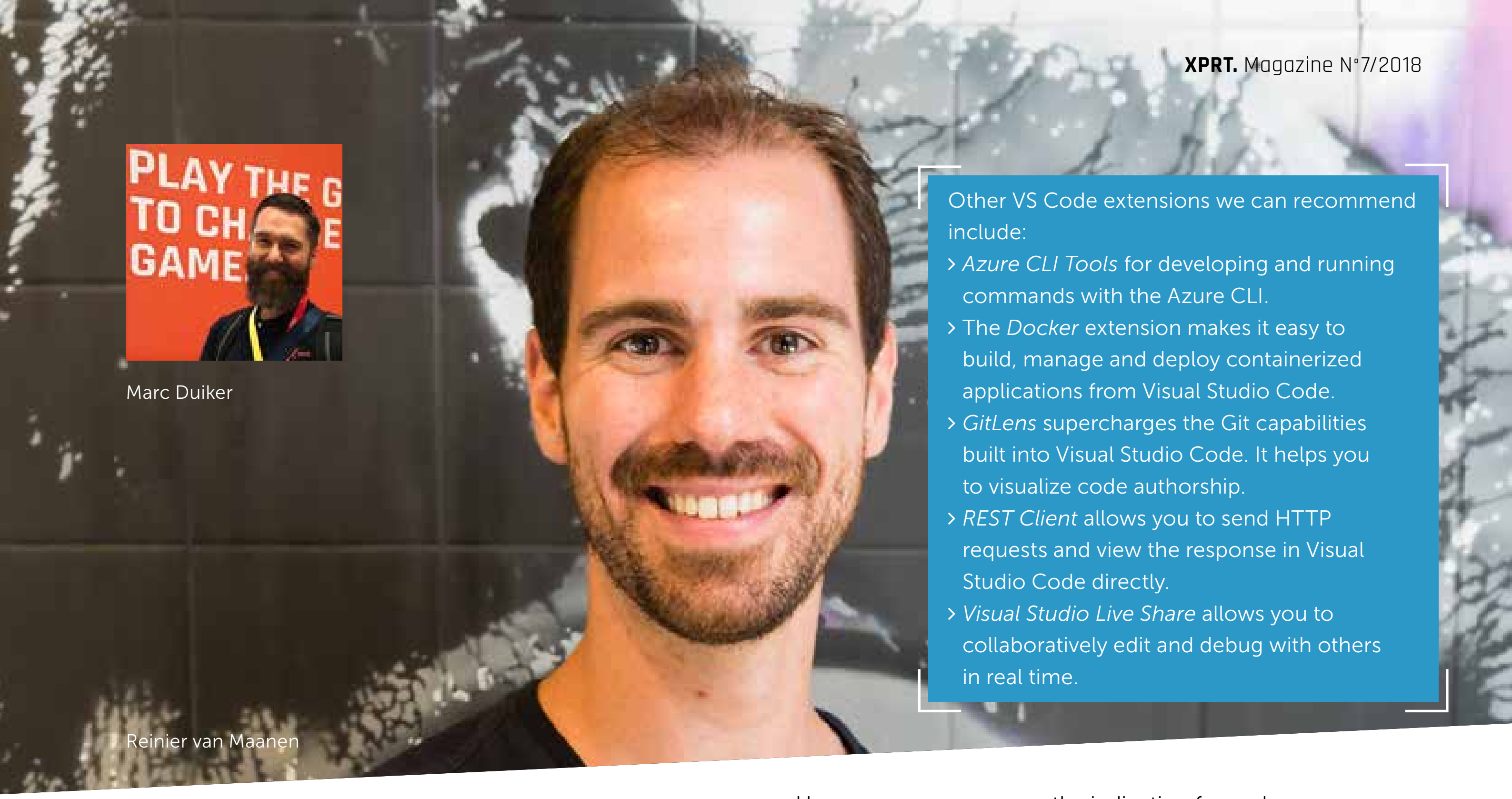
The ability to see how much of your code is being covered by tests is useful. You can use this to check if you missed any testing scenarios. It's also a common metric to review against: "new code should have at least X% code coverage".

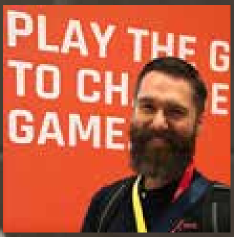
Code coverage percentage as a metric

Just checking for at least X% code coverage is not recom-mended – it says very little about the quality of your code and your tests. Also make sure your unit tests are being reviewed and that they are useful: check whether assertions are in place, make sure only one scenario at a time is being tested, and if dependencies are mocked or stubbed; also check whether tests follow the Arrange-Act-Assert pattern and that the test names are self-explanatory, etc.

Viewing unit test coverage in .NET Core has been a bit tricky in the past, but with a few libraries and add-ons this has beco-me a lot easier. We're going to use Coverlet[13] and Coverage Gutters[14] for this.

[12] https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-test?tabs=netcore21#filter-option-details
[13] https://github.com/tonerdo/coverlet
[14] https://marketplace.visualstudio.com/items?itemName=ryanluker.vscode-coverage-gutters





Marc Duiker

Other VS Code extensions we can recommend include:

- *Azure CLI Tools* for developing and running commands with the Azure CLI.
- The *Docker* extension makes it easy to build, manage and deploy containerized applications from Visual Studio Code.
- *GitLens* supercharges the Git capabilities built into Visual Studio Code. It helps you to visualize code authorship.
- *REST Client* allows you to send HTTP requests and view the response in Visual Studio Code directly.
- *Visual Studio Live Share* allows you to collaboratively edit and debug with others in real time.

Installing Coverlet can be done in two ways: as a global tool or as a NuGet package in your test projects:

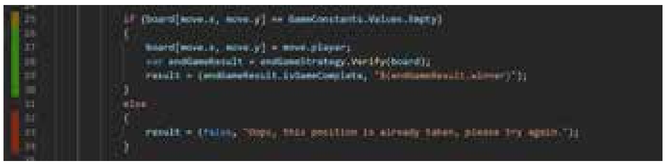
```
> dotnet tool install --global coverlet.console
> dotnet add TicTacToe.Tests package coverlet.msbuild
```

As we don't want to force every developer to install the global tool, we recommend the NuGet package. After installing Coverlet and Coverage Gutters, it's easy to get code coverage going. Just specify some extra arguments when executing "dotnet test":

```
> dotnet test /p:CollectCoverage=true /p:CoverletOutput-
  Format=lcov /p:CoverletOutput=./lcov.info"
```

This generates a "lcov.info" file in the root of your unit test directory that corresponds with the default settings for Coverage Gutters, so when you activate that extension the coverage should be visible immediately.

Of course, it's possible and recommended to add these parameters to your tasks.json file too. We recommend adding it to the *"--filter Category=Strategy"* parameter as well so you'll always have the code coverage within reach: just press the "watch" button in your VS Code taskbar[15]. This will visualize the code coverage and even update it when you rerun your tests and new coverage data is available. It should look like this by default:



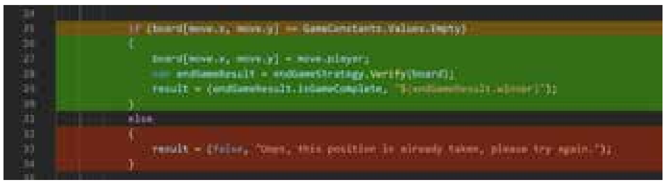
```
if (board[move.x, move.y] == GameConstants.Values.Empty)
{
    board[move.x, move.y] = move.player;
    var endGameResult = endGameStrategy.Verify(board);
    result = (endGameResult.IsGameComplete, $"{endGameResult.winner}");
}
else
{
    result = (false, "Oops, this position is already taken, please try again.");
}
```

Figure 5: Default visualization of code coverage by Code Gutters

[15] The default shortcuts for this are 'Control-Shift-8' to enable watching and 'Control-Shift-9' to disable watching. So we recommend remapping this or just clicking the button.

[16] https://www.hanselman.com/blog/AutomaticUnit testingInNETCore PlusCodeCoverageInVisualStudioCode.aspx

However, as you can see, the indication for code coverage is in the same place as where you would normally set your breakpoints. If you want to visualize the code coverage and use breakpoints at the same time, disable the gutter coverage and enable the line coverage via the settings: *"coverage-gutters.showGutterCoverage"* and *"coverage-gutters.show-LineCoverage"* respectively. This will highlight the entire code line, something we prefer anyway.



```
if (board[move.x, move.y] == GameConstants.Values.Empty)
{
    board[move.x, move.y] = move.player;
    var endGameResult = endGameStrategy.Verify(board);
    result = (endGameResult.IsGameComplete, $"{endGameResult.winner}");
}
else
{
    result = (false, "Oops, this position is already taken, please try again.");
}
```

Figure 6: Alternative visualization of code coverage by Coverage Gutters, enabling the use of breakpoints

You can read more on this topic on Scott Hanselman's blogpost[16].

Finally

As you've seen in this article, .NET Core development and unit testing in VS Code works well. Since the major unit test frameworks are .NET Core compatible, this is no excuse for not writing unit tests for your .NET Core code. Also, the tasks in VS Code are very flexible and easy to use so there's little need to use the dotnet CLI directly. The test explorer and coverage tooling are not as feature-complete yet as their Visual Studio IDE counterparts, but it is definitely workable.

In a follow-up article, we'd like to dig a bit deeper into unit testing with VS Code, especially how to handle larger solutions with multiple test projects.

Please let us know if you have specific questions regarding unit testing .NET Core projects in VS Code so we can address your issue in the next article. </>

Property-based testing

Failure is not the opposite of success; it's an essential part of it. It's through failure, in a controlled and tested environment, that we learn and improve; Trial and error is the way most of us learned how to work with a computer. We don't know any software developer who did not crash his OS because of the 'oeeee what does this button do' thought. It is because most of us could experiment safely with a computer, that we understood how a computer works, and now we fix computers for our entire family during Christmas because of that. *The surprising truth about success (and why some people never learn from their mistakes), is that it has everything to do with failure.*[1]

Authors João Rosa & Kenny Baas

Yeah! Science
In 2017 SpaceX showed for the first time in history how they managed to land a rocket back on Earth. For many, it was a fantastic event. But what few people don't see is the process getting to that point. But a few months later, SpaceX posted a video online called 'How Not to Land an Orbital Rocket Booster', making fun of the mistakes they made beginning from 2013 and onwards. The videos show the reason behind their success, failure! It is nothing new to the space industry because that industry is using a method build around failure, the scientific method.

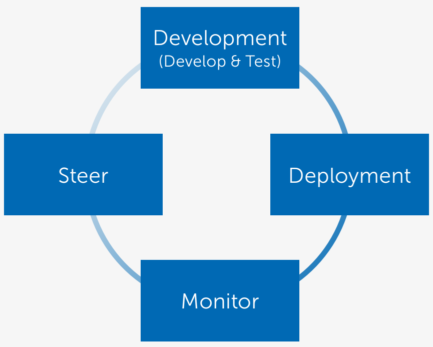
Falsifiability
When scientists use the scientific approach, they come up with a hypothesis and try to find evidence that can falsify that hypotheses. When we test our code with unit tests, we can confirm that our code works, and we can create an equivalence test for it that shows it will fail. When we do so, we test our code with fixed data called fixtures. By using fixtures we fall into the confirmation bias trap, the test confirms that our code works, but only with that input. These tests will only be as robust as the possible arguments or parameters tested against our code.

Quoting Romeu Moura: If we take a String as an argument, then the works of Shakespeare in Japanese & Korean are ONE valid input. We can achieve this robustness with parameterised testing. However, this makes the unit tests so big that it is harder to understand which behaviour it is validating. We want our unit tests to also serve as living documentation so they should be comprehensible and to the point. We can even get into more trouble as our systems evolve, and parameters can change, making refactoring messy and slow.

System evolves
Every organisation evolves, transforms itself. The organisation can start a new business, create a new product from market leads, or even expand the operations to new countries. The systems which support the organisation also have a demand to evolve side-by-side with the organisation, adapting to the new reality, allowing the users to be competent with their tasks. A classic example is the expansion of business operations to a new country, where they create new demand for the system; re-use or creation of new features can lead to a system evolution to accommodate the new requirements.

However, with the system evolution, it is also common to have some nasty side effects, such as unexpected bugs in re-used features. Using the previous example, where an organisation expanded the business operations to a new country, the countries have different national holidays; it can lead to different behaviour on features which are dependent on national holidays to support business calculations, such as in logistics operations. How can we leverage Property-based testing to create a supple design for our system?

Induce pain or stress the system
Using Lean Principles where we should seek to continue improving the system, one way is to induce pain or stress the system. There are two ways to induce pain in the system: in an uncontrolled and controlled manner.



[1] Black Box Thinking: Matthew Syed. ISBN13 9781473613775

Usually, we develop and test a system, and then deploy it to production. From the feedback (system observability, user feedback, among others), we continuously improve the system to provide additional value. How many times did we felt the pain from production? A nasty bug reported by a user that corrupts data, or even cripples the system. As a development team, we are under pressure to fix the issue observed in production, which is an example of induced pain in an uncontrolled way.
What if we shift left in the system stress, e.g., can we induce pain during the development phase before deploying the system to production? In this way, we can stress the system in a controlled manner, where we can gradually introduce stress, observing the system behaviour. If any unwanted behaviour arises, it is caught and fixed during the development phase, increasing the quality of the delivery.

Enter Property-based testing

Property-based testing is the construction of tests such that, when these tests are fuzzed, failures in the test reveal problems with the system under test[2]. In Property-based testing, we randomly generate data points within the boundary of a business invariant to verify the behaviour of the system. The English Oxford Dictionary defines property as following: "An attribute, quality, or characteristic of something".

Property-based testing not only lets us test edge cases that could expose unwanted and unexpected errors in the code but also enables us to make small tests that are readable and clear. Making these tests will also force us to think harder about the problem at hand and improve our design and code quality. Using Property-based testing pushes us to think about the state or the state transitions of the feature under test, rather than some value to satisfy some conditions. It leads the development teams to have tests focus on the behaviour of the system, rather than inputs to fulfil requirements.

The first framework implementing to use Property-based testing with was QuickCheck[3] for Haskell. A Property-based testing framework has 3 main components: (1) a fuzzer, generating pseudo-randomly values, (2) a sinker, which reduces in an algorithmic way the number of hypothesis for the input dataset, and (3) the tools for making the construction of the property-based tests with the fuzzer and the sinker.

Property-based testing in C#

In the .NET world (C#, F# and VB.NET) the framework of choice nowadays is FsCheck[4].
FsCheck ticks all the three boxes and offers integration with the 2 of the main .NET unit testing frameworks, xUnit and NUnit. This integration allows for a faster learning curve for the development teams since they do not need to learn yet another new tool, keep them focused on delivering value for the system.

[2] https://hypothesis.works/articles/what-is-property-based-testing/
[3] http://www.cse.chalmers.se/~rjmh/QuickCheck/ - QuickCheck resources, visited on 31/08/2018
[4] https://fscheck.github.io/FsCheck/ - FsCheck resources, visited on 31/08/2018

FsCheck in action

Imagine the following scenario: our team developed a system to handle the costs of a parcel shipment. The rules are straightforward; if the total cost of the parcel equals or is higher than 20 euros, then the parcel is entitled to free shipment.
The generation of the input datasets for our property-based tests can be based on 2 methods, **Primitive Generation** or **Model Generation**.

Primitive generation

The Primitive Generation is for the primitives offered by the language. With C# we have `bool`, `byte`, `sbyte`, `char`, `decimal`, `double`, `float`, `int`, `uint`, `long`, `ulong`, `short`, `ushort` and `string` (we are ignoring `object`, given it is the base for the complex data structures).

To test the parcel shipment scenario, we will start testing the parcels which have a price below 20 euro, thus are not entitled to free shipment:

```
public class WhenCalculatingParcelShipment
{
    [Property(Arbitrary = new[] {typeof(ParcelPriceBelow-
    20Euros)})]
    public void GivenParcelPriceIsBelow20Euros_
    ParcelShipmentIsNotFree(decimal parcelPrice)
    {
        var postalService = new PostalService();
        var isFreeShipment = postalService.IsFreeShipment
        (parcelPrice);
        Assert.Equal(false, isFreeShipment);
    }
}
```

Notice the Property attribute as an arbitrary for the test, where we explicitly set the context for the input dataset generation. The Arbitrary is responsible for generating the values for the feature under test, and for this case is defined as:

```
public class ParcelPriceBelow20Euros
{
    public static Arbitrary<decimal> ParcelPrice() =>
        Arb.Default.Decimal().Generator.
            Where(x => x > 0 && x < 20).ToArbitrary();
}
```

To complete the behaviour testing of the feature, we have a second test focused on the parcels that are entitled to free shipment:

```
public class WhenCalculatingParcelShipment
{
    [Property(Arbitrary = new[] {typeof(ParcelPriceEqualOr-
    Above20Euros)})]
    public void GivenParcelPriceIsEqualOrAbove20Euros_Parcel-
    ShipmentFree(decimal parcelPrice)
    {
        var postalService = new PostalService();
        var isFreeShipment = postalService.IsFreeShipment
        (parcelPrice);
        Assert.Equal(true, isFreeShipment);
    }
}

public class ParcelPriceEqualOrAbove20Euros
{
    public static Arbitrary<decimal> ParcelPrice() =>
        Arb.Default.Decimal().Generator.
            Where(x => x >= 20).ToArbitrary();
}
```

Model generation

Often our domain logic is implemented using domain models which is an abstraction of the real world. For this we need to use model generation (note that some properties and behaviour were omitted for brevity):

```
public class Parcel
{
    private readonly IEnumerable<Item> _items;
    public double TotalPrice => _items.Sum(x => x.Price);

    public Parcel(IEnumerable<Item> items)
    {
        _items = items;
    }
}

public struct Item
{
    public double Price { get; }

    public Item(double price)
    {
        Price = price;
    }
}

public class PostalService
{
    public bool IsFreeShipment(Parcel parcel)
    {
        return parcel.TotalPrice >= 20;
    }
}
```

Again, our first test will target the parcels that are not entitled to free shipment:

```
public class WhenCalculatingParcelShipment
{
    [Property(Arbitrary = new[] {typeof(ParcelPriceBelow-
    20Euros)})]
    public void GivenParcelPriceIsBelow20Euros_Parcel-
    ShipmentIsNotFree(Parcel parcel)
    {
        var postalService = new PostalService();
        var isFreeShipment = postalService.IsFreeShipment
        (parcel);
        Assert.Equal(false, isFreeShipment);
    }
}
```

With the dataset generator for the test as:

```
public class ParcelPriceBelow20Euros
{
    public static Arbitrary<Parcel> Parcel()
    {
        var input = from prices in Arb.Generate<double[]>()
                    where prices.Sum() > 0 && prices.Sum() < 20
                    select new Parcel(prices.Select(x => new
                    Item(x)).ToArray());

        return input.ToArbitrary();
    }
}
```

João Rosa

Kenny Baas

The complementary test, parcels that are entitled to free shipment:

```
public class WhenCalculatingParcelShipment
{
    [Property(Arbitrary = new[] {typeof(ParcelPriceEqualOr-
    Above20Euros)})]
    public void GivenParcelPriceIsEqualOrAbove20Euros_
    ParcelShipmentFree(Parcel parcel)
    {
        var postalService = new PostalService();
        var isFreeShipment = postalService.IsFreeShipment
        (parcel);
        Assert.Equal(true, isFreeShipment);
    }
}

public class ParcelPriceEqualOrAbove20Euros
{
    public static Arbitrary<Parcel> Parcel()
    {
        var input = from prices in Arb.Generate<double[]>()
                    where prices.Sum() >= 20
                    select new Parcel(prices.Select(x => new
                    Item(x)).ToArray());

        return input.ToArbitrary();
    }
}
```

Delayed feedback

For each time we run the tests, FsCheck will, by default, create 100 different inputs for one Property test. Because each tests get run multiple times, this means we get delayed feedback on our unit tests. The amount of time depends on the number of tests we use. FsCheck will not linearly increase the unit test time so that it won't increase the tests times a 100. Using the `dotnet test --logger:trx` we can verify the time that the tests take. On a MacBook Pro from 2017, we get:

```
testName="GivenParcelPriceIsEqualOrAbove20Euros_
ParcelShipment-Free" computerName="Joaos-MBP"
duration="00:00:00.0630000"

testName="GivenParcelPriceIsEqualOrAbove20Euros_
ParcelShipment-Free" computerName="Joaos-MBP"
duration="00:00:01.8040000"

testName="GivenParcelPriceIsEqualOrAbove20Euros_
ParcelShipment-Free_Unit" computerName="Joaos-MBP"
duration="00:00:00.0080000"
```

Not every Property-based test needs to be run 100 times either. In FsCheck we can quickly change the default 100 times to another number of our likings:

```
[Property(MaxTest = 50, Arbitrary = new[] {typeof(ParcelPrice-
Below20Euros)})]
public void GivenParcelPriceIsBelow20Euros_ParcelShipmentIsNot-
Free(Parcel parcel)
{
    var postalService = new PostalService();
    var isFreeShipment = postalService.IsFreeShipment(parcel);
    Assert.Equal(false, isFreeShipment);
}
```

Also, we don't need all tests to be FsCheck tests.
Since FsCheck integrates with xUnit and NUnit, we can combine the standard unit tests with Property-based tests. Decide on every single test if we will use FsCheck with the default value, a custom run value or just a standard unit test.

Deterministic vs Non-deterministic

A basic rule in testing is that we want our test to be deterministic, meaning that the tests will always result in the same outcome with the same input. With Property-based testing we are generating the data for our tests, so the tests are non-deterministic. Non-deterministic tests are not a problem as long as we can reproduce the errors that showed up. So if we change our test to fail:

```
public class PostalService
{
    public bool IsFreeShipment(Parcel parcel)
    {
        return parcel.TotalPrice >= 10;
    }
}
```

We will get the following error:

```
PropertyBasedTesting.Tests.Unit.WhenCalculatingParcelShipment.
GivenParcelPriceIsBelow20Euros_ParcelShipmentIsNotFree

FsCheck.Xunit.PropertyFailedException :
Falsifiable, after 5 tests (0 shrinks)
(StdGen (610985339,296499972)):
Original:
PropertyBasedTesting.Tests.Unit.Parcel

---- Assert.Equal() Failure
Expected: False
Actual:   True

----- Inner Stack Trace -----
   at PropertyBasedTesting.Tests.Unit.WhenCalculatingParcel-
Shipment.GivenParcelPriceIsBelow20Euros_ParcelShipmentIs-
NotFree(Parcel parcel) in /Users/joaorosa/Documents/code/
PropertyBasedTesting/PropertyBasedTesting.Tests.Unit/UnitTest1.
cs:line 37
--- End of stack trace from previous location where exception
was thrown ---
   at FsCheck.Runner.invokeAndThrowInner@318-1.Invoke(Object[] o)
   at <StartupCode$FSharp-Core>.$Reflect.Invoke@820-4.Invoke
   (T1 inp)
   at FsCheck.Testable.evaluate[a,b](FSharpFunc`2 body, a a)
```

Now we can use the StdGen tuple to seed an replay the failing test as:

```
[Property(Arbitrary = new[] {typeof(ParcelPriceBelow20Euros)},
Replay = "610985339,296499972")]
public void GivenParcelPriceIsBelow20Euros_ParcelShipmentIs-
NotFree(Parcel parcel)
{
    var postalService = new PostalService();
    var isFreeShipment = postalService.IsFreeShipment(parcel);
    Assert.Equal(false, isFreeShipment);
}
```

This way we can rerun our failed tests easily and fix the bug we just uncovered, a bug we usually would not find until maybe perhaps a user or even worse a hacker found!

Works on my machine!

Using Property-based testing also means that a test someone else wrote works on their machine, but fails on our machine, or on the build server. A good practice is that we as a team will fix this and own all the unit tests. Pairing or even when needed mob program the error and learn from it as a team!

In this example, we used a simplistic view of the logistics domain. In the real world, the domain logic usually is a lot more complicated, with a lot of invariants. Modelling a domain is always essential, and we would advise you to keep it simple. An approach you can use for this is Domain Driven Design. Still writing code for generators can take a lot of effort and time, definitely, but it will repay itself. When we create generators for our domain model, the code will get written faster because we don't need to be concerned with test data anymore. We only need to configure the Property-based test to put the domain models in a specific state. Property-based testing will increase the teams' domain knowledge, create more clear living documentation through the unit tests, and improve design and code quality.

You can find the code of this article at https://github.com/joaoasrosa/xpirit-magazine-property-based-testing. </>

Why Containers Will Take Over the World

Containers are the third model of compute, after bare metal and virtual machines - and containers are here to stay. Docker gives you a simple platform for running apps in containers, old and new apps on Windows and Linux, and that simplicity is a powerful enabler for all aspects of modern IT.

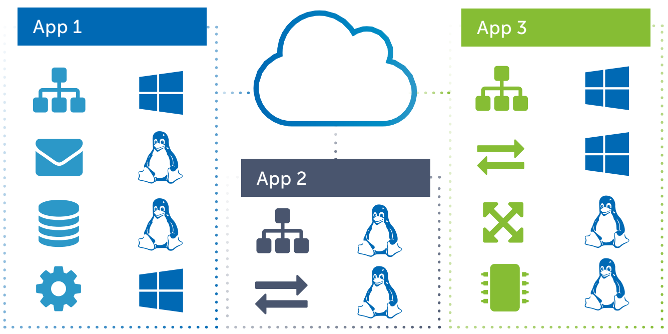
Author Elton Stoneman

In this article I'll walk through the major use-cases where people are using containers today, I'll show why Docker is a safe technology choice to invest in, and I'll finish with a learning path for you to get started with containers.

Use Case #1: Cloud Migration

Every company has a view on moving apps to the cloud. From five-year migration programs, to an immediate need to migrate because the data centre provider is shutting down in three months (this really does happen). Whatever the driver, running in the cloud should bring agility, flexibility and cost savings. To get there you used to have to choose between two approaches: Infrastructure-as-a-Service (IaaS), and Platform-as-a-Service (PaaS).

IaaS means renting virtual machines and deploying your apps in the same way you do in the datacenter. You can use your existing deployment steps, but you take all the inefficiencies of running virtual machines on-prem into the cloud. Take a simple example of three distributed apps, where each component runs in a separate VM for isolation:

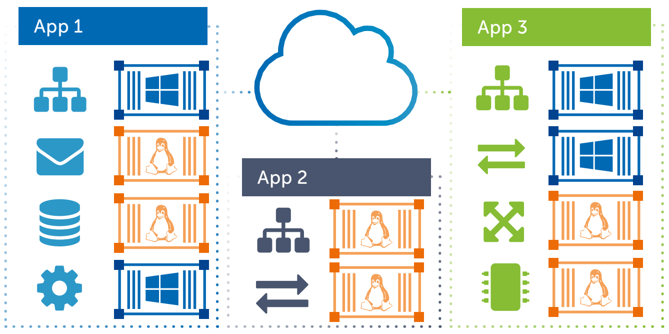


Running those apps to the cloud might use 30 VMs in production, for high availability and scale. That can cost $5K a month. You'll still have single-digit CPU utilization for your money, and you can't dynamically scale because of the time it takes to start and configure new VMs.

PaaS is at the other end of the scale. It means using the full product suite of your cloud provider and matching the products to the features your app needs. In Azure that could mean using App Services, API Management, SQL Azure and Service Bus queues.

You get complete managed solutions from the PaaS option, together with high-value features like auto-scaling. And using shared services means you should save on cost - but it's going to take a project for every app you want to migrate. For each app you'll need to design a new architecture, and if you're swapping out core components you're going to need to change code.

Docker gives you a new option which combines the best of IaaS and PaaS - move your apps to containers first, and then run your containers in the cloud. It's a much simpler option that uses your existing deployment artifacts without changing code, and it gives you high agility, low cost and the flexibility to run the same apps in a hybrid cloud or multi-cloud scenario:

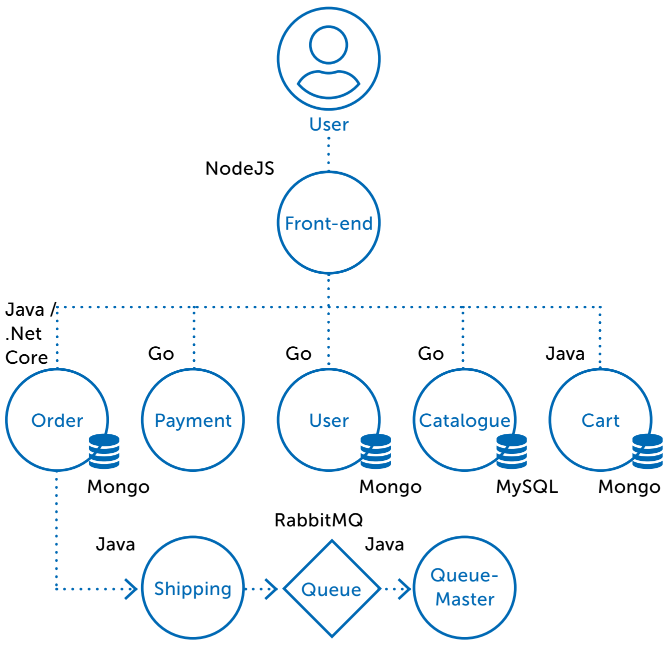


The process is simple. For each component in your app you write a Dockerfile, which is a script that deploys the component into a Docker image. Docker images are a snapshot of one version of your component - they contain the compiled binaries, dependencies and configuration - everything your app needs to run. The image is portable, you share it by pushing it to a central registry of images, which could be the public Docker Hub or your own private registry.

Then you run your app in a container and it runs in the same way everywhere.

Use Case #2: Cloud-Native Apps

Cloud-native applications are how greenfield apps should be built. The Cloud Native Computing Foundation definition is container packaged, dynamically managed, microservice apps. In a cloud-native architecture, each component runs in its own service, with its own private data store. The **microservices-demo** application on GitHub is a great sample architecture:



It's a web app with a single front-end, but the full feature set is provided by many small services – like the catalogue service, cart service and payment service. Those services each run in their own containers. Logically they form one app, but they're physically distinct components and that means they can all have their own deployment cadence. You can add catalogue features by deploying an updated catalogue service, without changing any of the rest of the app.

This is a huge enabler for the business because it removes lengthy regression-test cycles and decreases the time from idea to deployment. There's no need to test the cart service or the orders service if you're adding a catalogue feature, because those other components stay the same. It's also a great technology enabler. The sample project uses .NET Core, Go, NodeJS, Java, Mongo and MySQL - a whole range of technologies. The architecture gives you the freedom to use the right technology for each component.

You can also include production-grade open source components into your solution. Popular technologies are already packaged into public Docker images which are owned by the OSS project team, so you get the best-practice configuration of the latest version of the software, just by running a container. Check out the Cloud-Native Computing Foundation's landscape, which categorizes a huge range of technologies that fit neatly into cloud-native apps, from message queues and databases to metrics servers and dashboard visualizers.

Adopting cloud-native design accelerates your app delivery and the end result is a self-healing application which is cleanly defined in a single manifest file, and which you can deploy to any Docker cluster, knowing it will work in the same way everywhere.

Use Case #3: Modernizing Traditional Apps

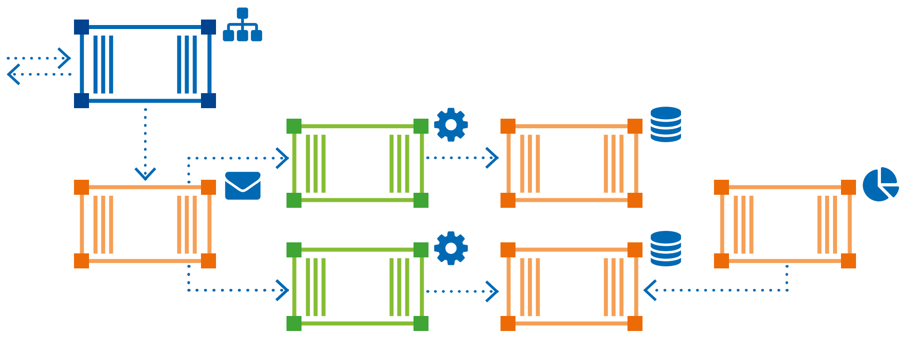
Cloud-native apps should be an important part of your future projects, but enterprises already have a much larger landscape of traditional applications. These are apps with a large monolithic codebase, running as single components.
They may have manual deployment processes, or they may be automated by joining many tools. They are complex and time-consuming to develop and test, and fragile to deploy.

Many organizations are also managing apps across a range of operating systems which are at or nearing end-of-life - including Windows Server 2003 and 2008. It's hard to maintain an application landscape which is running on diverse operating systems, which each have different toolsets and different capacities for automation.

Docker brings consistency to all containerized applications, old and new, on Windows and Linux. The Windows Server Core Docker image is maintained by Microsoft and it has support for older application platforms - including .NET 2.0 and 32-bit apps. You can take a 15-year old application and

run it in a Windows container, deploying your existing MSIs in the Dockerfile, with no code changes.

You can run your monolith in a container and get all the efficiency, portability and security benefits of Docker. Old apps which are still being used but not actively developed can stay as monoliths. Apps which are still active projects can make use of Docker to modernize the application architecture. You can split features out of the monolith, add new features and use functionality from open source components, all running in containers and all managed by Docker:



🟦 Boring ASP.NET 3.5 monolith

🟩 Cool new .NET Core app

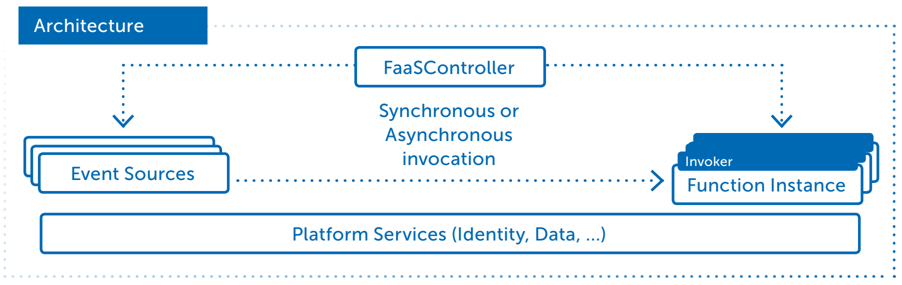
🟧 Awesome open-source project

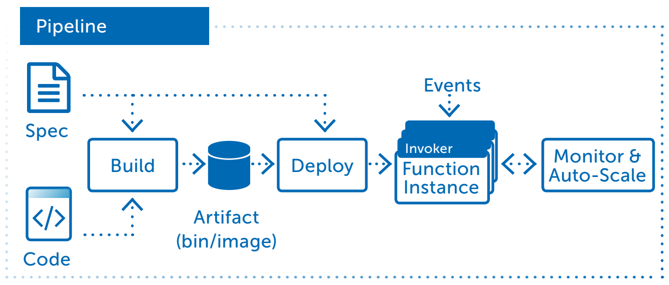
Containers let you evolve your traditional apps towards a cloud-native design, without needing a 2-year project to rearchitect them as micro-services. You can run a production Docker cluster which has a mixture of Windows and Linux nodes, for running cross-platform distributed applications. You could run Nginx in Linux containers to add performance, security and scalability to an ASP.NET WebForms app running in Windows containers.

Use Case #4: Technology Innovation

Technical innovation doesn't end with cloud-native apps. Trends like IoT, machine learning and serverless functions are all coming closer to mainstream, and they're all made easier and more manageable by Docker. I'll focus on serverless here.

Serverless is all about containers. Developers write code and the serverless framework takes care of packaging the code into a Docker image, and running it in a container when a trigger comes in - like an HTTP request or a message on a queue. The Cloud Native Computing Foundation has specced out the architecture and deployment pipelines which are common to all serverless platforms:


Architecture
FaaSController
Synchronous or Asynchronous invocation
Event Sources
Invoker
Function Instance
Platform Services (Identity, Data, ...)



Serverless started with AWS Lambda and Azure Functions, but it isn't just for the cloud. There are great open-source projects that use the same architecture and pipeline, but they run in Docker, so you get all the benefits of a consistent platform on the developer laptop, on test VMs in the datacenter, and on any cloud.

Open-source platforms like OpenWhisk, Nuclio, Fn and OpenFaas are powered by containers and have very active communities, as well as support from enterprises like IBM and Oracle. And because they're just containers, you can run a serverless platforms on the same Docker cluster that's already running your cloud-native apps and your traditional apps.

Use Case #5: Process Innovation & DevOps

The last big challenge facing enterprise IT is about cultural change and the move to DevOps, which should bring faster releases of higher quality software. DevOps is rightly positi-oned as people and process change, using frameworks like CALMS which focuses the change on *Culture, Automation, Lean, Metrics* and *Sharing*.

But it's hard to make big changes and measure their impact unless you underpin them with new technologies. Creating a folder called "DevOps" on the shared drive and putting all your deployment documents in there is not progress. Moving to Docker helps drive the change to DevOps, underpinning all the elements of CALMS:

› Culture	/	Common artifacts
› Automation	/	Build, ship & run
› Lean	/	Incremental updates
› Metrics	/	Consistent landscape
› Sharing	/	Production-grade OSS

The most significant benefit is helping the cultural change. Having teams working on the same technology and speaking the same language – Dockerfiles and Docker Compose files - is a great way to break down barriers. And people are excited by Docker. It's an interesting, powerful new technology which is easy to get started with and quickly improves practices from development to production. Teams adopting Docker are enthusiastic and that helps drive big changes like the move to DevOps.

Containers: Flexible and Open Technologies

Moving the to the cloud, delivering new apps, modernizing old apps, supporting technology innovation and process innovation - that's pretty much everything that's happening in the IT industry. Docker helps it all happen, which is why containers will take over the world.

But there is some work to do to get there. You need to write Dockerfiles to package your apps to run in Docker containers. You need to write Docker Compose files or Kubernetes manifests to define all the pieces that make up a distributed, containerized app. Deploying, managing and monitoring apps is different when they're running across hundreds of containers.

You need to make an investment to get the benefit of containers, but it's a safe investment to make. You can start with what you currently have, and you'll be moving to open technologies. You're not restricted to certain languages or frameworks - you can Dockerize pretty much anything if you can script the deployment. And you won't be locked in to any one vendor - the Docker image format and the container runtime spec are open standards, so you can run your apps on any container platform.

The Learning Path: Getting Started with Docker

If containers are going to going to take over the world, you'd better get on board. As soon as you start looking at the container space you'll see a huge array of technologies - Docker Swarm, Kubernetes, containerd, Istio, as well as all the vendor platforms - Docker Enterprise, AKS on Azure, and Amazon's EKS. Where do you start?

Here's my opinionated learning path. It starts you off easily and adds capabilities (and complexity) with each step. The idea is you stop when you get what you need. The endgame could be running a cloud-native .NET Core app on a service mesh with Istio on Kubernetes, or it could be running .NET Framework apps in Windows containers on Docker Swarm. Either of those is correct, if it works for you:

1. Run a container
   › Instal Docker Desktop and run Nginx or IS
2 Package your own app
   › Write a Dockerfile and build a Docker image
3. Run a distributed app
   › Use Docker Compose to define and manage many containers
4. Run an app with high availability and scale
   › Initialize a Docker Swarm cluster and deploy your Compose file
5. Add autoscaling, sidecar logic and network policies
   › Spin up a Kubernetes cluster for extra features over Docker Swarm
6. Control container integration with rate limiting & routing rules
   › Deploy and configure a service mesh on your cluster - this is where Istio comes in

There you are. It's simple really, and Pluralsight is your friend here, there's a whole set of Docker and Kubernetes courses, with lots more on the way. Now is the time to get started, so install Docker Desktop on Mac or Windows and go Dockerize! </>

Gathering metrics on Kubernetes

A million and one metrics to choose from, but what to monitor? The IT world has been evolving at a rapid pace, and we now have microservices that run inside docker containers on Kubernetes being hosted in public or private clouds. These shifts in technology and platform also introduce new challenges, such as how do you monitor your applications in these environments?

Authors Pascal Naber & Sander Aernouts

“Monitoring tells you whether the system works. Observability lets you ask why it’s not working.”
Baron Schwarz

Imagine diving straight into the combined logs of your application for the last hour and trying to find out why a certain part does not respond anymore, without any clear direction or suspicion about what type of problem you are facing. You’re searching for a needle in a haystack and it will be very hard to quickly and methodically find the problem. Furthermore, you might not even find a real problem in the logs of your application. You may be facing infrastructure issues that do not clearly manifest as problems in your application logs. In this article we will introduce you to metric systems that allow you to gather the right metrics in the right place in this new environment. We apply these metric systems specifically to Kubernetes but they are also valid for other platforms.

When you want to know the health of your system two terms are important: Monitoring and Observability.

Monitoring versus observability
First let’s talk about the difference between monitoring and observability and why we feel it is important to make this distinction. When we talk about monitoring we look at whether our system works. This is done using numerical measures or metrics that can be compared over time and plotted into graphs or tables. Also, alerts can be set on these metrics, so you can be notified when problems start to occur. Examples of these metrics are: number of failed requests per second, percentage of memory usage, average duration of requests, number of requests per second etc. The metrics used for monitoring are usually not very human readable on their own and they are often plotted against time or against another metrics in a graph in order to make sense.

When we talk about observability we look at investigating why the system is not working. An example of this is logging readable messages in a file or another more central logging system. These messages contain details about the behavior of your application and can be very useful when trying to find out what exactly is wrong.

There are various levels of logging that can range from fatal errors (your application is crashing) to very verbose messages that help you debug your running application to find out why your application is behaving in a specific way.

Monitoring and observability serve different purposes. Good metrics gathered in the right places will help you monitor your system and pinpoint where a problem is occurring, and it will give you some notion of what type of problem it is. But to answer what exactly is going on you will have to dive deeper and need more detailed information, and this is where observability comes in. When you have found the most likely place a problem is occurring, you can, for example, start diving into the logging and then into the suspicious application or suspect part of the application to figure this out. In the remainder of this article we will only talk about what metrics to use for monitoring, since observability is an entirely separate topic.

What metrics to use?

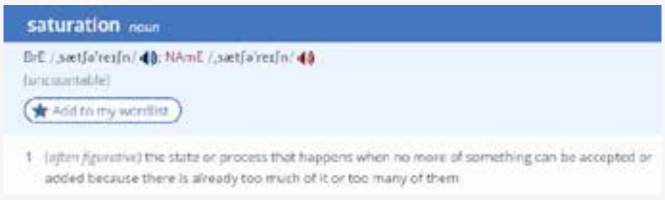
There are a million and one metrics that can be collected, but trying to monitor a lot of different metrics is confusing and does not help you to quickly find out what is wrong. Luckily, we do not have to reinvent the wheel, as there are already several sets of metrics available for both applications and infrastructure. In this article we will look at the two most well known in these areas: the four golden signals and USE.

Four Golden Signals

The most well-known way of logging metrics are the Four Golden Signals. Google has described this in the book Site Reliability Engineering, which can be read online for free: https://landing.google.com/sre/book/chapters/monitoring-distributed-systems.html#xref_monitoring_golden-signals

Google describes four kinds of metrics to monitor user-facing systems in their book. When the following four golden signals are measured, and a human is being paged in case of problematic signals, the service is decently monitored.
The four golden signals are:

- **Latency**: the time it takes to service a request.
- **Traffic**: a measure of how much demand is being placed on your system (e.g. http requests per second)
- **Errors**: the rate of requests that fail
- **Saturation**: the part of the system which is most constrained.



saturation *noun*

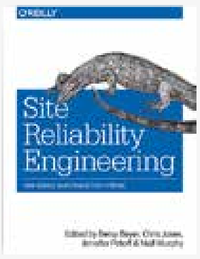
BrE /ˌsætʃəˈreɪʃn/ ; NAmE /ˌsætʃəˈreɪʃn/
[uncountable]

1 (*often figurative*) the state or process that happens when no more of something can be accepted or added because there is already too much of it or too many of them

These signals are suitable for monitoring your application or microservice, if you like. They don't monitor the CPU, Disk or Memory (it's hard to define how to monitor the traffic of a CPU for example). For this reason, we also need to monitor our infrastructure. For infrastructure a practice named USE is also available.

RED method

Tom Wilkie coined the RED method which is based on the Four Golden Signals without saturation. Wilkie believes saturation as it is used in the Four Golden Signals is for more advanced use cases and suggests to focus on the other three metrics first.
https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/



The Prometheus configuration and Grafana dashboards for the RED method are made available by Tom Wilkie on this github repository, which can be found here: https://github.com/kubernetes-monitoring/kubernetes-mixin

USE

When we look at the infrastructure, we can use the "USE metrics" to monitor the resources. USE was conceived by Brendan Gregg in his blogpost "The USE method" (http://www.brendangregg.com/usemethod.html). USE is a method to monitor resources such as CPU, Memory and Disk. USE is an abbreviation and stands for:

- **Utilization:** the average amount of time the resource was busy performing work – this tells us how busy the resource is.
- **Saturation:** the degree to which the resource has extra work it cannot perform directly. Often this work is queued. One hundred percent saturation means the resource is servicing the exact amount of work it can handle, so no queuing occurs yet.
- **Errors:** the number of errors that occur.

If you apply both the Four Golden Signals and USE metrics to your infrastructure and application stack, you have a decent visibility of the health of both your infrastructure and your application.

About errors

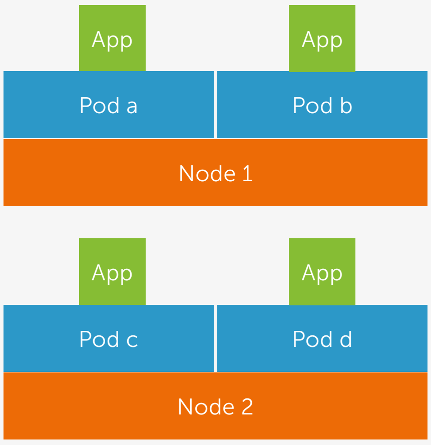
You may need to filter or separate some of the metrics out, depending on the type of errors that occur. For example, if your service returns many Bad Request (400) responses, your average duration might be very short, as Bad Request usually means the caller did not send the right information, so the response is sent almost instantly. So, you might want to exclude Bad Requests responses from your duration metric. Also, you may want to consider whether a Bad Request is really an error. In most cases it means the caller did not send the right information so it is not really an error in your application.

Depending in your situation these metrics may require some finetuning.

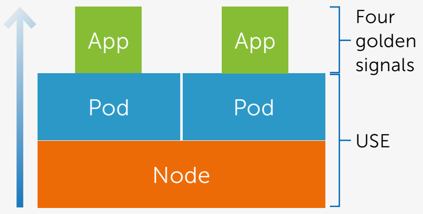
Next, we will look at how to apply these concepts to Kubernetes.

Four golden signals and USE on Kubernetes

Three different levels can be identified when monitoring an application that runs on Kubernetes. In Kubernetes your application runs inside a pod which runs on a node. So, we can look at these three levels. First, the node itself. If a node is experiencing issues, you may see issues in your pods as well. Next, we can look at the pod itself. If the pod is experiencing issues, you may also see issues in your application. Lastly, we can look at the application itself.
The underlying infrastructure (node and pod) may be healthy, but your application can still have issues that are unrelated to the infrastructure.
The following image illustrates how these three levels are related. A pod runs on a node and an application runs "on" a pod.

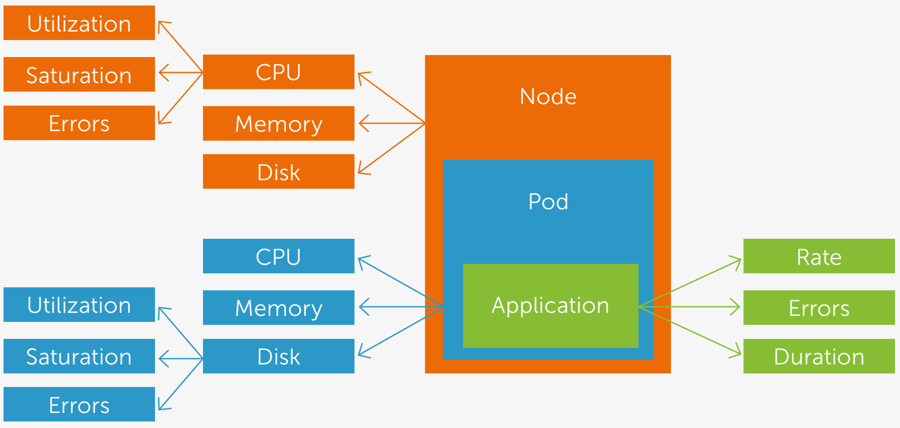


The following picture illustrates these dependencies. When a lower level is unhealthy, you are likely to see issues in the levels above as well. So when we are monitoring our systems we take a look first at the lowest level, and if that level is healthy we move up to the next level. This way gives you a structured approach to pinpoint which level is having issues.



We can treat the nodes and pods as infrastructure and thus apply the USE metrics to these levels. And for the application we can use the Four Golden Signals.
For infrastructure we need to go one level deeper, and we need to define what resources to monitor with the USE metrics. For nodes and pods we can look at the following resources: CPU, memory and disk. We apply the USE metrics to each of these resources.

The following diagram illustrates the various levels of monitoring and the resources and metrics that should be gathered when we apply the USE method and the Four Golden Signals:



How to gather these metrics on Kubernetes

The aim is to be able to gather various sets of metrics on three levels of our application. The de facto standard for gathering this type of information on Kubernetes is Prometheus, combined with Grafana for dashboards. With Prometheus we can scrape metrics from different endpoints. To gather metrics about the nodes, pods and application we will have to expose Prometheus-compatible endpoints for all of this.

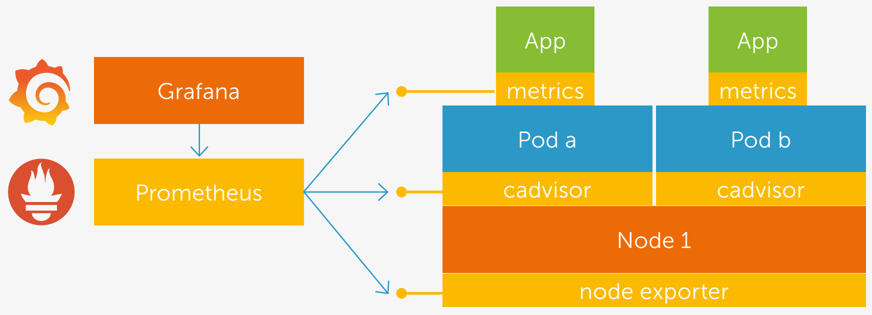
CNCF graduates
Both Kubernetes and Prometheus are part of the Cloud Native Computing Foundation (CNCF). At the moment of writing this article, both CNCF projects have reached the graduated stage. See https://www.cncf.io/projects/

To expose metrics for the nodes we can use the node exporter that is part of kube-prometheus. This is an application that you run in a pod on each node as a daemon set. It will expose metrics about your node through a Prometheus-compatible endpoint.

To expose metrics for the pods we can use CAdvisor. CAdvisor is part of your Kubernetes cluster by default and exposes metrics about your pods through a Prometheus-compatible metrics endpoint. You don't have to run any additional pods for these metrics.

To expose metrics for the application, the application itself will have to expose these metrics through a Prometheus-compatible endpoint. For .NET applications a good library to expose metrics to Prometheus is AppMetrics (https://github.com/AppMetrics/AppMetrics).

The following image shows how these parts work together to make metrics available



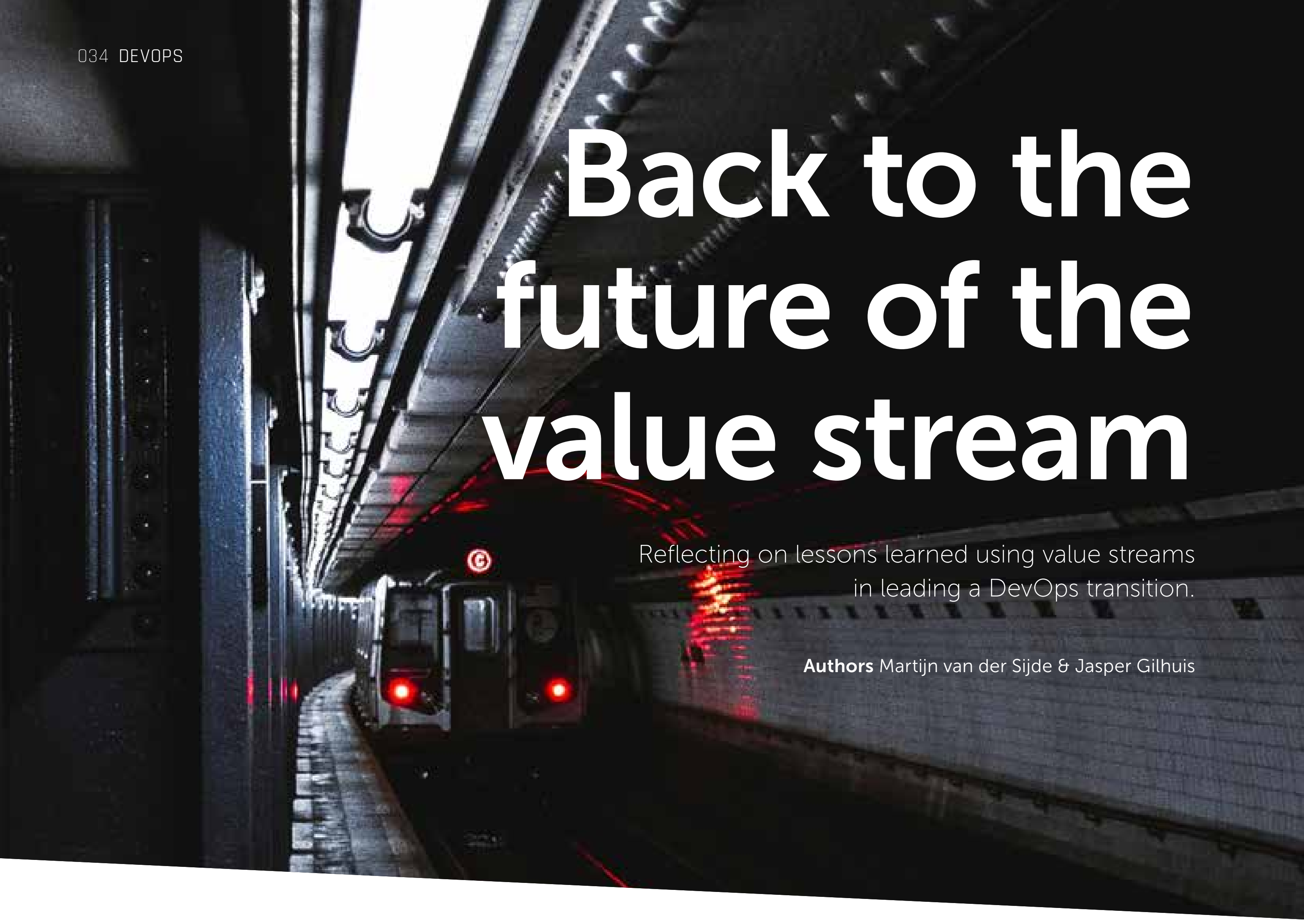
To get started quickly, you can use the Prometheus operator combined with kube-prometheus to set up Prometheus, Grafana and a set of default metrics and dashboards. Both can be found in this repository on github: https://github.com/coreos/prometheus-operator and are available as Helm charts that you can install into your cluster.

Summary

There are many metrics that can be gathered from infrastructure and applications. Luckily there are already several metrics systems available that can help you to collect the right metrics in the right place. In Kubernetes we can gather metrics on different levels: node, pod and application. You treat node and pod as infrastructure and apply the USE metrics to these. For the application, you can use the Four Golden Signals. You can use Prometheus and Grafana to gather and visualize these metrics.

When you apply these metrics to your application in Kubernetes, you have a solid foundation for monitoring. </>

Pascal Naber & Sander Aernouts



Back to the future of the value stream

Reflecting on lessons learned using value streams in leading a DevOps transition.

Authors Martijn van der Sijde & Jasper Gilhuis

A short history

In the past, technology did not have a dominant share in the products and services of companies. IT was managed in a decentralized manner, and was closely related to the business, directly supporting it. The world was not changing every second and products and services were not digital. Its role was to support the business, and this is why IT was seen as a cost center. Over time, digitalization of products and services slowly started to strengthen the role of IT, which resulted in larger IT departments. At that time, IT was often organized around software delivery value streams* with all required roles and expertise located on the same office floor. When you looked inside these software delivery value stream organizations, they were organized horizontally (frontend, backend, operations, etc.) and in that way they were siloed. Because these silos are often reflected in the software architecture as described in Conway's Law**, technology was not yet optimized for maximum flow. However, from an organizational perspective, the department was aligned in order to foster a climate of working towards a common goal. Technical dependencies caused by the aforementioned silos were manageable and also aligned for future optimization. These opportunities for optimization made Patrick Debois coin the term DevOps in 2009. The goal of DevOps was to implement changes that would enable the IT department to make steps towards optimizing the speed of high-quality software delivery by bringing development and IT operations closer together. Unfortunately, around the same time the idea of IT as a cost center was being strengthened due to a worldwide economic recession. This forced companies, banks and insurance companies to cut costs. As a result, IT was centralized in shared service centers and centers of excellence that focused on technical disciplines, or it was outsourced. Centralization of IT allowed these companies to optimize the utilization of their resources, but later on it became apparent that this is not optimal when you want to increase the speed of high-quality software delivery.

** **Conway's Law**

"Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations."
–M. Conway

Architectures of systems will be designed and shaped in the way that the organization to deliver these systems is shaped. If you have technology value stream teams organized around a frontend application or around backend services or other splits, the architecture will reflect this.

The idea of reversing this is to design the architecture you want, with low coupling and high cohesion and form your technology value streams around that, enabling you to maximize the flow of value.

Back to the future

Nowadays we are living in a world that is changing every second. Because of digitalization everyone and everything is connected with each other. Marc Andreessen's article on "Software is eating the world"[1] is being confirmed by changes in industries every day. Products and services are largely digital. The idea that *"IT is the business"* is making its way into the board room. This new mantra is giving IT a leading strategic role in the organization. If you doubt whether this is true, try the following: ask a banking executive what would happen if he would send all his bankers home. Then ask him or her to imagine how long it would take before the bank would be out of business. After that, ask how long this would take if all IT-operations personnel, the people who prevent or solve system failures, would be sent home. Or how long this would take if all software developers were not there, causing a halt to all changes in software required for selling products or for complying with regulations.

In this rapidly changing world, the business and thus IT needs to be able to respond quickly to external developments and threats in order to achieve their business goals. Competitors who are able to add new features more quickly and deliver these to customers faster will outsmart the competition. To be able to keep up, it requires companies to be able to innovate and adapt rapidly, whilst at the same time maintaining high quality and security. The good thing is that the characteristics of software and cloud-facilitated services and infrastructure, on which digital products and services are based, enable this when applied correctly. This is why IT is turning into a profit center instead of a cost center. To be able to maximize the benefits of IT as a strategic innovation driver, the focus needs to be on delivering value, using software delivery value streams, while optimizing its speed and quality. Applying DevOps principles and practices help to realize this.

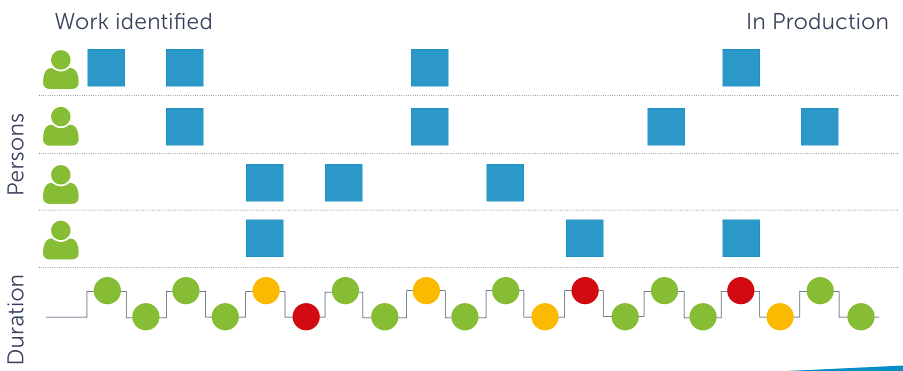
To help kick-start this optimization, we must identify value stream(s) and choose one to start with. In a way we are reverse engineering to where we came from before the centralization into centers of excellence took place. But in the case of reversing Conway's Law, the organization and technology value stream teams will have to follow the technology, instead of the technology following the way IT departments are organized. This is because our point of departure is the technology and the teams around it, while focusing on maximum flow of value. In other words: back to the future of the value stream.

* Value stream

Value streams originate from the manufacturing world. It typically covers the entire cycle from a request to delivery. Its sole purpose is to identify value. Value is only added when it is delivered to the requestor. Value streams can be used to identify flows of information and material. After identifying these flows it is important to add a timeline to these flows. Value streams can help identify bottlenecks or even waste in your system. In manufacturing this seems obvious, but one often fails to apply value streams in business and IT processes.

An example of a simple development value stream, covering personas and covering the process from work being identified towards the actual product being deployed in production.

Development Value Stream



How to carefully select a value stream to start with

With DevOps we concentrate on the software development value streams and the products and services they deliver. As stated in Gene Kim's DevOps Handbook[2] you should start by carefully selecting your first value stream. This is key because this selection determines the difficulty of the transformation, both technically and culturally. This is important from a change management perspective, because you need to increase your chances on a first success to be able to scale your transformation throughout the organization. In addition, the people leading this DevOps initiative must also be given the opportunity to experiment and learn to do this transformation. The following steps can guide you in selecting the right value stream:

1. First identify domains that deliver products and services. A way to do this is by looking for logical clusters of applications.
2. Evaluate these domains on a set of criteria. These criteria cover the areas of people, process and technology. Examples are: importance of time-to-market and ability to innovate, willingness for change, opportunity for quick wins. Creating a weighted score based on these criteria helps you choose and select a domain and value stream you want to focus on.
3. Identify the software delivery value streams (and their mutual dependencies) delivering products and services in that domain. Design a logical grouping of the capabilities these value streams provide. The technology should be functionally cohesive and not logically cohesive in order to reduce dependencies as much as possible, providing opportunities for maximizing flow.
4. Staff the team(s) for the value streams of the software delivery. You can use the DASA Competence Framework[3] for guidance on the required hard- and soft skills.

[1] Marc Andreessen, https://on.wsj.com/2oWxsdy
[2] DevOps Handbook http://bit.ly/2oYTgVT
[3] DASA Competence Framework http://bit.ly/2NDtrJj

Setting goals and measuring the effect of your improvements

In short the reasons for adopting DevOps are: being able to produce better products and services, delivered in a faster, and if possible, cheaper manner. Better products and services can be detailed further stating that the products and services need to be of high quality and can be easily adapted to changes in requirements and opportunities for innovation.

However, when do your improvements contribute to these goals? It is known that only one out of three experiments will truly be successful for your value stream. You need to know whether a change you have made was an improvement or not. That is why metrics are a hot topic when applying DevOps principles and practices, and rightfully so.

So far so good. Every well-known DevOps resource points you in that direction. Then the next question is: what metrics to start with and how to measure them. Based on DORA whitepapers[4] we recommend that you start collecting data for four metrics right away that tell you something about delivery speed and product and service quality (see Figure 1 Aspects of software delivery performance (– source: State of DevOps report 2018).

- **Deployment frequency**
  For the primary application or service you work on, how often does your organization deploy code?
- **Lead time for changes**
  For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?
- **Time to restore service**
  For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?
- **Changing failure rate**
  For the primary application or service you work on, what percentage of changes results either in degraded service or subsequently requires remediation (e.g., leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?

Figure 1 Aspects of software delivery performance (source: State of DevOps report 2018)

The count of releases to production, the changes the release contains and the number of reported incidents are metrics that are often already registered or available in a change and incident management system in a non-DevOps world. Or these metrics are part of your team's backlog in an Agile planning and CI/CD system like Azure DevOps***. The time of incident creation and the time of incident closure following a release can be registered, and this allows you to calculate the time to restore. The same is true for the number of releases to a specific environment. For the change failure rate, the relation between these two data entities, a release and an incident, needs to be made. A quick and easy way to do this is to look at the creation dates of both, and relate them to each other in that way. This leaves us with the lead time for changes. For automatically calculating this, the basic CI/CD pipeline needs to be implemented (as mentioned earlier) to be able to collect this data easily. The work item administration will be the main source for the required data.

*** Azure DevOps Metrics**

Azure DevOps, previously known as Visual Studio Team Services (VSTS), offers functionality for visualizing metrics. These extensions are available through the marketplace.

Figure 2 shows a sample Lead Time metrics chart available in Azure DevOps, using the Analytics extension from the marketplace.

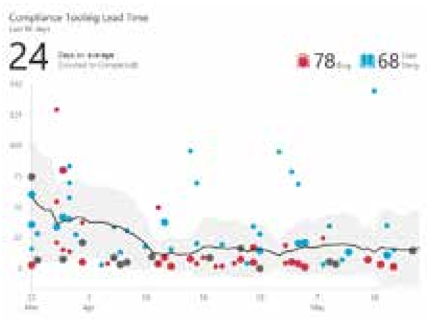


Figure 2 Sample Lead Time graph
For more information on Cycle Time and Lead Time, see: http://bit.ly/2wXDn6E using the Analytics extension http://bit.ly/2NvpciD.

Having these metrics in-place before the transformation allows you to forecast what benefits your DevOps initiative can bring. It also allows you to periodically measure whether your improvement experiments are contributing to this. From here you can add additional metrics as you see fit. You get the most value out of these metrics when they are measured over the integrated work across all teams on a product. It is important to note that these metrics do not measure individual or team performance, but are value stream-related.

What's next? Value stream mapping and continuous improvement

When we have our first technology value stream and have put the foundation for metrics in-place, it is time to adopt a continuous improvement process as a vehicle for making improvements. Or enhance this process if it is already there. The basis for this is creating a value stream map for the development value stream.

[4] Accelerate: State of DevOps 2018 http://bit.ly/2oW0sSN
Forecasting the value of DevOps Transformations http://bit.ly/2Qk9YLZ

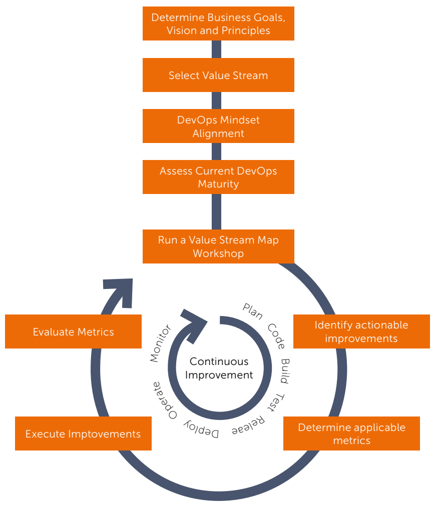

Determine Business Goals, Vision and Principles
Select Value Stream
DevOps Mindset Alignment
Assess Current DevOps Maturity
Run a Value Stream Map Workshop
Identify actionable improvements
Determine applicable metrics
Execute Imptovements
Evaluate Metrics
Continuous Improvement
Plan
Code
Build
Test
Release
Deploy
Operate
Monitor

Figure 3 Continuous improvement cycle

The value stream map will show the roles of people who take part in the software delivery value stream from requirement until monitoring a product in production. It shows which tasks they execute to make this possible.
The process to depict this will already be of help, because it allows us insight into what is required to get software into production. The next thing they will learn is where any inefficiencies occur in their work. In other words, it will expose bottlenecks that prevent a reduction in time-to-market and an increase in delivering quality. This gives them input for experimenting with improvements that reduce these bottlenecks.
The metrics data will clarify whether an improvement had the impact that was expected. Basic metrics need to be available from the start as explained, but based on a specific improvement, these metrics can be extended as required, as part of the continuous cycle.

Three take-aways

Of course, the steps explained in this article are just a few highlights of what is necessary to get a DevOps transformation up and running.
For example, we did not discuss giving people a DevOps foundation through training, or the steps required to change management behavior. The focus was on the elements that are related to the value stream. The three things you should take away after reading this article are as follows:

1. To determine where to start your DevOps transformation process, create a matrix of domains that you have identified. In this matrix you score each of the criteria that you think are important to arrive at a value stream selection. These criteria should take people, process and technology into account. It will tell you something about the relative amount of resistance to change that you can expect in the value stream teams, and how this value stream will benefit from technology optimization. It will also show the importance of time-to-market and innovation for the business of this value stream. The weighted score of this matrix will lead you to your first value stream and will enable you to start your DevOps transformation initiative.

2. Before you start making improvements with your value stream, make sure you can report the following four metrics:
   1. Deployment frequency
   2. Lead time for changes
   3. Time to restore service
   4. Change failure rate

   This will allow you to continually track progress towards the goals you have set. It is important to be able to measure your own success, but it is also important to be able to show your success to higher-level management. So, start by making an analysis of the raw data currently available in your release and ITIL management systems. If this data is insufficient, then make improvements before you start optimizing your value stream with DevOps principles and practices.

3. The value stream map is the heartbeat of your continuous improvement process. Adopting a continuous improvement process with teams allows them to make any change required in a structured and measured way. The first step is to create a value stream map of your software delivery value stream. The bottlenecks that are identified can be prioritized based on the aspect that has the most added value for achieving particular business goals. You should update your value stream map on a regular basis, for example every three months. </>

Martijn van der Sijde

Jasper Gilhuis

Migrating to Microsoft Azure?

Stay in control of your cloud transformation with
StackState Next-Gen Monitoring and AIOps

Microsoft Azure +

Accelerate Business Aligned Cloud Transformation

Moving complex systems to the cloud can be a massive undertaking. StackState ensures success of your Azure cloud transformation. Plan, migrate and run with confidence.

StackState seamlessly integrates with Azure cloud computing services and provides valuable visibility across your private cloud, public cloud or on-premise environments.

Auto-detect issues and immediately know what you can do about it. Even if they cross cloud and on-premise boundaries.

Make informed cloud transformation decisions with a precise picture of your dynamic environments. Optimize and realize value from your cloud initiatives.

www.stackstate.com

StackState
a Xebia Group Company

Put teams in control with Azure Security as Code

DevOps is the current trend in software development where autonomous teams should own the full life cycle of a product. In short, this means they need to build and run the product they own. Teams become responsible for writing the code, managing the infrastructure, monitoring the application health, and supporting the product. By using autonomous DevOps teams, organizations hope to be able to respond to the ever-changing demands of their clients and be able to differentiate from their competitors.

Authors René van Osnabrugge & Geert van der Cruijsen

What are the options for DevOps teams to perform permission management?
But how do we enable DevOps teams to manage permissions? There are several options to consider so let's take a look at a few examples for a team working on Azure.

Firstly, teams could utilize the Azure Portal to manually manage the permissions. That might work for small organizations and startups, but as soon as you start to scale up, or when audit regulations come in to play, the portal is very limited in functionality. It does not support audit trails, and it requires high privileges for many users. And by allowing certain users to be Azure subscription owners, the risk is that they might break other infrastructure by carrying out manual changes.

The most important downside is that this approach requires a lot of manual labor and it is impossible to reproduce the current state if you would need to recreate your entire infrastructure.

Hello ticket system!
When permissions need to be limited to fewer people, a second solution that is being used a lot in non-DevOps cultures is a ticketing system. These kinds of solutions help in streamlining the process but have some major downsides. When teams with the special privilege of setting these permissions have to support many teams, or get a large number of requests, queues might occur and thus these requests will block the flow of development teams.

Another downside is that these forms and tools are far away from the code and tools the development teams love to use. And last but not least, there is only a paper audit trail of what was requested, but this has no relation with what actually happened. There is no way to recreate the environment from the audit trail in the ticket system.

Automation is key
The best option is to use automation for these kinds of repetitive tasks. We've seen several companies build their own self-service portals and have even helped some of them release these types of solutions. These self-service portals can streamline the permission requests in a far superior way compared to ticketing systems, because all requests are automatically handled by the application instead of ending up in a queue for a human to process them. This automated process can be extended to match your own requirements.

However, an automated self-service portal also has some downsides. Building such an application requires quite some coding effort, and since we're working with permissions and access, security bugs can be easily introduced. And, if you want to build more complex requirements such as audit trails or 4-eyes principles, your application becomes more complex, which comes with substantial maintenance effort. Another downside of automated self-service portals is something it shares with the ticketing system: It is still a portal somewhere on the inter/intranet that is far away from the tools the developers like to use.

Our Goal: Rethink the way in which development teams interact with organizational silos

Permission management is one of the many examples where development teams have a dependency on another team, department or system. Other examples are firewalls, identity management, and computing resources. In our case, we wanted to build a better solution for development teams to perform permission management, but similar solutions can be created for other scenarios.

In our scenario, we wanted to create a solution for development teams that would tackle the difficulties of managing permissions with a number of key goals in mind:

A solution that works for developers

Our solution should be something that is close to developers, integrates in their workflow, and does not require all kinds of web applications that are located somewhere on the inter/intranet.

We did not want to customize too much ourselves but we had a number of requirements that we considered as "must haves". These requirements were things like audit trails, 4-eyes principle on changes, automated deployments, and versioning on changes

And last but not least, it should connect to developers. What do developers love most? Code! So the solution we build has to be based on code. Not just any code, but code that can easily be read by developers and by non-developers.

What can building a solution based on code offer us?

The most frequently used communication mechanism between business and developers is through documents. Documents that describe what needs to be done. A very nice approach to make this a bit more structured and readable by both humans and machines are structured files, For instance, YAML, XML or JSON, which is human and machine readable.

When requirements like permissions, firewall ports, etcetera are written down in a structured format, all of a sudden this can be implemented automatically by a machine. When we describe our permissions in structured files, which is essentially just code, developers can change it within their development workflow without having to make changes in other solutions.

Let's take a look at an example file:

```
resourcegroup: xpirit-asac-article
- userPrincipal: Asac-Group-Owners
role: Owner
- userPrincipal: Asac-Group-Readers
role: Reader
- userPrincipal: asac-user-01@road-
toalm.com
role: Contributor
```

This snippet, with a few lines of code, describes the end state of users and groups in a resource group. When you look at the snippet, you see straight away what the resource group looks like. If you compare this with requests in a ticketing system, you would have to take the start state and all changes together to see what the current or end state should look like.

This is great, but the main advantage of a structured file is that it can be shared with the team, the business and operations. And even better, it can be interpreted and executed by a machine. This makes recreating environments a lot easier as well.

Version Control as auditing tool

Now that we have a solution for writing the requirements in code, we should also have a solution for the audit trail, 4-eyes principle, and versioning.
Here is where source control, in our case Git, comes in. Combining code with Git, which most developers are already accustomed with, gives us most of the requirements we wanted, and for free! Git has built-in functionality for versioning, branching and audit trails. Most Git servers have pull request features which give us reviews, approvals or 4-eyes principles. And when we add automated builds and releases, we can also add automation to process these code changes into environment changes.
The last advantage of having the permissions in code is that the team itself is in full control of what the changes will be. They do not require certain admins or central teams to manage their permissions for them.

AZURE
SECURITY AS CODE

Introducing Azure Security as Code

As mentioned before, the principle of using structured files and Git instead of manual work and ticketing systems is broadly applicable. Many systems can be unlocked by applying the same principles. To give you a hint of what is possible we would like to guide you through our use case: 'Setting permissions on Azure'.

The first thing we thought about was the technology stack. We wanted a library that was cross-platform and could be easily extended using the Azure API. The eventual choices we made for our technology stack were as follows.

YAML

We use the YAML format to store the security configuration. YAML can easily be read by people, even the not so technical ones, and it is great for merging in Git.

AZ Command Line Interface (Azure CLI)

All interaction with the Azure API is done with the Azure CLI. We choose this because Azure CLI is a cross-platform implementation that is broadly used. Microsoft promotes the API and makes sure it is always up-to-date with the latest Azure API. Alternatives like PowerShell Modules or C# libraries are sometimes lacking behind.

PowerShell Core

To write the orchestration of the scripts and make it easily usable as cmdlets, we use Cross-Platform PowerShell. This runs on every platform and is perfect to write the script flow, using the Azure CLI for doing the real work.

Docker

As an optional service, especially for people on Linux or Mac, we provide a Docker container where both the CLI and PowerShell are installed together with the latest version of the Azure Security as Code library.

Installing the software

The second thing we thought of was the usage of the library. How will people consume the library? We wanted to make this is as easy as possible and came up with two different distribution mechanisms: installing via a Powershell Module, and running it in a Docker container.

Because we understand that every use case is different, we made it an open source library so people can extend and modify it to their needs.

Azure Security as Code can be installed from the source code on Github, by installing the Powershell Module from the PowerShell Gallery or by pulling and running the Docker container from Docker Hub. This article uses the PowerShell Modules.

To get started, open a PowerShell Windows (admin mode) and install the Azure-SecurityAsCode Module. Because the modules use the AZ CLI underwater, login to Azure with the Azure CLI as well.

```
Install-Module Azure-SecurityAsCode
#login with Azure CLI
az login
```

Looking at the available cmdlets there are 3 main categories:

- Get-Asac-All[Resources] – This retrieves all resources of a specific type into separate YAML files. For example. all resource groups and related RBAC settings will be stored.
- Get-Asac-[Resource] – This retrieves a specific resource into a YAML file.
- Process-Asac-[Resource] – This applies the configuration that is defined in the YAML file.

Let's take a sample scenario to understand how this works.

Scenario: Manage your RBAC on resource groups

To make it a bit more tangible, let's walk you through a scenario in which we want to baseline the security of our Azure Resource Group. We want to let teams manage their own security without giving them the keys. Therefore we need to baseline the resource group and store the settings as a structured file in Git, where the development team can then modify it.

```
# Get all the resource groups in the
subscription and their RBAC settings
in the current
directory
Get-Asac-AllResourceGroups
-outputPath $pwd
# Get settings for 1 resource group
Get-Asac-ResourceGroup -resourcegroup
xpirit-asac-article -outputPath $pwd
```

When these commands are executed, a YAML file for each resource group is created in the target folder as follows:

```
resourcegroup: xpirit-asac-article
rbac:
- userPrincipal: Asac-Group-Owners
role: Owner
- userPrincipal: Asac-Group-Readers
role: Reader
- userPrincipal: asac-user-
01@roadtoalm.com
role: Contributor
```

Let's assume we want to assign rights to asac-user-02, make them Reader, and remove the Asac-Group-Readers from the resource group.

We update the YAML as follows:

```
resourcegroup: xpirit-asac-article
rbac:
- userPrincipal: Asac-Group-Owners
role: Owner
- userPrincipal: asac-user-
01@roadtoalm.com
role: Contributor
- userPrincipal: asac-user-
02@roadtoalm.com
role: Reader
```

After updating the YAML, we call the following Asac cmdlet.

```
Process-Asac-ResourceGroup -resource-
group xpirit-asac-article -basePath
$pwd
```

The new settings are applied to the resource group.
This scenario is for resource groups, but the same actions can also be executed for other resources, for example SQL Server, DataLakeStore, and Key vault.

Making this part of the development process

Now that we have seen how easy and convenient it is to set roles and users, we need to ask the question: "How can we embed this in the development process?"

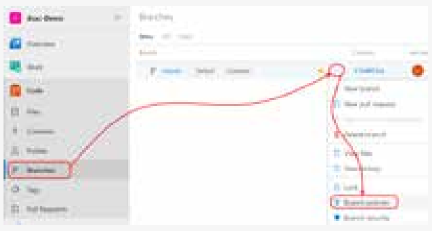
The answer is pretty simple. In exactly the same way as you treat your other code and configuration files.

The first step is to set up a Git repository that can hold all the configuration files.

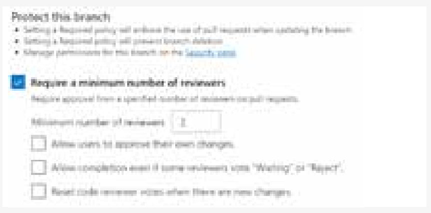
A perfect way to make your Git repository accessible to everyone is to use Azure DevOps Repos.

The next step is to set branch policies on your Git repository that allow you to control check-ins to the Git Repository. If you want teams to set and request their own security settings, but still want to have control over the process, branch policies are a perfect way to do so.

In Azure DevOps, navigate to the code repository and select branches.
On the master branch, select the Branch Policies option.



Select the [Require a number of reviewers] policy in the branch policy. Optionally add Automatic Reviewers to have someone from the security team review all the changes.



Protect this branch
- Setting a Required policy will enforce the use of pull requests when updating the branch
- Setting a Required policy will prevent branch deletion
- Manage permissions for this branch on the Security page

[✓] Require a minimum number of reviewers
Require approval from a specified number of reviewers on pull requests.
Minimum number of reviewers 2
[ ] Allow users to approve their own changes.
[ ] Allow completion even if some reviewers vote "Waiting" or "Reject".
[ ] Reset code reviewer votes when there are new changes.

Once you have set up the branch policy, you need to make sure the policy is applied once you have changed this. Of course there are many ways to do this. The easiest way is to have a Continuous Integration build that runs every time a change is merged to the master branch.

Just configure a build with a Powershell script that runs the Process-Asac-[resource] cmdlets.

What else can we do?

The way forward in a DevOps world where automation is key, is by moving towards a model in which configuration is stored as code. By using Git and Build Pipelines as a mechanism to move configuration changes to production offers a lot of benefits. First and foremost the auditability and traceability of a change, and secondly it is a nice and easy review mechanism.

But Configuration As Code can bring more benefits. You can use it to describe end-state, which enables you to rebuild things from scratch without having to know all history. And you can use it as living documentation of your system, or use the files as baseline to validate changes.

Azure Security as Code is one example of how to use development methodologies as a way to enable development teams. But of course this does not only apply to our specific scenario. Rethinking the way in which people interact, replacing humans with machines and manual actions with automated processes is the real take away. You can do *this by storing settings in a structured format, choosing* the right technology stack, and enabling your end-users to stay close to the tools they use and love. Because in the end it is all about enabling people to be more productive and deliver more value to the end-customers.

Would you like to contribute?

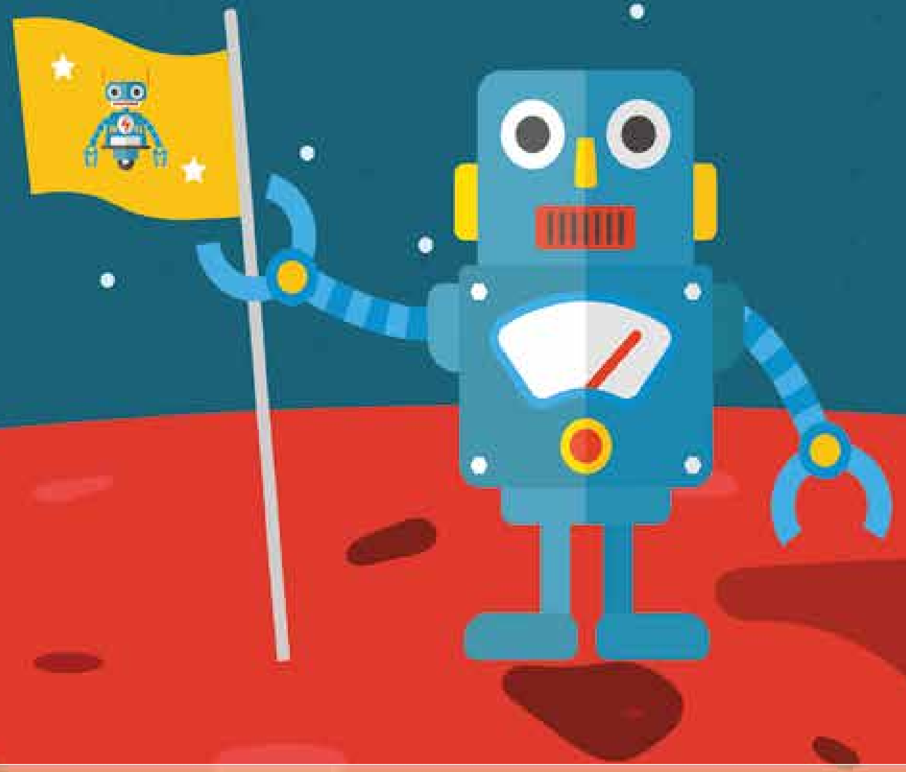
If you want to contribute to the library, please take a look at our Xpirit Github page https://github.com/XpiritBV/azure-security-as-code. </>

Rene van Osnabrugge

Geert van der Cruijsen

TECHORAMA

Taking your knowledge to infinity and beyond in 2019



Techorama Belgium
27 - 29 May 2019
www.techorama.be
@techoramaBE

Techorama Netherlands
Autumn 2019
www.techorama.nl
@techoramaNL

www.xpirit.com

Think ahead.
Act now.

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