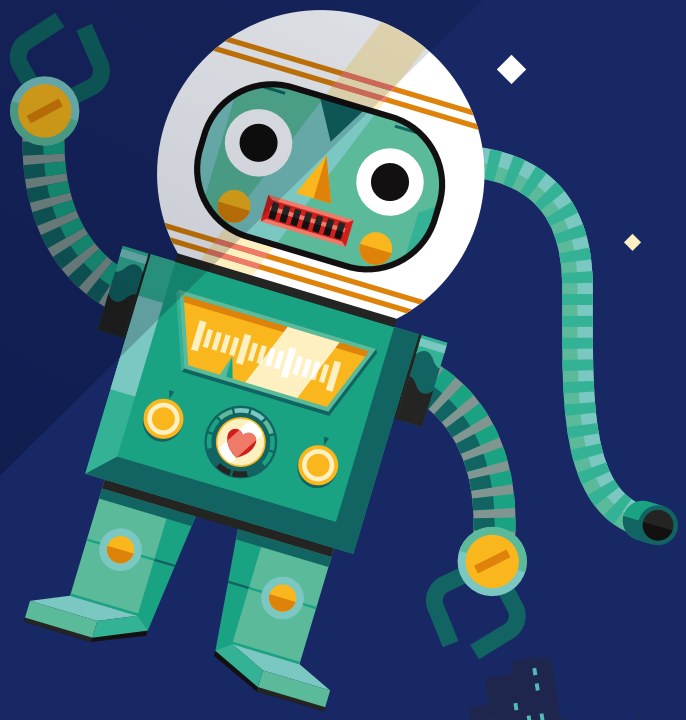
XPRT.

Magazine N°9/2019

TECHORAMA SPECIAL EDITION

The Rise of Software Complexity



Disaster Recovery Options for Azure API Management

Inclusive Programming Education

How API Thinking revolutionizes Healthcare

99% of code isn't yours

PROUDLY PART OF XEBIA GROUP

Xpirit

Think ahead. Act now.

Experts in Azure Cloud and DevOps.

Xpirit

Colofon

XPRT. Magazine N° 9/2019

Editorial Office
Xpirit Netherlands BV

This magazine was made by
Niels Nijveldt, Rob Bos, Chris van Sluijsveld, Anne Rose van Servellen, Pascal Naber, Reinier van Maanen, Maira Camu, Kees Verhaar, Arjan van Bekkum, René van Osnabrugge, Geert van der Cruijsen, Roy Cornelissen, Marc Bruins, Alex Thissen, Jasper Gilhuis, Sander Aernouts, Loek Duys, Jesse Houwing, Alex de Groot, Marc Duiker, Max Verhorst, Jordi Borghers, Marcel de Vries, Michiel van Oudheusden, Erick Segaar, Martijn van der Sijde, Thijs Limmen, Sofie Wisse, Manuel Riezebosch, Albert Starreveld, Pascal Greuter, Immanuel Kranendonk, Pieter Gheysens, Gill Cleeren, Stéphane Eyskens

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 Azure Cloud & DevOps.

The future of IT

Twenty-five years ago we built software using the programming language C++. Testing was done manually, and the way we distributed our software was on one or, occasionally, multiple floppy disks. To obtain new versions of the software you used, you made a payment to a bank account of the company you wanted the software from, and they returned an envelope with the disks included. Back then, the primary change factor in software was driven by business demand. Users wanted new things with your software, and that was what you build. Software was built within the constraints and pace of the change factors, distribution mechanisms, and the hardware that was available at the time.

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

Nowadays, there are many more change factors for your software. Even if your user does not require a change, you need to update your software because the components you used to build it, have a new version almost every day! The way you distribute your software has changed to a Software as a Service model, which completely changes the way you need to run your company. You now also run the software you build and that also needs to be efficient if you run it for all your customers. And last but not least you now need to provide 24x7 support.

Now, what is the future of IT? Confucius said: "you need to study the past to divine the future". So what has changed our industry in the past? What are the things we need to look out for that will impact the way we write and maintain software? Software is everywhere, but the ways we build software has not significantly changed. We still write in a programming language, produce some form of machine code, and that gets moved to the device that needs to run the software.

What we see is that change is the only constant, and we need a way to cope with all this change. This challenge is universal and hard to grab. It has to do with the ability of us human beings able to cope with change and embrace it instead of fighting it all the time. It is in our nature to fight change.

So if you ask us what the future of IT is, then it is mostly the need for people to embrace change. We need to change the way we work, the way we organize ourselves, the way we anticipate change. We need to understand the fact that everything we learned in IT for the past decades is often more a burden than a qualifier to be successful in an ever-changing world. The companies that have the best ability to embrace the change, understand their implications, and implement the change fast, will be the companies that will win.

At Xpirit, we have a special breed of people, who constantly search for the change, learn how to apply this, and embrace the fact this means they need to learn things over and over again. We search for the possible, not the impossible. We embrace the fact we sometimes fail and that this is a moment we learned something new. This is what we share, so we learn as a collective. We celebrate change, we embrace that we know every human being fears the change, but it is the thing that will bring us forward. Only by embracing change, we can bring the energy to create the future now.

We created this magazine to help you gain new insights and share our collective knowledge. In this edition we bring you topics on Azure Cloud (Getting Started with Pod Security Policies on Azure Kubernetes Service), DevOps (DevOps for Data Science Part II; Observability, Closing the DevOps loop) and Communities (Building an Open Source .NET Foundation). It is a variety of topics that are all intertwined. You not only need to work in communities to lead a change; you also need deep knowledge about the mechanics to make it technologically feasible and organizationally supported.

We hope you enjoy our magazine. </>

50 Shades Of Nay

When looking at the development of a product, there is hardly ever a lack of ideas and things to do. My own experience is that an average Product Backlog contains twice the amount of work that will ever be done on the product. Besides it being a system that contains a lot of waste, it also creates stress for everybody involved. The Product Owner loses grip on the Product Backlog, everyone feels constant pressure to deliver more, and stakeholders get frustrated that delivery times are so long. It's weird that a problem that seems so big can be easily fixed just by using a two-letter word: "No". However, using "no" is simple, but not easy! So, how do you do it? Let's explore this by means of an example.

Author Willem Vermaak

Meet Peter. Peter is a business analyst in the IT department of Tony. Their company is active in the automotive industry and they are about to develop a website to trade second-hand cars. Tony, as a modern Development Manager, has read all these things about DevOps, Agile and Scrum, and is adopting some of these theories into the development of the new web-site. To implement this the department has had a reshuffle, resulting in the creation of a new role: The Product Owner.

Peter, as a business analyst, has working knowledge of the product, the ability to transform ideas into concrete actionable work items, and as such, he seems to fit the role well and he gets selected as Product Owner. As Peter was already looking for a new challenge, he eagerly starts in his new role. Tony has written a clear brief for Peter about what the website should, and should not do, both from a technical and functional perspective. With this brief Peter starts to break down all types of Product Backlog items to feed to the Development Team.

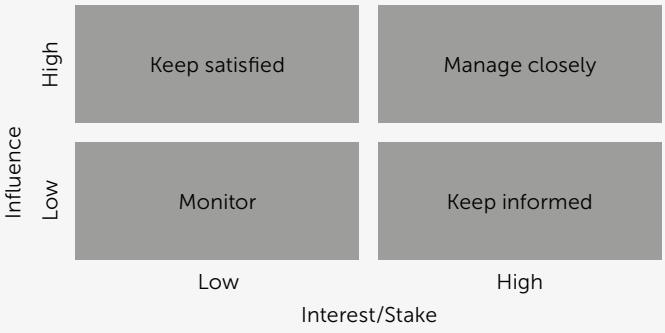
Whilst cracking away on transforming Tony's list of items, Judy walks in. Judy is a marketeer and will ensure that the website will be a great success in the market. Judy also has some requirements for the product, and she hands Peter a list with more work. Whilst she hands it over, Peter's phone rings. It's Daniel calling; the Global Head of IT (Tony's boss). Daniel heard about the shift and the new role for Peter. Besides wishing him luck in his new role he also stresses that good automation and a strong build pipeline are the core of success. As Peter thanks him, and Judy has already walked out, silence surrounds Peter.

Peter needs a short moment to catch a breath. What just happened? In less than 10 minutes multiple people walked by, called, mailed, all with requirements, wishes, tips, input, and questions. This is messy! How do I keep track of who I'm talking to? And how do I manage this way too long list of things to do?! When thinking about this he looks at his email inbox as an mail from Tony pops up: "Some help along the way!". As Peter opens the mail, he sees that Tony has arranged for John to join the team. John is a Product Management consultant hired to help Peter make his new Product Owner role successful. Things are looking up!

Fast forward 2 days – John has joined the team and John and Peter have had a chance to get acquainted. In the coming period, John will coach and mentor Peter in his role as Product Owner. The first thing John asked is whether Peter already has clarity on all his stakeholders. Who are you dealing with in this project? Who should you keep satisfied, who is truly important and who can you put on the back burner? As Peter had no good answer, John advised creating a stakeholder map.

Stakeholder mapping

A stakeholder map visualizes your stakeholders and their relationship with you. You divide the stakeholders over 2 axes: The amount of power/influence someone has, and the stake/interest they have in you and your product.



Influence means: Where are your stakeholders in the hierarchy? Is it a VP, Senior Manager, CxO? Or perhaps middle management, or somebody from the operational workforce? But besides hierarchy, some people have non-hierarchical influence. The type of people who get stuff done without

having a management label. Make sure to plot those as well. Interest/stake means: How high is the stake this particular stakeholder has in your product? Do they want - or even need - it to be a success? Also, is the stake from them to you (they want something from you) or from you to them (you want something from them).

Segmenting stakeholders like this generates four areas: Keep satisfied, manage closely, monitor and keep informed. Based on where you have plotted the stakeholders, a corresponding communication strategy can be determined. For instance, if you have stakeholders in the keep satisfied area, it means they have influence but not a major stake. Make sure you feed these stakeholders the right information at the right times and they will not become a burden. People who have a major stake but not the right influence can become very distracting, so keep them informed in the right way. When creating a communication strategy, think about these things:

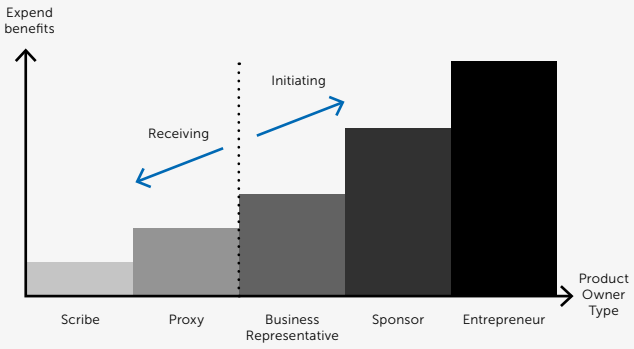
1. In what quadrant is the stakeholder?
2. Who is it?
3. What do they want from me?
4. What do I want from them?
5. What is the best communication medium? (how do I interact with them?)
6. What is the effect I want to achieve with them?

So, thanks to Johns' advice, Peter has created a stakeholder map. He has placed his stakeholders in their corresponding quadrants and is trying to manage them accordingly. Judy, the marketeer, has been placed in the Keep Informed section. Although she has a major stake in the success of the product, Peters feels she lacks the mandate to properly steer the course of the product. But he keeps receiving urgent requests from her. Every Sprint Review she pushes for new functionalities. Things like website banners, SEO tags, and she even meddles with the design. Peter is struggling as he feels he lacks the mandate to push back on her. When he tried, Judy simply went to the Marketing Manager and overruled Peter, meaning Peter still had to do it. Peter feels he is struggling with his mandate. How can I say No if they don't listen? What can and can't I decide? He decides to take it up with John when they meet later that week.

John recognizes the issue. It's hard to fully grasp your actual mandate as Product Owner. There are so many different implementations of the role. From a theoretical perspective you, as Product Owner, are the mini-CEO of the Product. No one is allowed to overrule you. Peter finds this hard to believe. Tony and Daniel have already overruled him multiple times. So how do you deal with this?

Product Owner maturity

Product Owners come in many shapes and sizes. Not every Product Owner has the mandate to make all the decisions for the product. As such, there is a tool to help understand how the various types of Product Owners work, and what their respective pros and cons are. It's referred to as Product Owner Maturity.



On the vertical axis, you have the level of added value/ expected benefits from a Product Owner, which is determined by the horizontal axis, the type of Product Owner. The higher the maturity of the Product Owner, the more benefits you will reap from the role. We see that low maturity Product Owners are often on the receiving end of work, they get told what to do, whereas more mature Product Owners are on the initiating side. They make things happen and kick-start ideas. So, the more to the right you are, the more mandate you have.

The tool categorizes 5 types of Product Owners, from Scribe to Entrepreneur. A Scribe is the most basic implementation of a Product Owner. The Scribe receives a list of work to do and hands it to the Development Team. Look at it like a homing pigeon. The Proxy is already more mature, making some decisions for the product. However, when the Development Team has a critical question or request, the Proxy does need to check with his manager, the team lead, a head of product or such to get approval or validate the question. Then, on the initiating side, the Business Representative is somebody who not only understands the technical domain of the product, but can also be a true representative of the business. Understanding how the business context works. The Sponsor is a Product Owner type who also has the budget to make decisions for the direction of the product, and lastly, there is the Entrepreneur, who has a full and total mandate over the product, its direction, the vision and strategy, and no one will overrule their decisions.

When using this tool, you should ask yourself two questions:

1. Where would you place yourself? What type of Product Owner are you and what is given to you by the organization? And then even much more important:
2. How do you act? How are you behaving?

Do you behave as a Scribe or Proxy, or do you take full ownership of your product and act as the products' entrepreneur? This is the very basis of powerful Product Ownership. If you reside in the fact you are only a Scribe, and thus act that way, you will never be seen as more than that. However, if you act as the mini-CEO of the product, people will recognize that and behave accordingly. So, what does that mean? What can and/or should you do? Here are some things to think about that can help increase your mandate:

› Do you have a clear Product Vision?

› Are you capable of explaining the strategic direction of the product?
› What are the total costs of ownership of the product?
› What are your key value indicators to know you are maximizing the value?
› How pro-actively are you managing your stakeholder map and engaging with the most important stakeholders?
› How often do you place yourself out there talking about your product, the ideas that you have and how it should progress?
› When something goes wrong, who do you blame? What could you have done differently?
› How often do you say No? And how often do you use the right no at the right time?

The last question triggered Peter. As he has his stakeholder map, he is improving his communication strategy towards the stakeholders, but he recognizes he is struggling with saying no. Too often he just agrees on what is being said as it all makes pretty good sense. That isn't helping his mandate. Actually, by looking at it closer, the more you say no, the more you create your mandate. But, then again, you can't just go around and say no to everybody all the time. Man, this is hard!

John comes around once more. Saying no - once you understand your mandate and stakeholder field - is not that hard. There are five steps to follow when you get a request from a stakeholder. Then, there are multiple versions of no, depending on the type of question and type of stakeholder.

Saying no

The five steps to get to a no:

1. Who? Who are you talking to, which stakeholder?
2. What? What is the question of the stakeholder? What are they asking? Do you properly understand it?
3. Intention. What is your intention; saying yes, no, or later/ maybe?
4. Say no. Formulate the right type of no.
5. Listen. Did the receiving party understand and agree? Is that matter done now?

As point 4 describes, there are various ways to formulate a no. One way to think about it is by categories. Selling no can be done by looking at the question from various perspectives. For instance, from a perspective of value for the user. Or from a perspective of product quality. Or from a budget or timing perspective. Looking at it this way, when a stakeholder asks for a feature, you can think about the timing, what it will cost, what the impact on quality will be. Based on the type of stakeholder (what you thought about in the first step) you will recognize that certain perspectives work better for certain stakeholders. Talking about the financial impact of a feature request will work well with stakeholders who are also concerned with the costs and revenues of the product.

So, Peter takes another look at his stakeholder map, and he takes another look at where he plotted himself on the maturity overview. He starts to understand that Tony and Daniel are more concerned about the technical quality, and somebody like Judy - as marketeer - is perhaps more concerned about the users and product value. Now, when they ask for new features, the no for Tony and Daniel can be from a perspective of product quality (i.e. It sounds like a great feature, but the technical implications would be so big that a change to the core architecture would be needed and that is not possible right now). The no for Judy can be around users (i.e. I don't have enough data to be sure the entire user base would benefit from this, so I would need more proof and/or insights before I can pick this up). This way the stakeholders might better understand why sometimes he would say no. Saying no helps with keeping the focus on the right things and enables Peter to deliver value better and quicker.

When looking at the development of a product, there is hardly ever a lack of ideas and things to do. We need to say no much more often. Having more focus on the things that matter and make a difference. Saying no is more than just starting to say no out of the blue and to everyone and everything. You need to understand your stakeholder field. Who are you dealing with? Where are they in the organization or your surroundings? In addition, you need to understand your mandate. Can I just use a blunt no, or should there be more context? Should I let the stakeholder choose between options? How do I best formulate the no?

Once you get a hold of your stakeholder field and your mandate, saying no can become second nature, enabling you to steer on true value maximization. Good luck! </>

Would you like to learn and read more?
This article is based on the content of the book "50 Tinten Nee" written by Robbin Schuurman and myself; Willem Vermaak. At the moment it is only is available in Dutch. Translations are upcoming! I'm also online, so feel free to reach out!

Willem Vermaak

From EventStorming to CoDDing

We live in a world of constant change. Today we generate more data than we can consume[1], and it contributes to the constant changes that we observe. Within the IT industry, we have the ambition to create flexible and reliable solutions that can cope with the demand for changes. We created frameworks, new programming languages, and abstracted the management of hardware with the cloud offering. Our industry provides tools and techniques that allow organizations to achieve the promised land of delivering business value to match the changing world. This is why we see many companies make a move towards microservices for mostly the same reasons; creating smaller deployable units, and to achieve a shorter and quicker feedback cycle. Moreover, many companies see the need for Domain-Driven Design to be able to do it.

Authors João Rosa & Kenny Baas-Schwegler

However, what we observe is that we still see the same code is written as follows:

```
private readonly MovieRepository movieRepository;

public MovieService(MovieRepository movieRepository)
{
  this.movieRepository = movieRepository;
}

public List<Seat> ReserveSeats(int numberOfSeats, string mvNaam)
{
  Movie movie = movieRepository.FindByName(mvNaam);
  List<Seat> avlbSeats = FindSeats(movie, numberOfSeats);
  if (avlbSeats != null)
  {
    // Gets the row of seats and adds the new available seats
    movie.Seats[avlbSeats[0].RowNumber].AddRange(avlbSeats);
    movieRepository.Save(movie);
    return avlbSeats;
  }

  return new List<Seat>();
}

private List<Seat> FindSeats(Movie movie, int numberOfSeat)
{
  foreach (string row in movie.Rows)
  {
    List<Seat> seats = DoesRowHaveEnoughSeats(movie, row, numberOfSeat);
    if (seats != null)
    {
      return seats;
    }
  }

  return new List<Seat>();
}

private List<Seat> DoesRowHaveEnoughSeats(Movie movie, string row,
int numberOfSeat)
{
  var seats = new List<Seat>();

  for (int i = 0; i < movie.NumberOfSeatsPerRow - numberOfSeat; i++)
  {
    bool seatsAvailable = true;
    for (int o = 0; o < numberOfSeat; o++)
    {
      if (!movie.Seats.ContainsKey(int.Parse(row)))
      {
        seatsAvailable = false;
        movie.Seats.Add(int.Parse(row), new List<Seat> {new Seat {Row-
        Number = int.Parse(row)}});
        seats.Add(new Seat {RowNumber = int.Parse(row)});
      }
      else if(movie.Seats.ContainsKey(int.Parse(row)))
      {
        var rowSeats = movie.Seats[int.Parse(row)];
        if (rowSeats.Count + numberOfSeat <= movie.NumberOfSeatsPerRow)
        {
          seatsAvailable = false;

          for (int j = o; j < numberOfSeat; j++)
          {
            rowSeats.Add(new Seat{RowNumber = int.Parse(row)});
            seats.Add(new Seat {RowNumber = int.Parse(row)});
          }
        }
      }

      if (seats.Count == numberOfSeat)
        return seats;
    }
  }

  return seats;
}

namespace AnemicDomain public class Movie
{
  [Key]
  [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  public long Id { get; set; }

  [Required]
  [MaxLength(500)]
  public string Name { get; set; }

  [Required]
  public List<string> Rows { get; set; }

  [Required]
  public int NumberOfSeatsPerRow { get; set; }

  // RowName plus List of seats
  [Required]
  public Dictionary<int, List<Seat>> Seats { get; set; }
}
```

On the one hand, it looks like a proper domain model, but if we look closer, the code is actually an anti-pattern described by Martin Fowler in 2003 as the Anemic Domain Model[2]. In his words, “The fundamental horror of this anti-pattern is that it’s so contrary to the basic idea of object-oriented design; which is to combine data and process together. (...) What’s worse, many people think that anemic objects are real objects, and thus completely miss the point of what object-oriented design is all about”.

What is the reason that we are still using this anti-pattern to date in complex environments? We won’t argue that in simple domains an anemic domain model works fine, only most of the software we write is intended for more complex models. One of the well-known reasons for it to happen is all the ORM examples shown on the internet.
Heck, we even fell for that trap ourselves when we started developing. Only we also have another theory, and it has to do with the way we communicate together, it is the way we do social-technical software engineering.

The common belief is that people will collaborate in open spaces, where the ideas can flow between the different persons involved in the software creation process. Thus, organizations invested in the creation of wide-open spaces, hoping the teams will deliver more value. What we observed is the opposite outcome; the communication between teams and team members decreased due to the physical work-place conditions, given that the open spaces produce high levels of noise.

[1] https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/
[2] https://martinfowler.com/bliki/AnemicDomainModel.html

Usually, people buy expensive noise-cancelling headphones and use virtual communication such as ticketing systems or instant messaging to communicate.

The issue with the way we communicate is that we are all subject to cognitive biases that screw up our solutions. One that has the most effect in these forms of communication is the confirmation bias, where we focus on information that only confirms existing preconceptions. Language plays an important role here. Any group or tribe has its own lingo, used to describe their concepts and processes. As developers, we create code to streamline the processes, trying to capture the concepts. However, if we don't listen carefully to the language, we will miss the essence of the concepts, leading to a mismatch between what the business expert thinks, and the actual code that is going to production.

Why Domain-Driven Design?

Domain-Driven Design (DDD) is a holistic approach to software development. Eric Evans coined the DDD term in his book, back in 2003; it addresses the difficulties software teams have in building software autonomously. Multiple teams design and build complex solutions as monolithic software that usually employs only one model that aims to solve different problems. He described the ambiguity between the language that the business speaks and the language coded in the model. Such ambiguity causes confusion and entanglement within and between teams.

To solve this problem, Eric created the concept of a Bounded Context, a pattern to divide software based on a model of consistent language. Within the Bounded Context, we create a shared language through conversations between business specialists and software people; this becomes the Ubiquitous Language. We focus on a language that concisely describes the situation within the domain. Instead of one canonical language for the entire business, we create several Bounded Contexts, each with their specific language and model.

Within a Bounded Context, one team can take ownership of its model and increase its autonomy as team members develop software. They can test in isolation since teams have a clear vision of who their customers are and can receive their feedback metrics. Studies[3] by Nicole Forsgren Ph.D., Jez Humble, and Gene Kim have shown that the strongest predictors of continuous delivery performance and successful organizational scaling are loosely coupled teams, enabled by loosely coupled software architecture. It makes the Bounded Context a fundamental pattern if you want to accelerate!

How do we create a shared language between business and IT?

"A picture says more than a thousand words" the saying goes. Conversations about complexity usually happen in a meeting room setting, with everyone sitting around the same table, watching a screen while being in a discussion. Most of the apparent communication in these meetings is through words. Studies have shown that the human brain processes images faster than words, and remembers visualization better than speech. It means that when we have a conversation about complexity that is more visual, the general level of ideas, decisions, and productivity will increase.

Figure 1: All you need to know about a Domain Event to get started
© EventStorming.com

It is good to know that we can use EventStorming for lots of approaches like discovering our business architecture and finding our software delivery flow. In this article, we will describe EventStorming for software design. When using EventStorming, it's always crucial to do a chaotic exploration and enforcing the timeline. For software design, it means that we need to embrace ambiguity. This process occurs between the problem space and the solution space. The problem space is our world as we perceive it, and it is the space of the business architecture, where it is independent of the software, and the language is fluid. On the other side, the solution space is the solution for the problem at hand, the world as we designed it through software architecture where we design models for creating software.

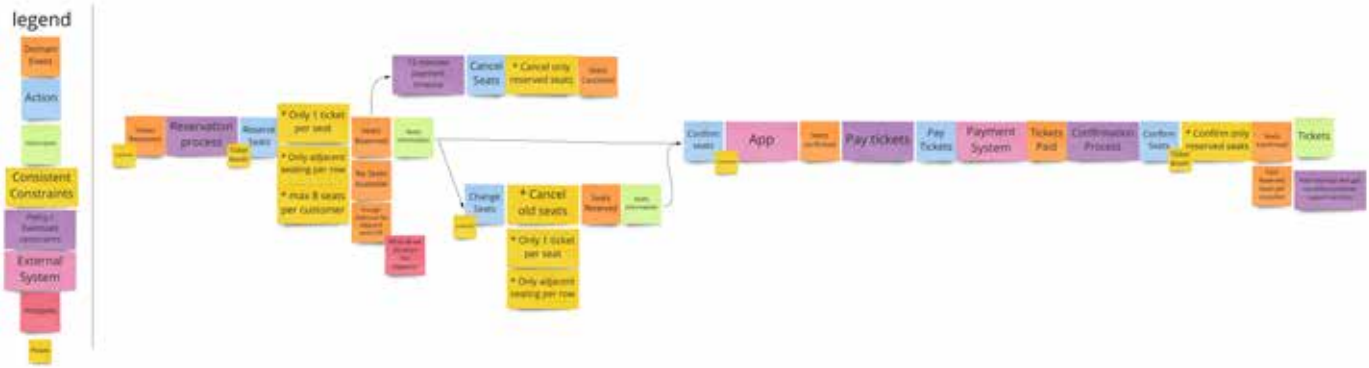


Figure 2: Example outcome of an EventStorming session on the cinema domain

[3] Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations - Nicole Forsgren Ph.D., Jez Humble, and Gene Kim ISBN-13: 978-1942788331

EventStorming for software design is a technique that iterates between problem space and solution space. As described, we embrace ambiguity when people present their perspective on the problem. From that point, the language is refined, and we make the implicit explicit. It means that during an EventStorming session, the facilitator needs to be able to steer between the problem and solution space.

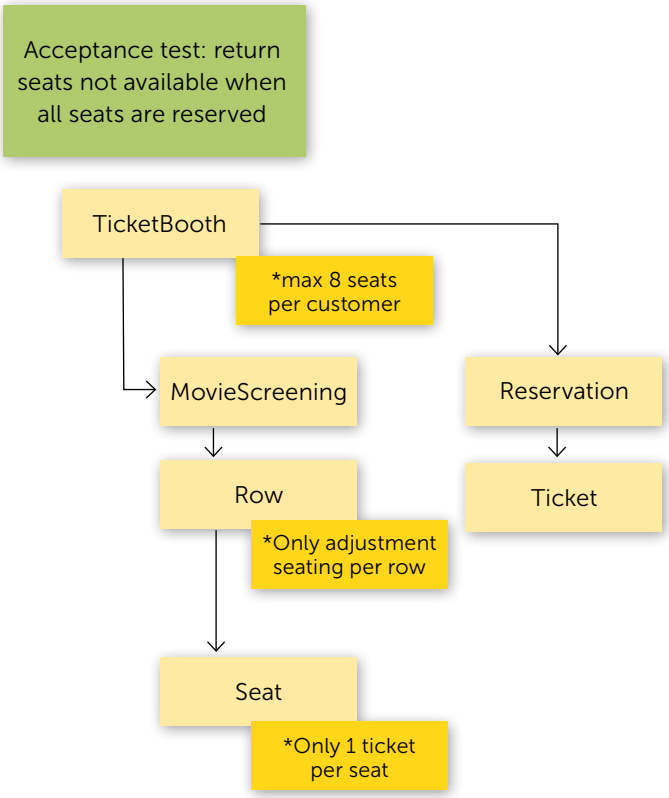
We had an EventStorming session, now what?

EventStorming gives us clues about how to design our bounded context. A bounded context is where we create a model for a purpose, and the language stays consistent — the first clue we can find in looking at the people. Different people have different needs, and we probably need to create different bounded contexts for that — the second clue resides in policies. Policies are reactions to domain events. In its essence, it means that if a domain event happened, we need to do X. Policies are always a good conversation starter. Policies are usually containers of more insights and information, and it is here that the communication between concepts takes place — the third clue you can find is in the language. We need to listen to the language spoken and what concept is meant by it. We want to create a consistent language in a bounded context. That consistent language then turns into the ubiquitous language used only in that bounded context.

Only when you believe that you have enough information and scenarios, you should leave the problem space behind and dive into the solution space and start modeling. It is of vital importance to make this decision consciously and explicitly. Eric Evans, after his seminal book (known as the Blue Book), created a good guideline for deliberate discovery, with his Model Exploration Whirlpool[4].

How do we design the bounded context model?

A vital facilitator skill is to be able to listen and filter the information. During an EventStorming session, people will use concrete examples to explain the business rules at hand. When it happens, as a facilitator, you can distil it.
Our technique is to write down the examples as they appear. You can use post-its or index cards. The collected examples are a valuable source of information, and from it, we can start to do Example Mapping either during or after our EventStorming session. Crossing these two techniques will push the group to generate more insights into the domain, allowing the development teams to decrease the assumptions, leading to better models. The examples laid down during the session are used to drive and validate the behavior of the models. Another important aspect is the domain concepts. Every time the group stops to discuss the meaning of a specific domain concept it takes the time to write it down. This information is crucial, given that it will provide clarity when creating the models. Also, it works as documentation, and it is up to the group to persist on having it in a more durable format.



From this point, we start to create object models. It is useful for the domain experts to join the session because we will further explore our language and (most probably) change the Ubiquitous Language. As a starter, we begin by putting down the domain concepts captured during the session and start making relationships between them. At that point, we link the business rules to the objects which will enforce that rule. A tip: we are still discovering, and laying down post-its; it is normal to make mistakes, and we invite you to create several potential models for the same problem. Running the examples captured before against the different object models, we can evaluate the solution, and make conscious decisions (trade-offs). Once we have the feeling that we explored enough and fine-tuned the language and concepts, we can start designing our models with the Model-Driven Design building blocks. At this point, the development team does not need the domain experts.

Building blocks of Model-Driven Design

When Eric Evans wrote the Blue Book, the prevalent programming language paradigm was object-oriented. However, it doesn't mean the building block is object-oriented. If you want to know how to do it in a functional language (F# in .NET world), we recommend reading Scott Wlaschin book "Domain Modeling Made Functional". We model in an interactive way, and we don't follow a clear path. In this section we will describe some key heuristics used in our decision making process.

From the building blocks of Model-Driven Design described by Eric Evans, we will use a subset of it: Value Object, Entity, Aggregate, Aggregate Root and Domain Service. After his first book, Eric wrote a reference book, bundling all the patterns[5].

[4] http://domainlanguage.com/ddd/whirlpool/attachment/ddd_model_exploration_whirlpool-2/
[5] http://domainlanguage.com/ddd/reference/

```
internal class Seat : ValueType<Seat>
{
  private readonly SeatStatus _seatStatus;
  internal RowNumber RowNumber { get; }
  internal SeatNumber SeatNumber { get; }

  private Seat(RowNumber rowNumber, SeatNumber seatNumber, SeatStatus seatStatus)
  {
    _seatStatus = seatStatus;
    RowNumber = rowNumber;
    SeatNumber = seatNumber;
  }

  internal bool IsAvailable => _seatStatus == SeatStatus.Available;

  internal static Seat CreateAvailableSeat(RowNumber rowNumber, SeatNumber seatNumber)
  {
    return new Seat(rowNumber, seatNumber, SeatStatus.Available);
```

Example of a value object

```
public class MovieScreening
{
  public MovieScreeningId MovieScreeningId { get; }

  private IList<Row> _rows;

  private MovieScreening(uint movieScreeningId, int numberOfRows, int seatsPerRow)
  {
    MovieScreeningId = movieScreeningId;
    _rows = new List<Row>();

    for (var rowNumber = 1; rowNumber <= numberOfRows; rowNumber++)
    {
      var row = Row.CreateNewRow(rowNumber, seatsPerRow);
      _rows.Add(row);
    }
  }

  public SeatsReserved ReserveSeats(ReserveSeats reserveSeats)
  {
    var rows = new List<Row>();
    Row rowWithAvailableSeats = null;

    foreach (var row in _rows)
    {
      if (!(rowWithAvailableSeats is null) || !row.HasAvailableSeats(reserveSeats.
      SeatsToBeReserved))
          {
        rows.Add(Row.CreateFromRow(row));
        continue;
      }

      rowWithAvailableSeats = row;
    }

    if (rowWithAvailableSeats is null)
      throw new SeatsNotAvailable();

    rows.Add(rowWithAvailableSeats.ReserveSeats(reserveSeats.SeatsToBeReserved));

    _rows = rows.OrderBy(x => x.RowNumber).ToList();

    return new SeatsReserved(reserveSeats.SeatsToBeReserved);
  }
}
```

Example of an aggregate root

“EventStorming is about merging the people and splitting the software using bounded contexts.”

Alberto Brandolini

Value Objects are simple because they are stateless, so one of our heuristics is trying to design everything as a Value Object. We can only do this if we only care about the reference and logic of the object. When its identity rather than its attributes distinguish an object, we need to use the Entity building block; this is another heuristic that you can use. Whenever we need to protect our business invariant over several objects, we cluster these into an Aggregate. We choose one entity to be the Aggregate Root and allow external objects to hold a reference to the root only. Some concepts from the domain aren’t natural to model as objects, and for this, we wrap the logic as a Domain Service. This last heuristic is essential, given that not everything needs to be an object (in the end people are developing in an object-oriented paradigm), and you need to balance the trade-offs. You can find the code at GitHub[6].

Conclusion

The problem with engineering teams is never the technical knowledge; it is the domain knowledge. Domain-Driven Design and microservices won’t help you if we don’t start to collaborate with the domain experts regularly and create a shared mindset. To do that, we need to find more straightforward and more accessible tools to collaborate. EventStorming is a simple tool to learn that can quickly give us enough knowledge as well as a shared team mindset. Quoting Alberto Brandolini “Software development is about learning, working code is a side effect”. We need just enough upfront design for our software, so that we can decrease the assumptions and increase the value delivered to the end-user. It is the cornerstone of an Agile mindset, with which we need to inspect, learn and adapt. </>

João Rosa

Kenny Baas-Schwegler

[6] https://github.com/joaoasrosa/xpirit-magazine-fromeventstorming-to-coddding

Disaster Recovery options for Azure API Management

APIs are becoming mainstream in most organizations, which is why API Management solutions are in high demand in order to standardize the way APIs are published and also to enforce some security policies. In this article, I will focus on a recurrent requirement that is ensuring some business resilience and minimizing the impact of a service outage. The requirement involves an architecture that supports a Disaster Recovery (DR) scenario.

Author Stéphane Eyskens

A common misconception is to think that it is up to the Cloud provider to make sure no service outage will take place. However, the reality is that some system maintenance has to be performed which causes some planned downtime, and this is a shared responsibility. The Cloud consumer has to ensure proper home work has been done to be resilient to both planned and unplanned downtimes (outage), and sometimes even to severe outages, impacting an entire region. In the rest of the article, I will first highlight the risks we are trying to mitigate and the various options we have at our disposal to ensure an appropriate response.

First things first. What are the risks?

What exactly are the risks we should mitigate when running Azure API Management? The first thing to check is the Service Level Agreement (SLA) associated to the service[1].
In short, Microsoft guarantees 99.9% of service availability, leaving room to approximately 8.76 hours of service unavailability over a year, which is not bad. However, if you have a higher requirement, you may need to opt for another solution.

Beyond the SLA of the service itself, there might be circumstances in which unexpected events occur:

- A regional outage: when such an event occurs, although rather rare, all the service instances deployed to that specific region become unavailable and unresponsive.
- A customer-specific APIM instance is not responsive.
- A customer-specific tenant encounters some issues not observed elsewhere.
- A dependency is not responding. For instance, you might have a gateway policy that reaches out to a third-party service or to a custom backend whose purpose is only to validate a token or to enrich a received token, etc.
- The backend services that are published to the API gateway might become unhealthy. It might be because of a service-specific issue (Cloud provider side) or because of poorly written code (Cloud consumer side) that does not sustain load very well.

The above list is certainly not exhaustive, but it already gives an idea of all things that could go wrong and for which we need to find compensation mechanisms.

APIM's pricing tiers

APIM comes with the following pricing tiers:

Developer	No SLA
Basic	SLA 99.9%. This tier is described by Microsoft as entry-level production use cases
Standard	SLA 99.9%. This tier is described by Microsoft as medium-volume production use cases
Premium	SLA 99.9%. This is described by Microsoft as High-Volume and Enterprise production use cases. At the time of writing, this is also the only flavor that integrates with Azure virtual networks.

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

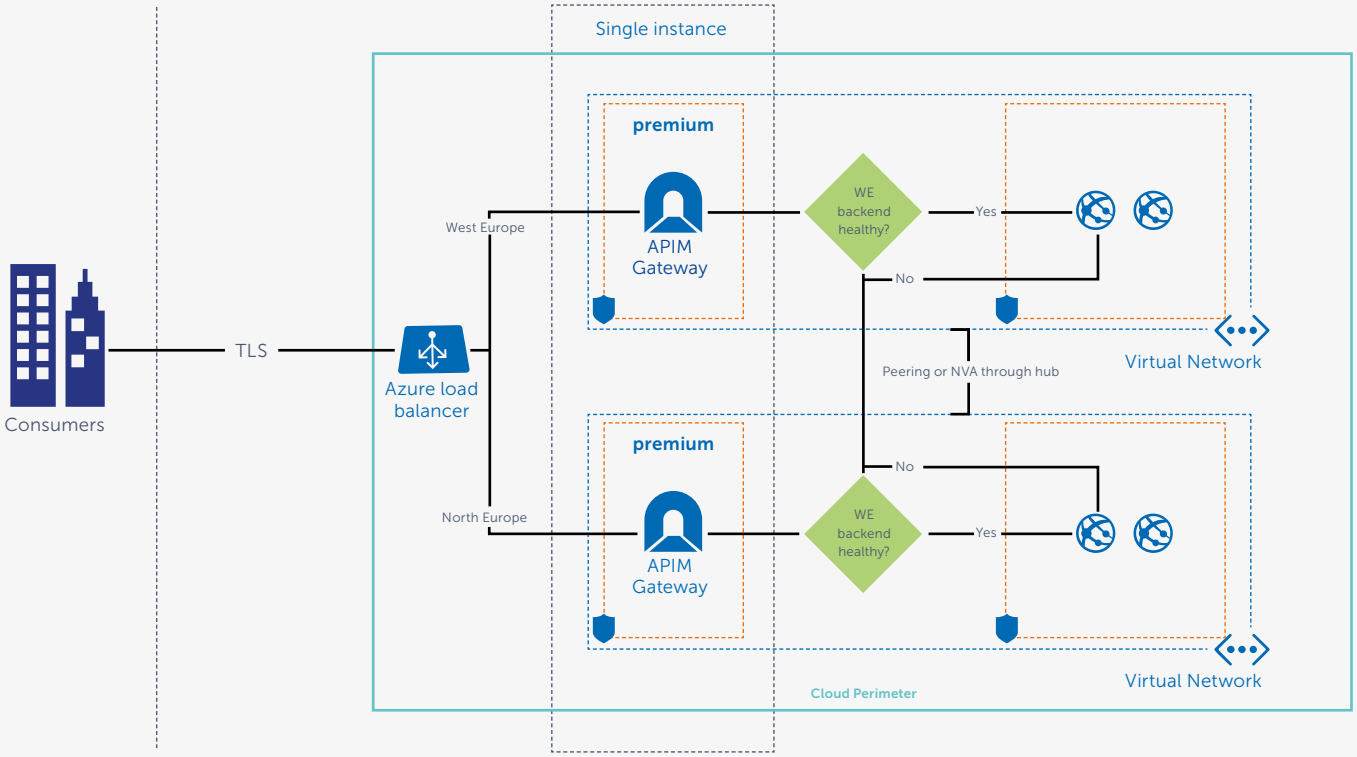


Figure 1: Disaster Recovery with APIM Premium

At first sight, we might think that the only difference between those tiers is the volume of requests they can handle since the SLA is the same for all. However, one of the major differences is the fact that the premium tier is the only one that can span across regions using multi-region gateway units. To understand what it means, let's see what composes an APIM instance:

- The instance itself, holding the overall configuration: policies, products, subscriptions, etc. and that is bound to a git repository.
- The API gateway that is in charge of applying policies and forwards the incoming requests (coming from API consumers) to the backend services.

The premium tier only offers to span multiple gateway units across different regions, but the instance itself remains in a single region. This means that in case of a regional outage, other gateway units will still handle HTTP(s) traffic but no configuration change will be possible until the region gets back to normal.

Let's see this in practice and explore some other options.

Disaster Recovery Architecture

Using the premium pricing tier

Figure 1 shows what can be achieved with the premium pricing tier.

A single APIM instance deployed to Western Europe (in this example) that has one gateway unit in Western Europe and another one in Northern Europe. By default, an Azure load balancer will route the traffic to the closest possible region.

So, if a consumer request comes from Western Europe, Azure will automatically try to forward it to the Western Europe gateway, otherwise to Northern Europe. This behavior could be changed by setting up a Traffic Manager in front of the gateways with another routing method.

An important thing to note is that APIM does not handle the routing logic towards the backend services. One must write custom policies to achieve this. Here is an example, extracted from Microsoft documentation[2] that shows the logic in action:

```
<policies>
    <inbound>
        <base />
        <choose>
            <when condition="@("West US".Equals
            (context.Deployment.Region,
            StringComparison.OrdinalIgnoreCase))">
                <set-backend-service base-url=
                "http://contoso-us.com/" />
            </when>
            <when condition="@("East Asia".Equals
            (context.Deployment.Region,
            StringComparison.OrdinalIgnoreCase))">
                <set-backend-service base-url=
                "http://contoso-asia.com/" />
            </when>
            <otherwise>
                <set-backend-service base-url=
                "http://contoso-other.com/" />
            </otherwise>
        </choose>
    </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <base />
    </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

Figure 2: Policy snippet

[2] Policy-based routing towards backend services https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-deploy-multi-region

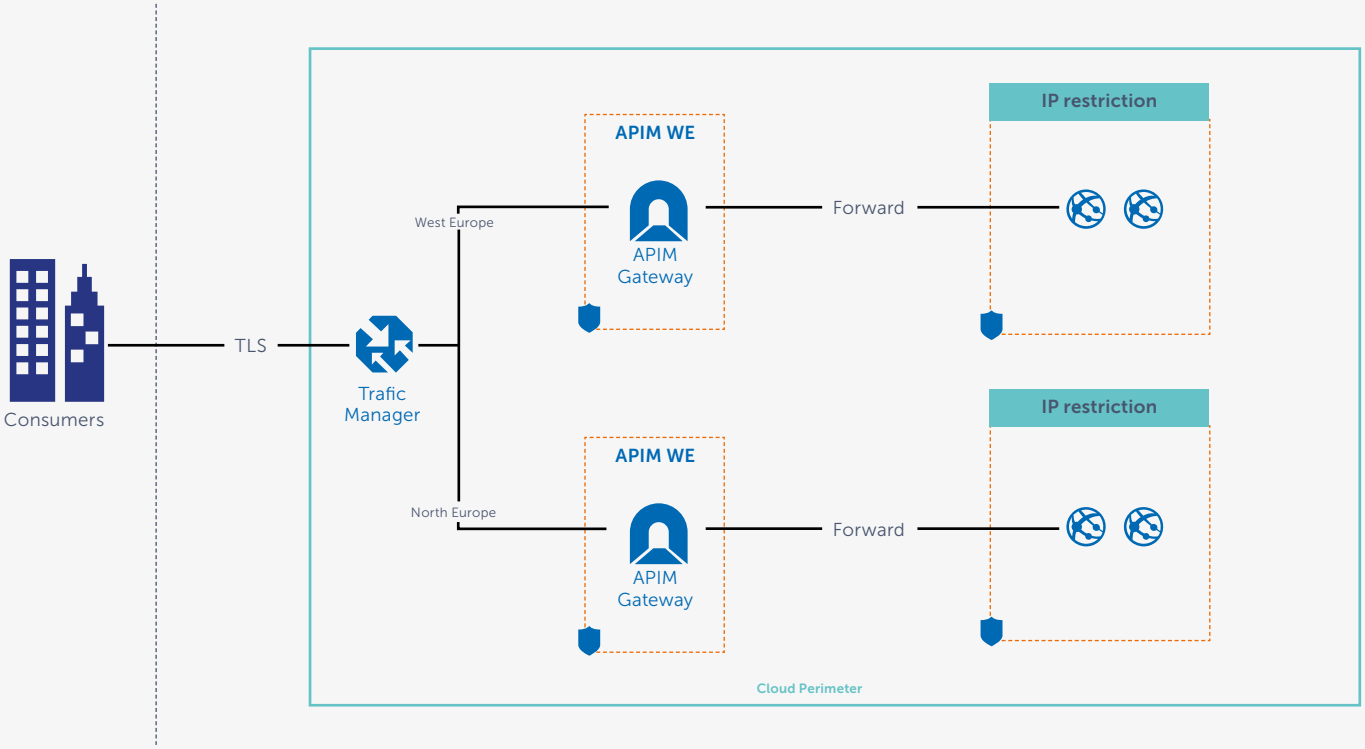


Figure 3: DR with the standard tier

But hey, isn't something missing here? What the above policy does is to transfer the incoming request to the regional backend. So, if the request is sent by the Western US gateway, it forwards to the US-hosted backends, else to Asia, else it falls back to a third region here. While this might sound logical, such an architecture sets your business resilience at risk.

What if the regional gateway is up and running but the underlying backend services are down? One must take this into account in the policy with the following logic:

› try to forward to the backend services of the same region;
› in case of failure, fall back to the other region.

There is a very good blog post[3] from Microsoft describing how to implement fault tolerance in a policy.

The benefits of using the premium pricing tier to have a DR architecture are:
› it is the easiest way to achieve DR;
› Azure has some internal plumbing that detects whether gateway units are healthy or not, and routes incoming requests automatically to the healthy ones.

The main drawback is:
› it comes with significant costs. Each gateway unit costs approximately €2000/month and of course, to be DR compliant, you need at least two of them.

Note that since April 2019, Microsoft has announced[4] a private preview of a self-hosted APIM gateway. At this stage, this is still speculation, but this could mean that we could achieve multi-region deployment of gateway units with the other tiers than the premium one. Now, Microsoft might come with some restrictions since this could defeat most reasons why the premium tier is chosen. Typically these are: multi-region gateway units and virtual network integration. By self-hosting the gateway, the latter could also be achieved easily.

Alternative to the premium tier

Because the premium tier is quite expensive, it might not be suitable in all the situations. Indeed, if a high SLA must be ensured but only a low workload is foreseen, paying 4000+ euros a month might be overkill and not every customer can afford this. I have seen customers building a DR architecture with the standard tier as shown by figure 3:

In the above diagram, two different APIM instances are deployed, each in their own region and each with their own bundled gateway unit. A Traffic Manager (Front Door would also be ok) is required to route incoming HTTP(s) requests to the gateway units. This time, no specific routing policy is required since there is no region context, although you may of course create a fault-tolerant policy as we have seen earlier. There would be a variant though: since there is no more virtual network, network peering could not be used any longer as a way to communicate between regions. Fault-tolerant policies should use the other region's gateway URL instead of the backend URLs themselves. In this scenario, backend services are using public app services with IP restrictions, whitelisting both gateways. If hosting the backend services within a virtual network is a strong requirement, you'd need a reverse-proxy between the API gateway and the backend services to bridge them all.

The main benefit of this approach is:
› Running costs are substantially cheaper. The standard tier costs about €500/month, so four times cheaper than the premium tier.

The drawbacks are:
› Since you run two different instances, the configuration should be pushed to both instances and make sure they remain in sync.
› Each APIM instance has its own user/subscriptions, meaning that one cannot use the out-of-the-box developer portal to onboard new subscribers since each APIM instance has its own portal. In case of regional outage, one must ensure that API consumers can use the same subscription key whatever region they are redirected to. It is possible to use the Product Subscription delegation feature that lets you hook a custom page with a custom logic to register subscribers. That way, the custom page can use APIM's REST API to push changes to both instances.
› Basic and Standard tiers do not integrate with virtual networks. By default, using APIM only, you can't host your backend services into a private network.
› Overall, the approach is more convoluted.

Conclusion

From my experience, the most frequent architecture to respond to DR requirements implies the use of the premium tier. The other reason that pushes organizations to go premium is the fact that, at the time of writing, it is the only flavor that integrates with a virtual network. Most companies still rely heavily on the network to secure their workloads, which leads them to host the backend services inside a virtual network, usually onto an Internal Load Balancer App Service Environment also known as ILB ASE. Microsoft documents a PCI (Payment Card Industry) compliant architecture based on an ASE[5], which is why it is a very common pattern. More than often, the premium tier is also selected for a single-region deployment, only to comply with PCI requirements.

That being said, the alternative described in this article may also be a fit, certainly for smaller customers who do not especially have PCI-like obligations. </>

Stéphane Eyskens

[3] https://devblogs.microsoft.com/premier-developer/back-end-api-redundancy-with-azure-api-manager/
[4] https://azure.microsoft.com/en-in/updates/self-hosted-api-management-gateway/
[5] https://docs.microsoft.com/en-us/azure/security/blueprints/pcidss-paaswa-overview

Getting started with Pod Security Policies on Azure Kubernetes Service

Azure Kubernetes Service or AKS, is a semi-managed container orchestrator cluster, running Kubernetes. You can use it to run different kinds of workloads, e.g. web servers and background workers. Even though it's called a 'managed' cluster, as an AKS consumer you are responsible for upgrading Kubernetes versions and rebooting nodes to apply security patches. You are also responsible for application security matters, such as running pods using the principle of 'least privilege[1]', which means that containers do not have any capabilities they do not explicitly require to run.

Author Loek Duys

If an attacker gains control over one of your pods, they can use it to attack the rest of the system. In this article you will learn how restricting pod privileges will make it much harder for an attacker to use a compromised container to attack the rest of the system.

Pod security policies

To run your containers with the least amount of privileges, you can use a tool within Kubernetes that is called 'Pod Security Policy' or PSP. A PSP defines what containerized processes can and cannot do. It works on the pod level, so it applies to all containers running within the pod. For example, a policy can be used to restrict network, disk, and access to the container host kernel. Policies are defined at the cluster level and can be applied to all starting pods automatically. This way, you cannot accidentally forget to restrict pod settings when deploying new software to your cluster.

When enabling the feature, which currently is in preview, on AKS, you will get two (cluster-level) policies straight out of the box.

1. privileged – using this policy has the same effect as using no policy at all, all operations are permitted. This policy can be useful in test scenarios, but you should use it with care.
2. restricted – using this policy applies a set of restrictions. For example, it prevents the container from running as root. After enabling the feature, this is the default policy that gets applied to all new pods.

Of course, you can also define custom policies to match your security requirements even more closely.

What they do

Pod Security Policies restrict containerized processes in the following aspects:

What	Prevent or allow...
Privilege restrictions	containers to run as *root*, or to escalate to root. Use of privileged *fsgroup* and *group*
Process capabilities	container process access to capabilities like 'CAP_NET_BIND_SERVICE,' that controls the use of a privileged network port.
Host access	containers to access processes, storage, and network on the container host.
Container root volume	processes to write to the root volume within the container.
Volumes	access to types of volumes attached to pods.
SELinux, AppArmor, seccomp	the use of these Linux security features. For example, you can use Seccomp to disallow a process from making unsafe system calls. AppArmor is used to restrict process capabilities. Note that the AppArmor and SECComp features are currently in preview.

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

For more details, have a look at the documentation[2].

How to get started?

Enabling the PSP feature applies the **'restricted'** policy to all new pods, which could potentially make your system unusable. So you should always create and apply policies first and enable the feature second. This way, you won't break running systems.

To make sure you don't block or break stuff in the 'kube-system' namespace, every pod deployed in that namespace can be configured to run using the built-in 'privileged' policy, which allows all rights to pods; privilege escalation, privileged ports, and read/write access to the container root file system. You can also restrict privileges by using a custom policy.

In the following paragraphs, I'll explain how you can configure your containers to support running in restricted environments. I will do this for two well-known platforms; Nginx and Kestrel.

Prevent the pod from running root

By default, a container is allowed to run as root. Running a container as root is risky because it allows complete access to everything within the container. These privileges can be used by an attacker, to break out of the container[3] and access the container host. To run a container as non-root, you must make sure it does not access resources that require privileges. For example, it must not write to protected folders.

Nginx

If you are running Nginx as a web server, have a look at the **'nginxinc/nginx-unprivileged'** image. It can run as non-root, does not use privileged ports, and does not access privileged locations on disk.

Kestrel

If you are running a dotnet core web application on Kestrel, make sure to configure it to run on a port higher than #1024. For example, you can do this by defining these environment variables in the Kubernetes template:

```
ENV ASPNETCORE_URLS="https://+:8001"
ENV ASPNETCORE_HTTPS_PORT=8001
EXPOSE 8001
```

Specify a security context in your pod definition to indicate the user that runs the containers. You could define your deployment as shown in Figure 1. The values of 'runAsUser' and 'runAsGroup' should be a number above 999, all lower numbers are usually reserved by the system.

```
apiVersiob: extebtions/v1beta1
kind: deployment
metadata:
   name: dep-webapi
spec:
  template:
    spec:
       serverAccountName: be-pods
       containers:
       - name: webapi
         securityContext:
          runAsUser: 1000
          runAsGroup: 3000
```

Figure 1: Pod security context

Prevent the pod from writing to the file system

Denying a pod write access to its file system can prevent an attacker from downloading and installing tools within a compromised container. You will need to make sure that the container does not require write access to function.

Nginx

Running Nginx without write access is tricky because it buffers large requests & responses and log files on disk. There are (un-supported) ways to get it to work[4], but I haven't had success running our application.

Kestrel

Running a dotnet core process on Kestrel can be done, but that also requires an undocumented workaround. You need to disable a feature called COMPlus diagnostics (which seems to be there for diagnostic support) by defining an environment variable in your Kubernetes template or dockerfile:

```
COMPlus_EnableDiagnostic=0
```

By applying these measures, you can now safely run such containers using the **'restricted'** PSP. But how do you create a policy and apply it to a pod?

Defining a policy

You can create a custom policy by deploying a YAML file to the Kubernetes cluster. Your restrictive policy could look like Figure 2:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: 00-restricted-policy
  annotations:
    seccomp.security.alpha.kubernetes.io/
    allowedProfileNames: 'runtime/default'
    apparmor.security.beta.kubernetes.io/
    allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/
    defaultProfileName:  'runtime/default'
    apparmor.security.beta.kubernetes.io/
    defaultProfileName:  'runtime/default'
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
    - ALL
```

[2] https://kubernetes.io/docs/concepts/policy/pod-security-policy/#what-is-a-pod-security-policy
[3] https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/
[4] https://medium.com/urban-massage-product/nginx-with-docker-easier-said-than-done-d1b5815d00d0

Loek Duys

```
volumes:
   - 'configMap'
   - 'secret'
   - 'persistentVolumeClaim'
 hostNetwork: false
 hostIPC: false
 hostPID: false
 runAsUser:
   rule: 'MustRunAsNonRoot'
 seLinux:
   rule: 'RunAsAny'
 supplementalGroups:
   rule: 'MustRunAs'
   ranges:
     - min: 1
       max: 65535
 fsGroup:
   rule: 'MustRunAs'
   ranges:
     - min: 1
       max: 65535
 readOnlyRootFilesystem: true
```

Figure 2: Restrictive policy

The first lines enable the use of seccomp and AppArmor (default) profiles by using annotations. The policy also prevents running as root and use of root groups, using a non-zero value. It also prevents access to the host. Note that the last line denies pods access to the root file system within the container.

Applying a policy

You can apply a Pod security policy to a pod, by using 'Role-Based Access Control' (RBAC). First, you create a ClusterRole that allows the cluster-wide use of the policy, as you see in Figure 3:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: restricted-policy-clusterrole
rules:
- apiGroups:
  - extensions
  resources:
  - podsecuritypolicies
  resourceNames:
  - 00-restricted-policy
  verbs:
  - use
```

Figure 3: Cluster role that allows use of policy

You may have noticed that the name of the policy starts with **'00'**; this is because policies are applied in alphabetical order when multiple policies match the pod requirements. The built-in 'restricted' policy applies to every authenticated user, so to apply your policy it must be higher in the alphabetical sorting order. Adding the **'00'** prefix ensures your policy prevails.

We now have a role that allows the use of the custom policy.

The next step is to configure a service account with that role. We can do this by creating a new service account to run pods in a namespace, and a RoleBinding that connects the service account to the cluster role, as displayed in Figure 4:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: be-pods
  namespace: prod
automountServiceAccountToken: false
---
#schedules backend pods under policy
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: restricted-policy-clusterrolebinding-be
  namespace: prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: restricted-policy-clusterrole
subjects:
- kind: ServiceAccount
  name: be-pods
  namespace: prod
```

Figure 4: Service account with role binding

We configured the pod security context to use the service account named '**`be-pods`**' using the setting '**`service-AccountName`**'. If we now run this deployment, all new pods will use the PSP named '**`00-restricted-policy`**'. Every pod that runs under this service account in the namespace 'prod' will be forced to comply with the attached policy.

I've shown a very specific way to bind a specific service account to a PSP. Note that you can use various settings in the '**`subjects`**' property to target multiple service accounts, for example, to include all service accounts inside a namespace. Read more about this online[5].

You may have noticed that we create the service account with an additional setting **`'automountServiceAccountToken'`** with the value **`'false'`**. We do this to prevent Kubernetes from providing an API token to access the management API to pods. Most containers don't need access to management API. Omitting this token from pods is an additional security measure.

[5] https://kubernetes.io/docs/reference/access-authn-authz/rbac/#rolebinding-and-clusterrolebinding

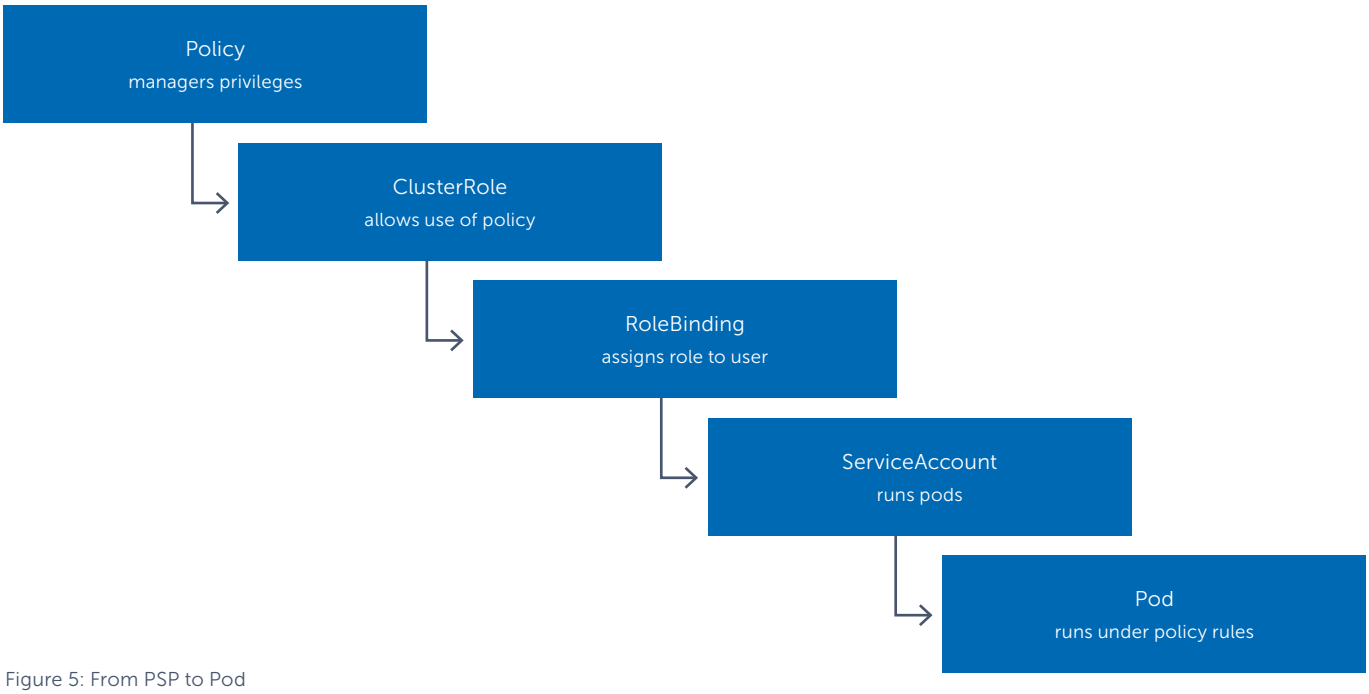


Figure 5: From PSP to Pod

Figure 5 shows a schematic flow that describes how a policy is applied to a pod.

Checking which policy is applied

You can use the **'kubectl'** CLI tool to see the effect of a policy applied to your pod. Examine the output of this command:

```
kubectl get pod/webapi-6cbd96c775-s42pq --namespace
prod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    container.apparmor.security.beta.kubernetes.io/
    backend: runtime/default
    kubernetes.io/psp: 00-restricted-policy
    seccomp.security.alpha.kubernetes.io/pod: runtime/
    default
```

Please note that the annotation **'kubernetes.io/psp'** indicates the value **'00-restricted-policy'**. This value means that the custom PSP was applied to this pod. If the pod reported the value **'restricted'**, it would mean that the built-in 'restricted' default policy was applied instead.

Checking service account role binding

You can verify that a service account is configured properly to use the cluster role by examining the output of this command:

```
kubectl --as=system:serviceaccount:prod:be-pods `
--namespace prod auth can-i use podsecuritypolicy/
00-restricted-policy
yes
```

If the output value is 'yes', the service account is allowed to use the custom policy. You can also assert that the service account can not use the built-in privileged policy:

```
kubectl --as=system:serviceaccount:prod:be-pods `
--namespace prod auth can-i use podsecuritypolicy/
privileged
no
```

Dealing with disaster

Once you have enabled the PSP feature with incorrectly configured policies, your pods may fail to start. You may have configured too many restrictions to your pods, or system services may be affected by the built-in **'restricted'** policy. If you cannot fix this immediately, you can disable the feature by using the following CLI commands:

```
az account set --subscription <<your subscription>>
az aks update --resource-group <<group>> --name
<<cluster>> --disable-pod-security-policy
```

Disabling the feature will not remove any existing policies, roles or bindings. The policies will simply not be enforced any longer.

To (re)enable the feature:

```
az account set --subscription <<your subscription>>
az aks update --resource-group <<group>> --name
<<cluster>> --enable-pod-security-policy
```

Conclusion

Pod security policies provide a powerful tool that restricts privileges assigned to your pods without you having to define rules for every individual pod. If you are not already using them, start using Pod Security Policies today, and make it much more difficult for compromised containers to harm the rest of your system. </>

The .NET Foundation: What .NET Developers Need To Know

I've been the Executive Director of the .NET Foundation since early 2017. Usually when I talk to people about it, they say something like "Wow, that's great... um... what's that?" Honestly, before I joined the team, I didn't know much about the .NET Foundation, or software foundations in general. The previous Executive Director of the .NET Foundation, .NET open source icon Martin Woodward, let me know that he was moving on to an exciting new role at Microsoft, and he wanted me to consider taking his role in the .NET Foundation. Fortunately, I got to learn about the .NET Foundation from Martin and Beth, and when I understood what the .NET Foundation is, I was excited both to get involved and to spread the word.

Author Jon Galloway

I think the best way to understand the .NET Foundation is to look at what problems and challenges it endeavors to solve.

As a .NET developer, my experience with open source started out all sunshine and rainbows: "Wow, people are just going to write software, then give it away... including the source code?" Then after a while, I started getting involved in contributing to open source projects, and that was pretty awesome, too! I got to collaborate with some top notch programmers, and we got to decide what features we wanted to build and how things would work. Then, over time, open source became more mainstream, and more companies started shipping code under open source licenses and supporting open source projects. It was great!

But after a while, I started seeing some recurring challenges:

- Projects I really liked would sometimes be abandoned. Often what seemed like a really strong, established project came down to just one or two people, working nights and weekends, and then they'd get a new job, have some kids, or just get burned out, and the project would grind to a halt.
- Successful projects would start to get overloaded administrative burdens, miscellaneous costs for things like web hosting and certificates would grow, and all of those things would distract from fixing bugs and shipping features.
- The growing interest in open source from large companies was great, but it brought some complications, too. How could community members collaborate effectively on a project that was mostly run – and funded – by a big company?

It turns out that many of these challenges aren't new, and different open source communities have already tackled them by establishing software foundations. You may be familiar with some of them by name, even if you're not sure exactly what they do: Apache Foundation, Linux Foundation, Eclipse Foundation, Software Freedom Conservancy, OpenJS Foundation, etc. These foundations are all unique in their approaches and communities, but at a high level all of them exist as independent entities that are focused on keeping an open source ecosystem healthy and growing.

The .NET Foundation is an independent organization (founded and partially supported by Microsoft, but separate) focused on supporting the open source .NET community. So let's talk about how it has approached some of the challenges I mentioned above, and then look at how we're going to build on that in the future.

An independent home for .NET open source collaboration

One of the original key focus areas of the foundation was to allow for healthy, authentic collaboration on the .NET platform that Microsoft had been developing largely behind closed doors until 2014. At that time, as .NET Core was being created as a new cross-platform development platform, it was important that it be a real open source project, developed in the open, and with significant community collaboration and ownership. This was especially important due to the history of .NET as a closed source product; in

order for the community to see this as a true open source, collaboratively run project, it couldn't be "owned" by Microsoft. So it's not – and if you look at the source code for .NET Core, you'll see that the code is copyright .NET Foundation, not Microsoft.
All developers who contribute code to .NET Core sign a Contributor License Agreement (CLA), and every commit is contributing that code to the .NET Foundation. This allows for developers worldwide – independent developers in the community, developers working at thousands of companies, and Microsoft employees – all to collaborate on the same codebase, since the source code is all under the ownership of a neutral third party whose central focus is to support the open source .NET community.

And it's worked! Since that time, we've seen a ton of community contribution and involvement in .NET Core. 87% of our contributors are outside of Microsoft, and over 61,000 pull requests from the community have been accepted. Matt Warren has written a yearly blog post series[1] where he analyzes community contribution to .NET repositories, and it continues to show a huge amount of momentum. The Cloud Native Computing Foundation shows .NET as eighth on their list of the top 30 highest velocity open source projects on GitHub.

Supporting .NET Open Source Projects

Another important focus area for the .NET Foundation has been to support community contributed projects.
The .NET community has built some really useful open source projects, but many of these are run by small teams of volunteers. In order to build a healthy ecosystem, we want to do what we can to make sure these projects can continue to grow and thrive over the long term. This is important for everyone – we want project leaders to be successful and happy so they'll keep building and releasing great projects; as consumers of open source, we want to be able to rely on projects staying around, releasing updates, fixing issues, etc.

The .NET Foundation supports over 75 member projects in a lot of important ways. One really important aspect is Intellectual Property (IP) and Legal support. When a project joins the .NET Foundation, the source code is contributed to the .NET Foundation and we set them up with the same CLA system that's used for .NET Core so all future contributions are contributed to the .NET Foundation. That's helpful for the project leaders since we can legally defend any issues around code ownership or infringement. It's also really important to consumers. Instead of depending on "one or two random people on the internet", the project is supported by a legal entity for the long term, so even if the project leaders disappear (I like to say "wins the lottery" rather than "gets hit by a bus"), the project can add new maintainers and continue on.

We also provide a lot of services to projects to cut down on administrative work and costs, so the project leaders can focus on building software.
This includes things like website hosting, devops services, certificates, marketing support, and subscriptions to a lot of online services. One example is secret sharing services like our LastPass enterprise subscription to allow project leaders to securely share passwords and other keys so that there's no single point of failure.

There's also a lot of case by case support where a project needs help with a specific issue. In one case, one day I noticed on Twitter a community project that offers debugging symbols for NuGet packages was shutting down. They'd been offering a service with both free and paid levels of support; after running that for a few years, they decided that the paid model wasn't earning enough to support the free service, and they were going to have to shut it down. I reached out to them and, together with the NuGet team, we worked to bring them on as a .NET Foundation project and run the free service using .NET Foundation Azure resources.

Another fun project was getting code signing certificates and services for .NET Foundation projects. It's a best practice for open source projects to sign their binary distributions (installers, NuGet packages, etc.), but getting a code signing service requires that your project be registered as a legal entity, and setting up code signing for builds is a little complex. Oren Novotny, who was then an advisory council member and has since been elected as a board member, came up with a great solution where we would register projects as trade names under the .NET Foundation. We worked with DigiCert, a certificate provider, to get discounted certificates for .NET Foundation project. We actually got the .NET Foundation set up as a sub-certifying authority, which allows each project to be issued a certificate in their own name.
Then we set up a code signing service on our .NET Foundation Azure subscription and brought on any of our projects that wanted to make use of it. It took several months of meetings, false starts, legal agreements, and technical setup, but it really helps out our projects. Again, this was Oren's idea, but I was really happy to be able to make it an official .NET Foundation initiative and help get it done.

Building a healthy worldwide .NET developer community

In addition to the open source collaboration and project support areas, the .NET Foundation does a lot to support the worldwide .NET community.
We set up a Meetup Pro group to make it easier to find a local group, and it's since grown to over 300 groups in 60 countries. We work with Meetup leaders to organize local events.
For instance, every September the .NET Foundation helps run an online conference called .NET Conf, and we work with our Meetup network to help them run local viewing parties and follow-on events through the end of October. We send them swag packs to

[1] https://aka.ms/dotnet-oss-community-contributions

give away to attendees, presentation materials, and help promote their events. Some Meetups have turned these events into mini-conferences that last a few days and bring in hundreds of attendees.

Scaling up

Being the executive director of the .NET Foundation is an interesting, exciting, challenging job. I'm a Microsoft employee, and they donate a lot of my time to the .NET Foundation, kind of like when a company allows an employee to contribute to an open source project. I report to our board of directors and work with our advisory council and technical steering / corporate sponsor group. I just listed all the things the .NET Foundation does; my job is to make all those things happen. I manage everything from budget and business registrations, legal agreements, new initiatives, communications, our swag store, local events, and new things as they happen. Obviously, having just one person do everything doesn't scale well, so another important part of my job is to evolve the organization to allow for more people to get involved.

Recently, we made some big changes to help the community get more involved: .NET Foundation Open Membership[2] and a community elected board of directors. The .NET Foundation has been a separate entity since it was founded, but two of our three directors have been Microsoft employees and the third was appointed by Microsoft, so it wasn't really that independent. We looked at a lot of other open source software foundations and decided we liked how the GNOME foundation worked: people who are active in the developer community can apply to become members, and the members elect their own board. So our new board has one Microsoft appointed member (Beth Massi), and the remaining six directors are members who ran for the position. Each of them serves for one year, after which they can run for re-election if they want. It's important for two main reasons:

- This very clearly gives control of the .NET Foundation to the open source .NET Community. That allows them to decide what the foundation does, and also gets the word out to the community so a lot more people can get involved.
- This is a good model to scale up what we can get done. Instead of one executive director (me) doing all the work, we now have seven board members and hundreds of community members who can form teams and work on things they see as important.

Time for Action!

One of the first things our new board of directors did was agree on some new action groups. The whole idea is to scale up, from one person doing all the day to day stuff (me) to teams of dozens of community members. They're basically committees, but I liked the name action group since committees can be really focused on talking and we want these action groups to be focused on doing. Action!

Here's the list of action groups:

- Project Support: As we bring on more projects, we need to scale up better to review new project applications, get new projects set up, handle case by case support issues, etc.
- Outreach: Focused on reaching new developers, with a special focus on diversity and inclusion. We'd like our membership, and the .NET developer community, to be available and welcoming to everyone, spanning genders, races, cultures, age groups, and blind spots we weren't even aware of.
- Membership: This team reviews new member applications, and helps figure out what membership means – benefits, ways to get involved, etc.
- Corporate Relations: .NET is heavily used in business, and this team focuses on that relationship. It reaches out to corporate sponsors, and looks at how we can better involve corporations and corporate developers in.NET open source.
- Speakers and Meetups: Now that we have a network of over 300 worldwide Meetups, this group works on connecting speakers and meetup. We're working to set up a speaker bureau, speaker grants program to cover travel, mentorship for new speakers, etc.
- Technical Review: The goal of the technical review group is to provide an independent viewpoint, separate from Microsoft, on the technical direction of our projects.
- Marketing: The main goal of marketing is to create consistent, powerful storytelling in order to increase share of voice and establish .NET Foundation industry relevance. We focus on the marketing efforts for the .NET Foundation itself and .NET in general.
- Communications: This team focuses on communicating and coordinating our regular communications with the .NET Foundation members and broader .NET open source community.

An important goal of all of these action groups is to document how we do things, so new people can get involved and we can eliminate single points of failure. In the same way that the .NET Foundation has been working for years to make sure projects are sustainable and not dependent on individuals to keep functioning, we also need the .NET Foundation itself to be set up as a sustainable and scalable organization.

As expected, there are growing pains. A lot of things are easy to do when it's one or two people following systems and policies that are only in their head. However, they take a little time to document and turn over to a team. The process has also required us to figure out how to effectively communicate with larger teams. We've settled on GitHub organizational teams for discussions and moved things like our monthly newsletter and project onboarding to public GitHub repos. It's already paying off – things are moving faster, and having more people involved is resulting in a lot higher quality. We've started with the Project Support and Marketing teams since those processes were the best defined, and in some cases partially documented; now we're working to roll those practices out across our other teams.

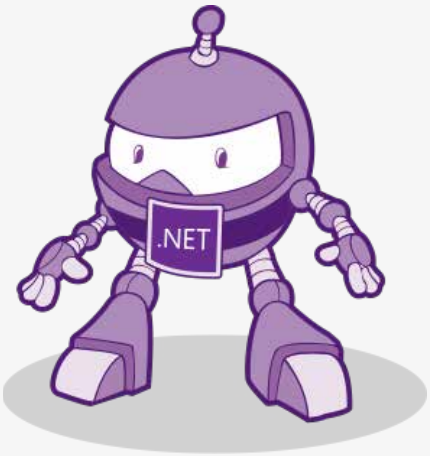
[2] https://dotnetfoundation.org/blog/2018/12/04/announcing-net-foundation-open-membership

Get Involved!

If you're interested in getting involved, now is a great time! Start by applying for individual membership[3]. In order to apply, you need to have contributed to the open source .NET community, but the requirements aren't designed to keep anyone out. Contributions may include code contributions, documentation, or other significant project contribution, including evangelism, teaching, code, organizing events, etc. If in doubt, please ask us at contact@dotnetfoundation.org. Once you've joined the .NET Foundation as a member, you can get involved in one of our action groups, and you can participate in our annual board elections, either by voting or running for a seat.

If you're a .NET developer, the .NET Foundation is for you. We exist to make sure that the .NET community is healthy and growing, and to support the projects you care about. Join us! </>

Jon Galloway



[3] https://dotnetfoundation.org/become-a-member

How to teach programming to blind children

In the Netherlands, there are approximately 3300 children who are visually impaired or blind. Within this group, and especially children between the ages of 7 and 10, there is a lack of appropriate material or methods to teach them programming. Block-based programming, often used by children without visual disabilities of this age group, is not accessible for visually impaired children. Common text-based languages, such as Python or Javascript, are still too difficult for this age group.

Authors Marc Duiker & Reinier van Maanen

We felt that these children should also be able to learn to code and not be treated any differently than the rest. To fix this, we are collaborating with researchers from LIACS of Leiden University[1] within their "Inclusive programming education" project. In this project, the researchers are looking at what materials children of elementary school-age can use to learn to program in the classroom.

This article focuses on the process we went through, where we are now and what our future plans are for this project.

How we got the idea

More than a year ago we were both in Oslo having some drinks together. Reinier was there for the NDC conference, and Marc was working on-site at a client. We discussed our work and that we would both like to promote social responsibility efforts within Xpirit. We just didn't know exactly which form it would take. Reinier attended the talk "How to teach programming and other things?" by Felienne[2] at NDC, and that provided us with some inspiration.

Goal

Our goal is to create a fun and educative experience for kids with and without visual impairment. The inclusivity is important because visually impaired children are often mixed in with non-visually impaired children. According to the researchers from LIACS, both teachers and children are asking for inclusive teaching materials.

How we started

At Xpirit, we have a couple of innovation days each year. We can use these days for anything we like as long as we share what we have learned so that the entire team can benefit from it. Some colleagues investigate the latest version of a development framework while others contribute to an open-source project of their liking. In the morning, we start with a 'stand-up' format to announce the topics and create teams. The teams spend nearly a full day on the topics and later in the afternoon they give short demos on what they achieved & learned (and how many yaks have been shaved[3]).

We started the project with a brainstorm session on how we could achieve our goal. Here are the mind maps from the brainstorm:

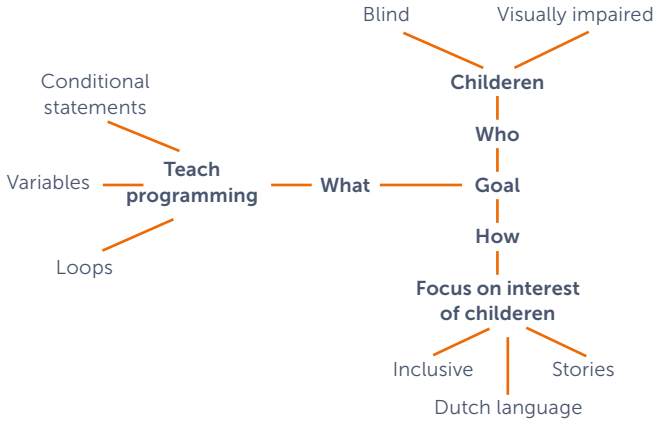


Figure 1: Defining our goal

[1] LIACS website: https://liacs.leidenuniv.nl/
[2] NDC talk by Felienne: https://www.youtube.com/watch?v=Ygk9CCRWOJs
[3] Yak Shaving: https://en.wiktionary.org/wiki/yak_shaving

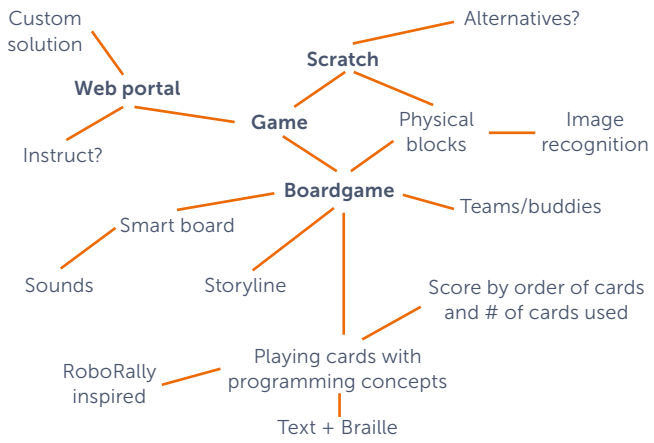


Figure 2: Exploring game ideas

After two brainstorm sessions, we started with the idea of making a physical board game with playing cards, containing both text and braille, which are used to build a programming sequence. Although it was fun to think about programming concepts in the form of playing cards and laying them out in a structure to form a small program, it has some significant drawbacks. The first one is that it's limiting the kind of programs you can 'write' with these cards. Secondly, and this was the biggest problem, there was no right way of quickly determining whether the children had placed the cards in the correct sequence. The correct answers could be presented in a separate document (again with braille), but we found that the feedback cycle from putting down cards and verifying the output was not convenient.
Parallel to prototyping the physical game, we also tried to create a quick mockup in Unity to simulate the game. However, since we didn't have much experience with this framework, it took us too long to do quick simulations of the game so we abandoned the idea.
We realized that, in order to have decent progress, we should stick to the tools and techniques we already know, or that are close to our abilities. So we decided to create a digital game instead of a physical one.

Web-based text adventure

When we started to think about a computer game we arrived very quickly at a web-based game, a web-based text adventure to be more specific. Our reasons for liking this solution so much are:

1. The web and web browsers have good support for screen readers used by visually impaired people to have the text read aloud. This means that the format of the game has to be text-based.
2. A web-based game can be played on any device, so schools do not have to invest in special or extra hardware. Lots of schools use laptops or Chromebooks these days and our game runs perfectly on these.
3. We can easily add more content and add additional features to a solution hosted in the cloud.

The advantage of making a digital game over a physical one is that we now have better control over the gameplay. We can guide the children through the game, provide help when needed and verify their input.

Another great advantage of making a digital game is that we can separate the content from the gameplay. So we're making a game engine with a generic web interface (it's all text-based) which can run different adventure stories.

The Story

An adventure game needs a story. We created a very small one just to prove the game engine works. Our intention is that new stories can be added by non-technical people. The story is a very basic escape room situation in which the user needs to find a key to unlock the door. The user needs to type in commands such as: **`Open the cupboard and pick up the key`**.

How we made it

The solution, named Louise after one of the technologies used, consists of a back-end with the chatbot and a front-end for exposing the chatbot over the web. Both the chatbot and the front-end are deployed to Azure. Besides these three things, we're also going to talk about LUIS, as that is an important part of the solution.

Back-end project

This project makes use of three major technologies:
› Microsoft Bot Framework[4]
› Microsoft Language Understanding Intelligent Service (LUIS)[5]
› Bing SpellChecker[6].

The Microsoft Bot Framework makes creating chatbots easy. It's a pretty standard ASP.NET Core application bundled with a bunch of NuGet packages like **`'Microsoft.Bot.Builder'`**. This package contains the **`'IBot'`** interface, which you implement by creating the **`'OnTurnAsync'`** method:

```
public async Task OnTurnAsync(ITurnContext
turnContext, CancellationToken cancellationToken)
{
    ...
    if (turnContext.Activity.Type == ActivityTypes.
    Message)
    {
        var dialogContext = await _dialogSet.Create-
        ContextAsync(turnContext, cancellationToken);

        if (dialogContext.ActiveDialog != null)
        {
            await dialogContext.ContinueDialogAsync(
            cancellationToken);
        }
        else
        {
            await dialogContext.BeginDialogAsync
            ("StoryPrompt", "1", cancellationToken);
            var replyMessage = _story.ToMessage-
            ForFirstScene();
            await turnContext.SendActivityAsync(
            replyMessage, cancellationToken);
        }
    }

    ...
}
```

[4] Microsoft Bot framework: https://dev.botframework.com/
[5] Microsoft LUIS: https://eu.luis.ai/home
[6] Bing SpellChecker: https://docs.microsoft.com/en-us/azure/cognitive-services/bing-spell-check/

This method is invoked with every incoming activity with the bot. The `ITurnContext` interface passed in provides access to information about the current activity and the text provided by the user. A few of the possible activities are `'Message'`, `'Typing'` or `'EndOfConversation'`. The interface also allows you to respond with another activity, for instance, a reply message.

In this particular instance, when a new message comes in, we use the `'DialogSet'` which we initialized in the constructor. A DialogSet is a collection of dialogs implemented in this bot. A dialog is a structure which the framework uses to guide the person interacting with the bot to the goal. We make use of a `'WaterfallDialog'` and a `'TextPrompt'`. The text prompt is a simple prompt for text to the user with built-in validation. Other prompts are number prompt, choice prompt and many others. The text prompt isn't used directly but is referenced by the waterfall dialog. A waterfall dialog is just a sequence of steps, in our case a practically unlimited amount of the text prompt mentioned earlier. This is the basis for our game engine.

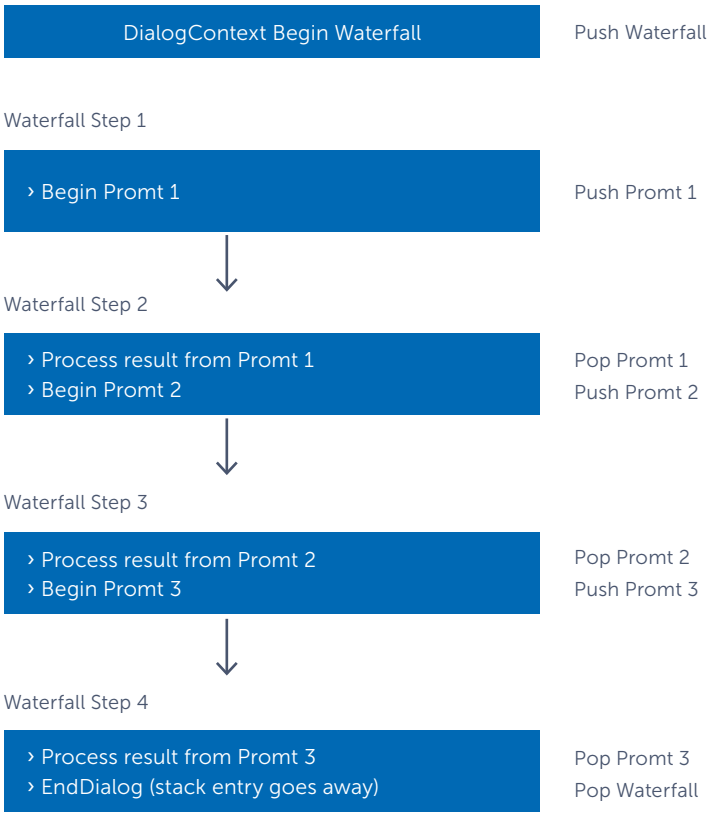


Figure 3: Waterfall Dialog

When a new message comes in, we check whether we already have an active dialog, and if not, we begin the dialog `'StoryPrompt'`, which is the id of our `'WaterfallDialog'`. If we begin a new dialog, we also immediately send a text message (with or without an audio attachment) for the first scene of the story.

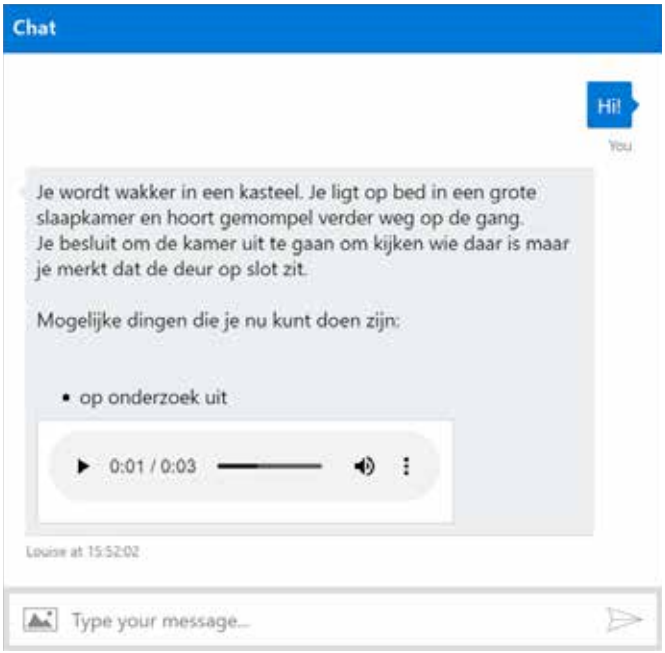


Figure 4: First scene in the story

At this point, we're in the first step of the waterfall dialog and the user is in the first text prompt. For any messages coming in afterward, we detect that there is already an active dialog and continue with that and process the answer given in the text prompt. At that point the custom validation step of the text prompt kicks in. A couple of things happen there: The Bing spellchecker corrects any mistakes in the input, and the corrected input is passed into LUIS, which is Microsoft's Language Understanding Intelligent Service. With LUIS we try and figure out what the intent of the user is. What is he/she trying to do? More on LUIS and some key concepts later.

Creating a recognizer (another concept of the Bot framework) for LUIS, with Bing spellchecking, is easy to do:

```
private IRecognizer CreateLuisRecognizer()
{
    var luisAppId = Configuration.GetSection
    ("LuisAppId").Value;
    var luisEndpointKey = Configuration.GetSecti-
on("LuisEndpointKey").Value;
    var luisEndpoint = Configuration.GetSection
    ("LuisEndpoint").Value;

    var app = new LuisApplication(luisAppId,
    luisEndpointKey, luisEndpoint);
    var options = new LuisPredictionOptions()
    {
        BingSpellCheckSubscriptionKey = Configuration.
        GetSection("LuisBingSpellCheckSubscription-
        Key").Value,
        SpellCheck = true,
        Log = true
    };

    return new LuisRecognizer(app, options, true);
}
```

This dependency is injected into our bot and we just call the method **'RecognizeAsync'**, passing in the turn context which provides access to the answer given by the user. The recognizer result, containing the intent of the user and how sure LUIS was about that being the correct intent, is then passed into a new class we wrote, the IntentHandler. That class contains most of our "business logic", dealing with all the possible scenarios. Because we felt it was quite difficult to properly unit test the bot framework code, we decided to keep the framework code separate from our code as much as possible, which is probably good practice in any case.

The final code of our custom validation looks like this:

```
public async Task<bool> CustomPromptValidatorAsync
(PromptValidatorContext<string> promptContext,
CancellationToken cancellationToken)
{
    var turnContext = promptContext.Context;
    var recognizerResult = await _luisRecognizer.
    RecognizeAsync(turnContext, cancellationToken);
    var intentHandler = new IntentHandler(_accessors,
    _story, turnContext, recognizerResult);
    var result = await intentHandler.Handle
    (recognizerResult.GetTopScoringIntent(),
    cancellationToken);

    return result;
}
```

If the intent of the user matches with a possible intent as defined in the story, the story continues with a new scene and a corresponding new message is sent to the user. If the intent wasn't clear, or LUIS wasn't sure enough of the intent, we also sent a new message, asking the user to be more specific. The last major part of implementing a chatbot is registering the implementation in the Startup.cs:

```
public void ConfigureServices(IServiceCollection
services)
{
    services.AddBot(sp => new LouiseBot(), options =>
    {
        if (Env.IsProduction())
        {
            var appId = Configuration.GetSection(
            "MicrosoftAppId").Value;
            var appPassword = Configuration.
            GetSection("MicrosoftAppPassword").Value;
            options.CredentialProvider = new SimpleCre-
            dentialProvider(appId, appPassword);
        }

        ILogger logger = LoggerFactory.CreateLogger
        <LouiseBot>();

        options.OnTurnError = async (context,
        exception) =>
        {
            logger.LogError($"Exception caught:
            {exception}");

            var message = "Sorry, it looks like
            something went wrong.";
            if(Env.IsDevelopment())
            {
                message += " Exception: " + exception;
            }

            await context.SendActivityAsync(message);
        };

        IStorage dataStore = new MemoryStorage();
        var conversationState = new Conversation-
        State(dataStore);
        options.State.Add(conversationState);
    });
}
```

Next up is the stories project, which contains all the code we have to support the various stories of the text-based adventure.

Stories project

The content and flow of the game are captured in a story file in markdown format. A story file consists of several scenes, with IDs, descriptions, sounds and possible actions that map to LUIS intents and entities.

Here's an example of a scene in a story file:

```
## [2.2, investigate the window]

You walk towards the window. You feel thick metal bars
are placed in front of it. Outside a bird flies past.
No, you can't exit the room here.

Choose what you're going to do now:
[- investigate, Investigate,,, 2]
[- investigate the door, InvestigateObject, object,
Door, 2.1]
[- investigate the cupboard, InvestigateObject, object,
Cupboard, 2.3]

Audio [crow.mp3]
```

Although markdown works well for us now because it is quick to edit, we realize we need a more structured way to persist the story to make this more scalable and manageable.
We are going to move to a cloud-based data store and an API to manage the content.

The bottom part of the story file contains instructions for LUIS in LUDown format[7]. We use a script to extract this part, and convert it to JSON which can be imported in the LUIS management portal.

Front-end project

The chatbot is exposed through a website, which is pretty easy to do. We added a web project with a single Razor page, containing just the following code which integrates with the Web Chat channel of our bot (more on that in the Azure section).

```
@page
@model IndexModel
@{
    ViewData["Title"] = "An adventure!";
}

    <div class="text-center">
        <p>Hi, say something in the chat-window below
to start an awesome adventure!</p>
        <iframe src='https://webchat.botframework.com/
embed/MviLouiseBot?s=SECRET' style='min-width: 400px;
width: 100%; min-height: 500px;'></iframe>
    </div>
```

The result is a simple chat, which works pretty well on all the devices and browsers that we tried. Autoplaying of audio files doesn't work for all browsers, but no major issues. One thing that is still on our todo list is checking the compatibility of the chat with well-known screen readers, which is a pretty major thing for blind kids, but our primary focus was to first get a working prototype; UI can always easily be changed if needed. This has been something we've been struggling with during the entire process: we're not used to developing for the visually impaired, constantly using words as "you see this or that", or wanting to use colors, images and many more that's just not possible. It's been a great learning experience!

Azure

The Azure side of things isn't too complicated either: the bot itself is deployed to an App Service. At that point, you could probably interact with the deployed endpoint directly (we didn't try), but the easy way to integrate your bot is by creating a "Bot Channels Registration". You just specify the name of your bot, its messaging endpoint (URL of your App Service + "/api/messages") and choose a pricing tier (free or standard). You can link to Application Insights as well if you want some analytics for your bot, which gives you a couple of nice diagrams showing the number of users, user retention, amount of activities separated by channel.

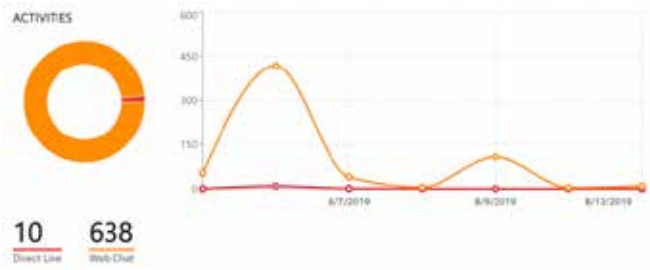


Figure 5: Bot Channel Analytics

These channels are your integration possibilities which you create inside of the Bot Channels Registration and there are a lot of options: Web Chat, Slack, Cortana, Teams, Telegram, Facebook, Email, Direct Line (custom integrations) and more. As we are exposing the bot with a website, we created a Web Chat channel. We also experimented with Slack, which works fine as well but isn't well suited for our use case as it's not easy enough to access.

Another thing you can do with the Bot Channel Registration is testing your bot with the "Test in Web Chat" option, which gives you a nice built-in chat in the Azure Portal to test whether things are working. Of course you don't want to deploy every time and while we mentioned earlier that unit testing was a bit difficult, there is another way to test your bot: The Bot Framework Emulator[8], a tool provided by Microsoft to test and debug your bot locally.

[7] LUDown: https://github.com/microsoft/botbuilder-tools/tree/master/packages/ludown
[8] Microsoft Bot Framework Emulator: https://github.com/microsoft/BotFramework-Emulator

“Our goal is to create a fun and educative experience for kids with and without visual impairment.”

Marc Duiker

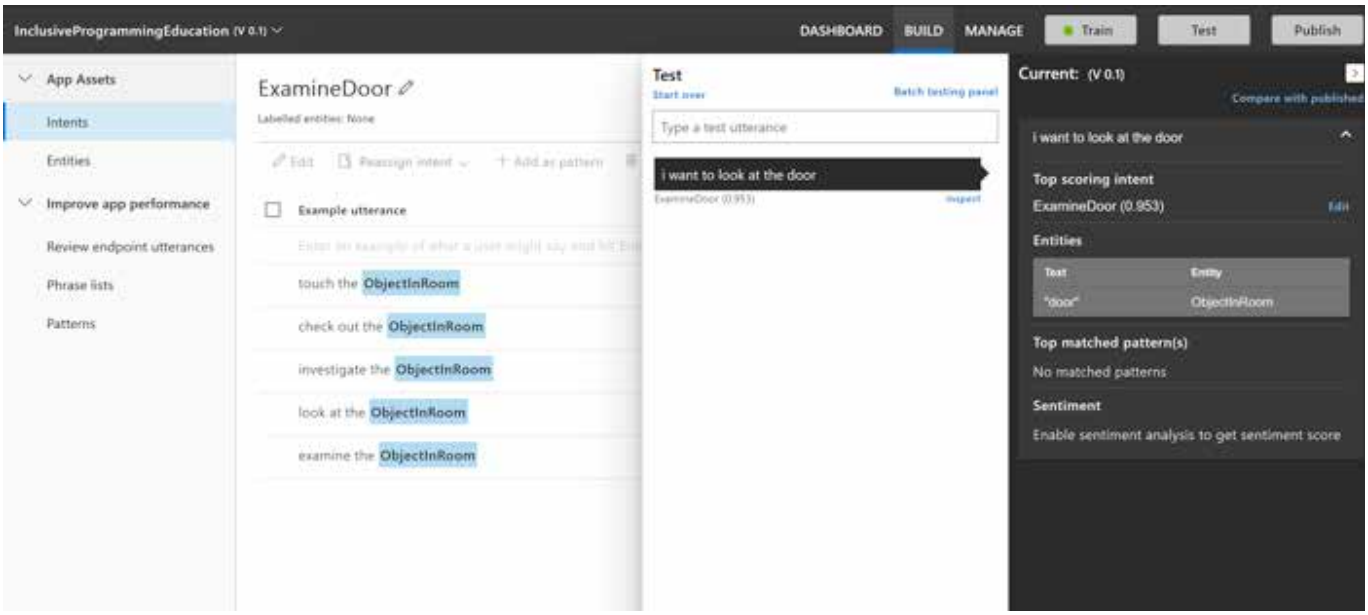


Figure 6: LUIS Intents in the portal

LUIS

So what are these intents and entities we've mentioned earlier? They are concepts from LUIS, the Language Understanding Intelligent Service from Microsoft. With LUIS you can add natural language understanding capabilities to your application. It is used frequently when creating a chatbot interface.

Intents are configurable goals you want your users to achieve. Usually, there are many different ways to describe one intent and your end-users will use different phrases to achieve the same goal. LUIS helps to understand the user input and translate it to an intent you can use in your application.

Let's take this example intent: Examine the door. Alternatives to this include: Look at the door, Investigate the door and Check out the door. They all have the same meaning.

Entities are objects which play a role in achieving the goal. In the above example, door is the entity. You can use custom Entities based on a certain type (Simple, Hierarchy or List) or use a prebuilt one, such as Age, Money or Temperature.

Once the intents and entities and have been imported from the story file we train and test the LUIS model using the portal. We found we always need to provide more examples for LUIS to understand the many variations we humans use in describing our goals and actions. Once we're satisfied with the level of understanding, we will publish the model as a service so it can be used with the bot framework.

Field testing, future work & sponsorship

At the start of the new school season, the researchers of LIACS will use our solution in a real classroom so children with and without visual impairment can test it.

In parallel, we will focus on features on our backlog such as:

- Sharing a completed story with family and friends.
- Multiplayer support so children can work together.
- A portal for teachers to:
  - manage story content
  - analyze user input to help improve the story
  - monitor story progress for an entire classroom.

We've made great progress, but there's still lots to be done. We're still looking for sponsors in order to keep working on this amazing project outside the innovation days. Please contact us if you want to support this project in any way! </>

Marc Duiker

Reinier van Maanen

Building an Open Source .NET Foundation

It was April, 3rd 2014 when Anders Hejlsberg, father of the C# language, got on stage during the keynote at the Build conference in San Francisco and released the .NET Compiler Platform ("Roslyn") as open source and made the first pull request. That same keynote, Scott Guthrie, Executive Vice President of Cloud & Enterprise group and one of the original creators of the ASP.NET web stack, announced the creation of the .NET Foundation. This was a pivotal point in .NET's open source journey which spawned the avalanche of releasing software as open source at Microsoft. This is the story of the .NET Foundation.

Author Beth Massi

On November 12th[2014], we announced .NET Core, a new open source project to build a cross-platform .NET, which started with just four libraries.

The ASP.NET web stack had been open source since 2008 as well as the F# language in 2010, but with the C# and Visual Basic.NET compiler now open source, this opened the door for the entire .NET platform. Later that year on November 12th, we announced .NET Core, a new open source project to build a cross-platform .NET, which started with just four libraries.
We wanted to "do open source right" by starting from the beginning in the open and we did it on GitHub.
The announcement landed #1 on Hacker News for most of the day, beating the Philae probe landing on a comet. It was a big deal.

Getting to that point was also a big deal. It took a lot of work from many people inside and outside Microsoft.

Looking at ASP.NET's open source history, the source code was open and the community could contribute issues and code. However, the work from Microsoft wasn't truly done in the open at the time. Bits were worked on internally and then "dropped" into the repo (Codeplex back then). Still, it was a first step into changing the way we build software.

We (the "Developer Division" engineering team at Microsoft) knew that we needed to change the approach to how we were working. We were coming off the Windows 8 hangover and most of the industry was moving to open web technologies and open standards. We needed to modernize our platform in order to grow. The only way we were going to succeed was with the help of the community.

Why did we do it?

So, why did we need an open source software foundation? It was S. Somasegar (Soma) that pushed this idea to us. Soma was the Corporate Vice President of Developer Division at the time and our executive sponsor.
Soma believed that the survival of the .NET ecosystem depended on the open source community and we needed a foundation to foster it.
He approached my manager, Jay Schmelzer, who owned the .NET Framework and languages, and we started working. We looked to the ASP.NET team run by Scott Hunter, a separate team in the Azure group back then, as the role model open source project at Microsoft. Soma knew that we needed to change the perception of Microsoft in the open source world and the creation of the .NET Foundation, and the open sourcing of the platform would prove to be a strong step.

We also had projects from the community as well as our own that needed help; not just legal and licensing help but basic development services like code signing and CI/CD. We also had customers that needed to trust and rely on .NET. I was the community manager for the .NET platform team before any of our stuff was open source. And I was on the v-team that stood up the .NET Foundation itself. We were going through a culture change internally and our customers needed to also come with us.

The challenge was to make sure we didn't lose trust – to make sure our customers understood that open sourcing .NET was not the end of the platform, but the beginning.

Many of our customers expected all the software they used to come from Microsoft. It was a direct result of us creating a hugely successful closed source ecosystem. Microsoft also didn't have the greatest track record with some of the open source projects we did release – where they were basically "thrown over the wall" and abandoned. The challenge was to make sure we didn't lose trust – to make sure our customers understood that open sourcing .NET was not the end of the platform, but the beginning. We had to get it right.

How did we start?

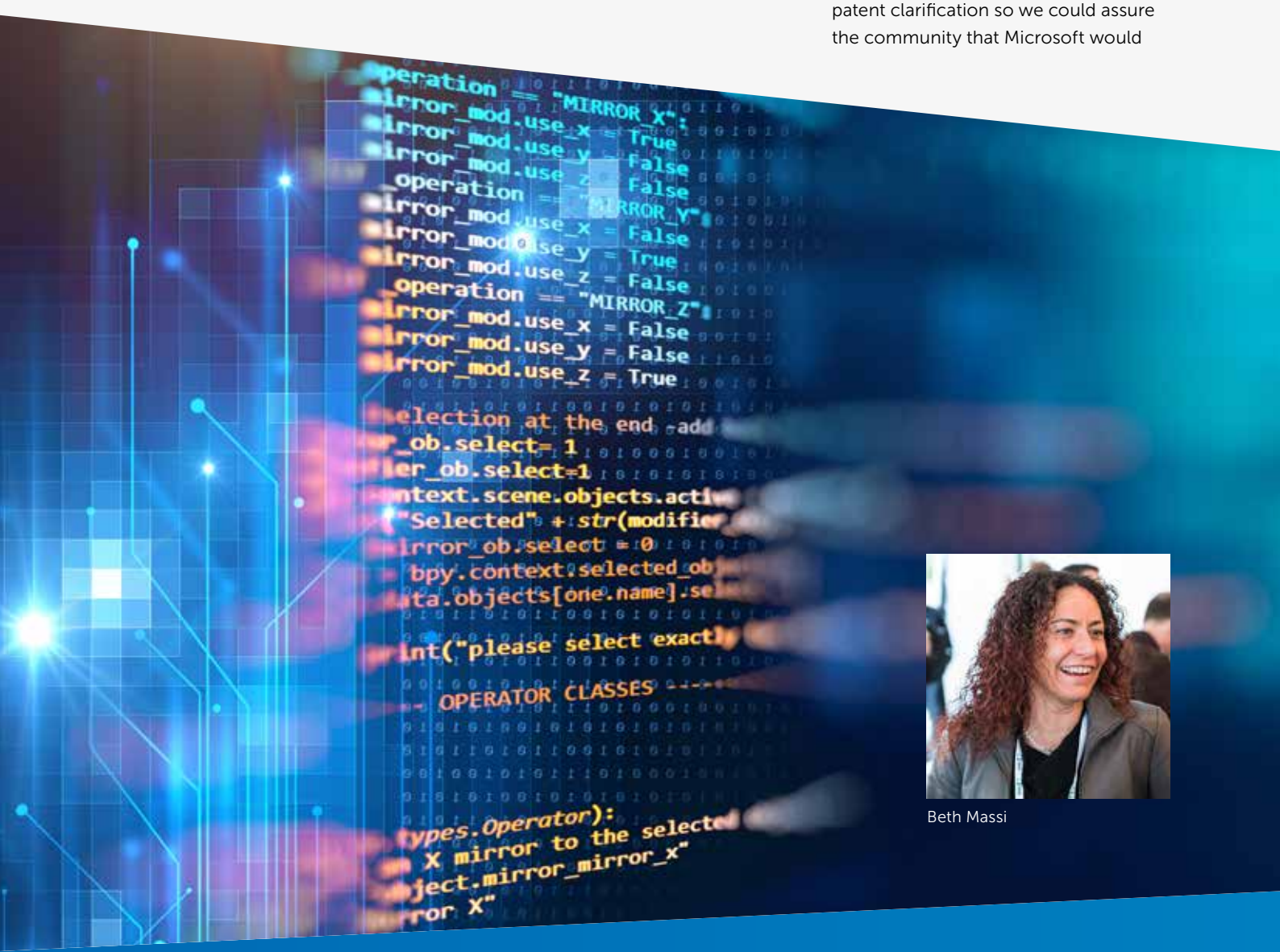
The .NET Foundation needed to be an independent organization, but it also needed heavy Microsoft backing so our customers would feel safe. We also wanted to bring in commercial partners to help us modernize the platform. Initially, Samsung and Red Hat joined us in those efforts, and then eventually we expanded these partners to form the technical steering group and corporate sponsors we have today.

Because we also had existing open source projects maintained by the community that already had their own governance models, we decided to build the infrastructure slowly and learn along the way. And let's face it, we didn't know what we were doing, so we needed to go with a modest approach to governance. There was a joke at the time; create the "minimal viable foundation". So that's what we did. Believe me when I say there were some people who didn't think we could do it at all.

We consulted lots of people. Robin Ginn, who was also on the .NET Foundation v-team, played a critical role introducing us to open source leaders. She was working for MS Open Tech at the time and has a vast network in the open source community. Many leaders including Miguel de Icaza, Ross Gardler and Jim Zemlin guided our thinking. As a community manager for a closed product line, I soaked up open source learnings like a sponge. It was a whole new world for me. The open source community is huge, and I had (and still have) a lot to learn.

The first thing we needed to tend to when we were starting the .NET Core project was the licensing of .NET Framework (our original Windows implementation of .NET).

The first thing we needed to tend to when we were starting the .NET Core project was the licensing of .NET Framework (our original Windows implementation of .NET). We needed patent clarification so we could assure the community that Microsoft would



Beth Massi

not come after anyone for using the code. .NET Framework's code is source open, meaning the code is available but we didn't take contributions back in the true open source sense (you can't make PRs). We called it reference source. We changed the license for the reference source to MIT license so anyone could copy the .NET Framework code. This was important for the Mono and .NET Core implementations.

We literally had PRs coming in the moment we opened the repo.

We knew we made the right decision right away. When we first started .NET Core the community was overwhelmingly helpful, and we literally had PRs coming in the moment we opened the repo. Within a couple of months, while we all were focusing on Linux, one person in the community, @kangaroo, added macOS support to the .NET Core runtime! We were deeply humbled by the energy. I recall someone saying that the community had increased our core team size by 60% right off the bat.

Of course, it all didn't go smoothly. Engineering leads now had to be accountable for public code reviews. We needed to have the same processes for internal and external PRs. We needed to balance internal conversations with public conversations. We needed to change our marketing strategy. We needed to figure out how to explain completely changing a direction in designs (project.json to csproj anyone?). How do we get our customers to understand the "new way of software development" from Microsoft? Making a sausage isn't pretty.

Ushering culture change

Exactly one year after announcing the .NET Foundation, we hired our first Executive Director, Martin Woodward. I was still working as the community manager and I was super excited to have someone that cares as deeply about the community as me join the team. Martin started in the Java community and has a lot of experience running open source projects and using open source software. He was a key person in changing our culture. He was actually backstage on April 3rd, 2014 at Microsoft Build making sure the Roslyn code went public on Codeplex without any hiccups, as he was the lead for Codeplex at the time. He also looked after the Microsoft org on GitHub and did a lot of other great stuff for our ALM business.

Martin worked to make the .NET Foundation real with an advisory committee and technical steering group. He created the dotnet org on GitHub and did a lot of the actual implementation of the "vision" of the foundation. Lots of paperwork. He wanted to democratize the contributions to enable anyone to contribute. He created value with project services like contributor license agreements, build and deployment services, code of conduct implementation, and conflict resolution processes. Basically, all the stuff that takes people away from making actual contributions (writing code, raising and discussing issues, writing docs...).

There were many sleepless nights looking after employee welfare and making sure we were building up the skills on our team to manage and work with the community together effectively.

There were many sleepless nights looking after employee welfare and making sure we were building up the skills on our team to manage and work with the community together effectively. Martin wanted to make sure we could innovate quickly, but still have an SLA to make our customers comfortable. This requires employee resources way beyond just people writing code. We needed "social engineers" working in our repos. We needed to build a new muscle. But it allowed us to be extremely agile and get instant feedback.

He also started the vision to create a user group consortium, to bring all the .NET meetups around the world together to teach, learn, and collaborate. He also began a blueprint for a much more open membership model, as he knew eventually the foundation would need to scale. As a community manager I worked closely with Martin. It was one of the best times and proudest moments in my career. We all worked toward making the .NET Foundation the center of gravity for .NET open source.

New role, same passion

Then I moved to product marketing. I became the Product Marketing Manager for the .NET platform in late 2015. I decided to move to marketing for two main reasons. First, after being a community manager and developer advocate at Microsoft since 2007, it was time for me to try something new. Second, I felt that the engineering team had become good community representatives themselves as part of going open source. They didn't really need me in that capacity anymore. Fortunately, I remained (and still remain) an important part of the .NET Foundation execution and strategy.

Today we have over 75 projects in the foundation.

In this new role, I worked with Martin to bring the .NET Foundation message to a much broader audience. In November 2016, at one of our big online developer events called "Connect", we announced Google joining our technical steering group and brought in a bunch more projects. Today we have over 75 projects and 550 repos in the foundation. I was also able to help the .NET Foundation by building strategic relationships and getting our presence into non-Microsoft events and placements.

New leadership, more growth

In February 2017 Jon Galloway became the next Executive Director. Jon was a developer advocate and .NET expert for many years and it was a natural fit for him to continue to drive the .NET Foundation forward. Well-known in the .NET community, he has pushed to organize our user groups scattered around the world into one cohesive community. He's brought on a huge amount of new innovative .NET open

source projects, facilitated a partnership to provide free code-signing certificates and signing services to member projects, spoken at many events, produces a lot of technical content, and has been the keystone of "running the business" for the .NET Foundation.

We've expanded our meetups to over 300 groups in 60 countries, and organized our largest online .NET Conf ever in September.

We've continued to push the .NET Foundation forward with Jon at the helm. We've expanded our meetups to over 300 groups in 60 countries, expanded our social and online footprint, conducted Hackfests and participated in Hacktoberfest, and are bringing on more projects and partners. Our annual, online .NET Conf in September was the largest ever, and we anticipate it being even bigger this year with the launch of .NET Core 3.0 on September 23 and many open source project leaders delivering sessions (see www.dotnetconf.net for details).

Jon's passion for the community has clearly shown the progress we've made. Jon is awesome at helping overworked teams streamline their processes and cutting out costs associated with building open source software.
He wants project teams, large and small, to be successful. You'll see that there is a varying degree of team sizes across the open source projects in the foundation today.

Growing up

Even with all that success, there was still only so much the foundation could do. The next step for the .NET Foundation was to scale. Microsoft was the only company providing funding for the .NET Foundation and had two of the three board seats. Although one seat out of the three board seats was a community-held position, and the advisory council and technical steering group consist of strategic non-Microsoft partners, we knew it was time to go broader and get fresh ideas. It was time to grow up.

Over the course of Jon's tenure, we've worked to make the vision Martin laid out for an open membership model a reality.

In December 2018, we announced membership model changes so that the community will directly guide foundation operations.

The Board of Directors has expanded to seven members, one seat appointed by Microsoft and the other six open to the wider .NET community.

The Board of Directors expanded to seven members, one seat appointed by Microsoft and the other six open to the wider .NET community for people to volunteer for a seat on the Board.

Board elections were completed in March 2019 and will happen annually. Any person who has contributed in any way to any .NET Foundation open source project is eligible to run for the Board and to vote. This new structure is helping the .NET Foundation scale to meet the needs of the growing .NET open source ecosystem.

We had a ton of fantastic, diverse candidates run for the board. I was truly impressed with many of the campaign pages and qualifications that each person could bring to the table.
In the end, the community elected Ben Adams, Iris Classon, Jon Skeet, Oren Novotny, Phil Haack, and Sara Chipps.

I am the one appointed to the Microsoft seat on the new Board of Directors and I promise to always have the best interests of the .NET platform and community in mind when making decisions.

What are we working on now?

Open source software foundations are important for the entire open source ecosystem, including contributors, project leaders, consumers, as well as businesses that depend on open source. The .NET Foundation's role is to provide a center of gravity for .NET open source and to make sure the code that everyone relies on lives beyond the initial creators. We also foster the ecosystem by supporting our community in many different ways.

The Board of Directors is in the process of defining action groups and committees in the following areas: Membership, Technical Review, Marketing, Corporate Relations, Community Outreach, Speaker Bureau and Meetups, and Project Support.

Right now most groups are just being defined on goals and setting up to scale out to the many volunteers. I'm leading the Marketing group with Phil Haack and we just opened up our meetings to our broader set of volunteers. It's exciting to see the passion our members have and a fun challenge to help enable them to do their best work.

How can you get involved?

If you rely on .NET and want to see the ecosystem thrive, then become a .NET Foundation member! Join in the member discussions on GitHub and help us with our action groups.
Anyone who has contributed anything to the .NET open source ecosystem can become a member. You don't have to contribute code, you could contribute to documentation, file an issue, write a blog, run a meetup group or organize .NET events. We're looking for members that have a wide variety of backgrounds, not only coders.

Get started here:
https://dotnetfoundation.org/become-a-member

Conclusion

I am incredibly excited about the future of the .NET ecosystem and honored to be on the .NET Foundation Board.
The platform is expanding and innovating constantly, our community is growing, and our customers are growing with us. I am thoroughly enjoying the ride and know that the future is very bright. I hope you get involved and participate with us!
You can learn more about .NET Foundation and get involved on the website dotnetfoundation.org. You can reach me on Twitter @BethMassi. </>

DevOps for Data Science Part II

From theory to practice using Azure Machine Learning Services

In the previous issue of XPRT magazine[1] we discussed the DevOps process for a Data Science team. We explained how the general principles in DevOps are used when developing Artificial Intelligence or Machine Learning models that can be used in a multitude of applications.

Authors Rob Bos & Kees Verhaar

We arrived at the DevOps cycle as shown in Figure 1, in which the DevOps pipeline joins the worlds of Data Scientists and App developers together. In this article we'll explore how to set up each of the steps in this workflow using components available in Azure Machine Learning Services and Python scripts.

Data Scientist IDE

It all starts with the Integrated Development Environment. All teams, and data science teams in particular, have their own way of doing things that they have cultivated and tweaked through time. The tools in Azure have been built with this in mind, and support a model of 'bring your own': whether it is your own datastore, compute power or source control methods: Azure Machine Learning usually supports them. You can choose to use Jupyter Notebooks, Data Bricks Clusters, or your Python scripts. The available solutions range from "everything on your own laptop" to "fully SaaS Jupyter notebooks" and a lot of options in between.

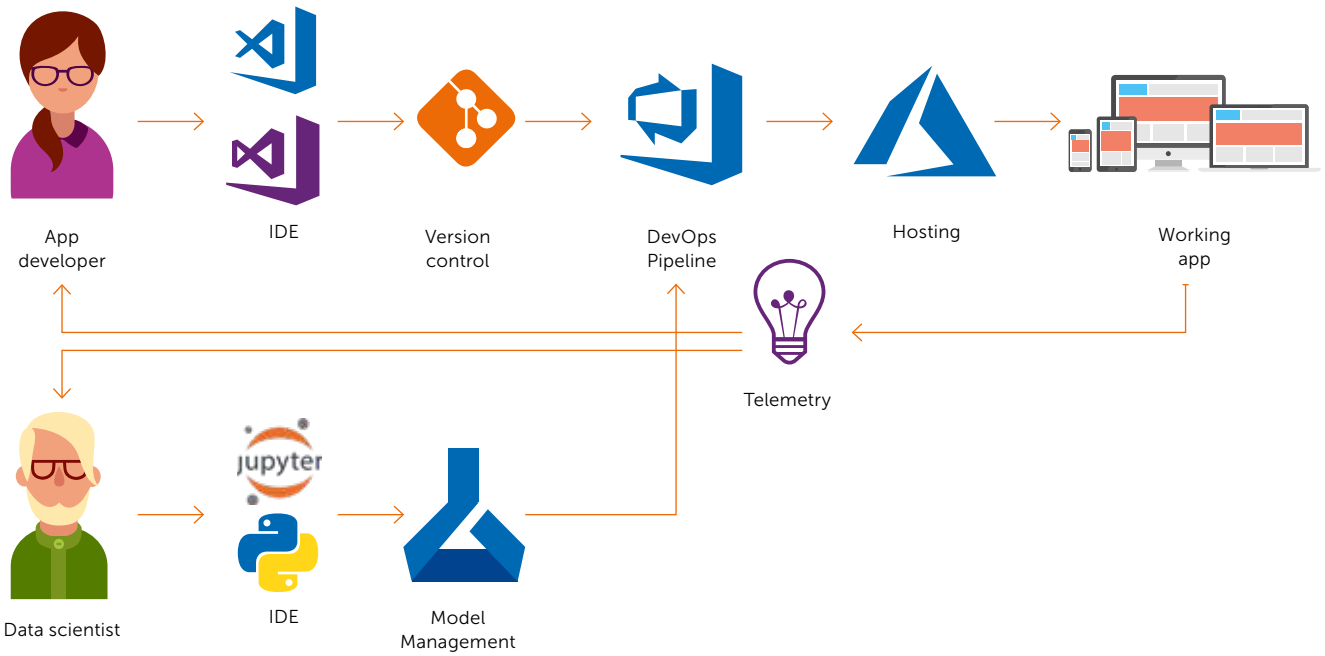


Figure 1: Implementation of the typical DevOps cycle for application development combined with Data Science

[1] https://pages.xpirit.com/magazine8

For example:

- Local computer: install all the tools you want yourself, you are responsible for maintaining everything yourself, including backups, etc.
- Data Science VM: essentially the same as your local computer, except everything is pre-installed, with the most common libraries already installed. This also is an easy and repeatable method for deploying machines in Azure with all the tools installed. When you are done with your project or analysis run, you can safely delete them.
- Python scripts: to ensure you are completely independent of a specific runtime environment.
- Azure Notebooks: hosted Jupyter notebooks, where you do not have to update the runtime environment.

These options enable you to create a process that matches your own workflow. In the rest of this article we will use Python scripts to show you an example workflow and how that integrates with Azure ML Services.

Model management

Data Science Model development is essentially a three-stage process, as shown in Figure 2: you prepare training data, then use that to build and train your model, and finally deploy it to production. Each stage has its own iteration cycle and results in components that can be used in the next stage.

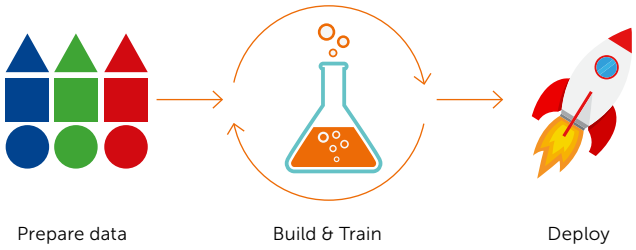


Figure 2: A typical Data Science workflow

Azure ML Services takes a key role in this process. It provides a place to manage the compute resources available to your team, automate your model training process, track model versions and results, and manage deployments.

Creating a workspace

Before you can use Azure ML services, you'll need a workspace and connect to it from your Python code. The workspace can be seen as a project or collection of everything you need in the development process of a model. You can use Azure's role-based access control (RBAC) to share the workspace with your team.

To create a workspace from Python, you import the Azure ML SDK and write a couple of lines of Python:

```
import os
from azureml.core import Workspace

# Load the parameters we need to create a workspace
subscription_id = os.getenv(
    "SUBSCRIPTION_ID", default="xprt-1234-xprt-1234")
resource_group = os.getenv("RESOURCE_GROUP",
default="mlpipeline")
workspace_name = os.getenv("WORKSPACE_NAME",
default="magazine9-mlpipeline")
workspace_region = os.getenv("WORKSPACE_REGION",
default="westeurope")

# Connect to existing workspace or create a new one
try:
    ws = Workspace(subscription_id=subscription_id,
                   resource_group=resource_group,
                   workspace_name=workspace_name)
    # write the details of the workspace to a
      configuration file to the notebook library
    ws.write_config()
except:
    # Create the workspace using the specified
      parameters
    ws = Workspace.create(name=workspace_name,
                          subscription_id=
                          subscription_id,
                          resource_group=resource_
                          group,
                          location=workspace_region,
                          create_resource_group=True,
                          exist_ok=True)
    ws.get_details()

    # write the details of the workspace to a
      configuration file to the notebook library
    ws.write_config()
```

The code shown above tries to connect to an existing workspace based on the set environment variables. If the workspace doesn't exist, it will create it. In both cases, the workspace configuration is written to a file so it can easily be reused later. The SDK uses this file to connect to the workspace and is used in all the environments in the same way. You can include this code in a Python script or run it from a Jupyter Notebook.

Managing compute

Preparing data and training a model takes a lot of compute power. The power of the cloud is available in various options (e.g. DataBricks or Azure ML Services compute). That enables you to skip doing the heavy calculations on your development machine and utilize a server in the cloud. This compute power is available on demand, so you can easily do periodical reevaluations when the dataset has changed as new data comes in. Managing your compute targets through Azure ML service allows your team to share (sometimes costly) compute targets, which of course saves costs. It also makes it simple

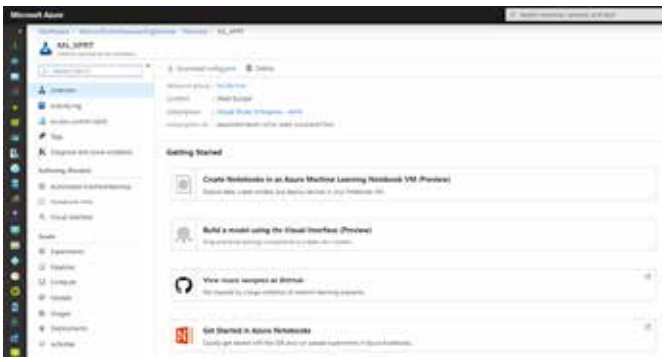


Figure 3: An Azure ML service workspace in the Azure portal

to execute computationally highly intense jobs on large and complex clusters of machines. It also enables you to scale the compute in and out based on your usage. You can even scale the compute down to zero if you are not using anything! The scaling is calculated on a metric like the percentage of CPU or RAM used over a period of time. This is used for scaling up as well as for scaling down.

Before you can execute jobs on a compute target, you'll need to define it in your code. The following code segment shows how this is done in Python, using the Azure ML SDK:

```
# Create compute
cpu_cluster_name = "cpu-cluster"

# Verify that the cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws,
    name=cpu_cluster_name)
    print("Found existing cpu-cluster")
except ComputeTargetException:
    print("Creating new cpu-cluster")

    # Specify the configuration for a new Azure Machi-
      ne Learning Cluster
    compute_config = AmlCompute.provisioning_
    configuration(vm_size="STANDARD_D2_V2",
                                        min_nodes=0,
                                        max_nodes=4)

    # Create the AML cluster with the specified name
      and configuration
    cpu_cluster = ComputeTarget.create(ws, cpu_
    cluster_name, compute_config)

    # Wait for the cluster to complete, show the
      output log
    cpu_cluster.wait_for_completion(show_
    output=True)
```

As mentioned before, you can connect many types of compute targets to Azure ML service[2]. The code as shown above connects to an existing "Azure Machine Learning Cluster" (which is the simplest form of compute to create) in the ML Service workspace, or it creates one if it doesn't exist. When creating an AML cluster, you specify the VM size and the minimum and maximum number of nodes. A minimum of zero means the cluster will be shut down after some time of inactivity, which again helps in reducing cost. It is common practice to start with a maximum of four nodes and run the experiments to get a feeling for the problem you are trying to solve, and whether or not it will benefit from more compute power.

Data preparation & model training

When you have your IDE and compute targets set up, you can start building your model. This starts with preparing data and then training the model. This is an iterative process, which can easily take many cycles before getting a model that meets your requirements. Azure ML services provides a place for automating this process, which is important for repeatability and traceability. This automated process is captured in a ML pipeline, which consists of multiple steps. For example, to create a step that executes a Python script on the cluster we just created:

```
# Specify the directory that contains the Python code
for this step
source_directory = './train'
print('Source directory for the step is {}.'.format(
    os.path.realpath(source_directory)))

# Specify the script to execute from the source_
directory
# and the compute target for this step (the cluster
we just created in this case)
step1 = PythonScriptStep(name="train_step",
                         script_name="train.py",
                         compute_target=cpu_cluster,
                         source_directory=source_
                         directory,
                         allow_reuse=True)
```

By specifying a unique source directory for each step, Azure ML pipelines will cache the result of the step, so it can be re-used in subsequent pipeline runs if the files in the source directory haven't changed. In a similar fashion, you can define pipeline steps to be executed on, for example Azure Data-Bricks or Azure Batch. Once all steps are defined, you combine them into a ML pipeline:

```
# list of steps to run
steps = [step1, step2, step3]
print("Step lists created")

# Build the pipeline. All steps will be executed in
parallel
pipeline1 = Pipeline(workspace=ws, steps=steps)
print("Pipeline is built")

# Submit the pipeline to be executed
pipeline_run1 = Experiment(ws, 'Hello_World1').
submit(
    pipeline1, regenerate_outputs=False)
print("Pipeline is submitted for execution")

pipeline1.publish(name='Hello_World1 pipeline',
                  description='My very cool pipeline')
```

In this case, all steps will be executed in parallel, since they are independent. If you explicitly specify inputs and outputs of steps, the Azure ML pipeline will calculate dependencies and execute steps in the appropriate order.

By publishing the pipeline, it becomes available for your team to view, edit, and re-run. You can trigger the pipeline from the Azure Portal, but you can also do this programmatically through a simple REST request:

```
# Retrieve an AAD token to authenticate your REST
request
auth = InteractiveLoginAuthentication()
aad_token = auth.get_authentication_header()

# Get the rest endpoint from the pipeline. You can also
get this from the Azure portal
all_pub_pipelines = PublishedPipeline.list(ws)
pipeline_to_start = all_pub_pipelines[0]
rest_endpoint1 = pipeline_to_start.endpoint
```

[2] https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-compute-target#train

```
print("You can perform HTTP POST on URL {} to trigger
this pipeline".format(rest_endpoint1))

# Start the pipeline. You can optionally specify para-
meters for the pipeline as a json body
response = requests.post(rest_endpoint1,
                         headers=aad_token,
                         json={"ExperimentName":
                         "My_Pipeline1",
                               "RunSource": "SDK",
                               "ParameterAssignments":
                               {"pipeline_arg": 45}})
run_id = response.json()["Id"]
print(run_id)
```

When tuning a model and its parameters, you typically execute your pipeline many times. The pipeline results are stored in an experiment, where each iteration of an experiment results in a "run". Azure ML services tracks these runs and the results in a central place. By logging the appropriate attributes from the ML pipeline, the Data Science team can collectively see graphical logged information, like model performance indicators and duration metrics. They can even see the version of the data the model was trained with, so they can take decisions based on that information.

To log a specific value, invoke the start_logging() method on your experiment and then use the log(...) method to log a metric:

```
exp = Experiment(workspace=ws, name='test_
experiment')
run = exp.start_logging()
run.log("test-val", 10)
```

By invoking the log() method during training you will get a visualization of that metric in your Azure ML workspace. In this example we will show the charts for two values we are logging:

Registering a model

When you are happy with your model results, it's time to register it. By registering your model, you assign a certain status to it. This can be anything from an Alpha/Beta labeling scheme to a version number. Published models can then (potentially) be deployed to production. The model to register is the output of an experiment run. Registering a model from a run is simple:

```
run.register_model('super cool model')
```

More advanced scenarios include automated evaluation of the model, and depending on the outcome of the evaluation, this can be registered. For example: only register the model if it performs better than our currently deployed model. The way to quantify "performs better", is up to you. It is then trivial to include this step in your ML pipeline, further automating your process.

Deployment using an Azure DevOps pipeline

When you have a versioned model that performs at the required level, it is time to deploy it. A deployed model enables an application to actually use the model to make predictions with. There are a lot of options to deploy a model, like hosting it in a webservice or a Docker container that can be deployed anywhere you want. That choice depends on the application that will use the model and the requirements you have.
The most common scenario will be to create a Docker container and deploy, for example in Azure Kubernetes Service (AKS) or Azure Container Instance(ACI), or even in an Azure App Service!

In this scenario you'll need a scoring script that is executed against the model. The scoring script accepts inputs, feeds them to the actual model, and presents the output back to the user. An example of a scoring script is a simple webserver that accepts a REST request with some input and sends the model response back in JSON format. The scoring script is packaged in a Docker container, together with the actual model.
There are various ways to deploy the model from Azure ML Services, but the most flexible method is to use Azure DevOps. This also opens up options for integrating with other parts of your application and their deployment pipelines.

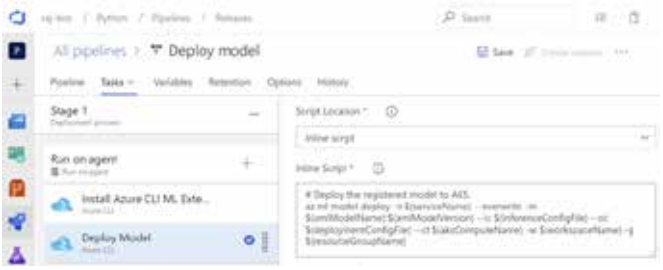
It also has additional benefits in terms of monitoring model performance, as we'll see later.

Deploying your models with Azure DevOps can be done in multiple ways, depending on the preferences of the Data Science team:

› From a Python script with the Azure ML Service and include it in you Azure Pipeline.

```
from azureml.core.model import Model
model = Model.register(workspace=ws, model_path=
"model.pkl", model_name="model-test")
```

› Using the Azure ML extension in the Azure CLI[3].



[3] https://aka.ms/aml-cli

This also registers the deployment in Azure ML service and annotates it with the version number and other metadata to provide end-to-end traceability. Especially in an Enterprise environment it is important to have logging on what was changed and by whom.

Machine Learning extension in Azure DevOps

A very helpful Azure DevOps extension is the "Machine Learning" extension[4]. This enables you to trigger a release whenever a new model is registered. The new model is then an artefact to trigger your release on. It also provides extra tasks to deploy a model to Azure ML services and to set the optimal values for CPU and memory options for the Docker container to use, and which are retrieved from the metadata in Azure ML services.

Telemetry

In any DevOps culture, monitoring the running software is a key ingredient to a healthy process. This is also true for Data Science: once your model is deployed, you want to know how it performs. Without any monitoring you do not know anything about the data that the model gets and generates, or when the model needs tuning due to data drift. You might even be influencing some results with the predictions from the model! There are two levels of performance that are interesting: model performance and technical data.

Model performance

Monitoring model performance over time means you need to gather data on accuracy and data drift. Data drift occurs when production inputs turn out to have different characteristics than the data used to train the model. In order to analyze model accuracy and the amount of data drift occurring, it is necessary to collect model input and outputs and analyzing them.

If you have your model deployed in AKS, it's very easy to gather the required data by just including a few lines of Python in your scoring script[5]. Collected data is stored in blob storage. From there you can retrieve it, and analyze it through e.g. DataBricks or PowerBI.

Rob Bos

Kees Verhaar

"In any DevOps culture, monitoring the running software is a key ingredient to a healthy process. This is also true for Data Science."

```
from azureml.monitoring import ModelDataCollector

global inputs_dc, prediction_dc
inputs_dc = ModelDataCollector("best_model",
identifier="inputs", feature_names=["feat1", "feat2",
"feat3", "feat4", "feat5", "feat6"])
prediction_dc = ModelDataCollector("best_model", iden-
tifier="predictions", feature_names=["prediction1",
"prediction2"])

data = np.array(data)
result = model.predict(data)
inputs_dc.collect(data) #this call is saving our
input data into Azure Blob
prediction_dc.collect(result) #this call is saving our
prediction results into Azure Blob
```

Technical data

Just like any other application or service, it is vital to make sure your model is always up and running like it should. For this you'll need to gather performance data, such as response times, any exceptions that occurred, etc. In Azure ML, this is implemented through Application Insights. When deploying to AKS from Azure ML, technical data collection is enabled by setting the appropriate parameter in your deployment configuration:

```
aks_config = AksWebservice.deploy_configuration
(collect_model_data=True, enable_app_
insights=True)
```

Data is then automatically gathered in Application Insights, from which you can analyze it, configure alerts, etc.

Conclusion

The best options for your Data Science process depend heavily on what you're trying to achieve and on the skills of the team. We've shown a very simple setup to show the basics, and much more is possible. We've also shown how flexible the tooling has become to set up an Azure DevOps Pipeline using Python, used by most Data Science teams. </>

[4] https://marketplace.visualstudio.com/items?itemName=ms-air-aiagility.vss-services-azureml
[5] https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-enable-data-collection#enable-data-collection

Observability: Closing the DevOps loop

When we coin the term observability people often think "ah, you are hype-wording what we already do, but we call it monitoring". In the DORA 2018 State of DevOps report[1] this is being reported as respondents not seeing a difference between Monitoring and Observability. That is unfortunate, and we believe there are big differences. Let us start with comparing the definitions.

Authors Geert van der Cruijsen & Jasper Gilhuis

MONITORING[2] is tooling or a technical solution that allows teams to watch and understand the state of their systems. Monitoring is based on gathering a predefined set of metrics or logs.

OBSERVABILITY[3] is a measure of how well internal states of a system can be inferred from knowledge of its external outputs".

Looking at both definitions there is a fundamental difference between them. Freely stated, that difference comes down to this. Monitoring and dash-boards are often used to get alerts in issues that have already occurred in the past. Think of a disk-full message. It is great to get a message but the issue itself is not resolved. We can categorize this as "Known-Unknowns".

When we think of Observability, we should think of complex software systems that sometimes fall over. Think of a distributed calculation across multiple services that may provide a result. But is the perceived result a result that is a combination of all required calculations? We can classify this as "Unknown-Unknowns". To be able to answer these kinds of questions, our complex systems need to gather information that is queryable so we can determine whether calculations are fine. Gathering insights in this way is very valuable.

If we are in the middle of a crisis, it is likely that multiple issues are happening at the same time. The root causes of a failure often consist of multiple chained events. Having extensive telemetry available about your application(s) and its architectural landscape allows you to define metrics that will help you determine the issues at hand. We need smarter decision making.

Why aren't our current monitoring solutions sufficient?

The way we build software and infra-structure is changing rapidly. Trends like Cloud, Containers, Microservice architectures, and serverless (Functions as a Service) are fundamentally changing the way we build and operate applications. Almost all modern applications are far more distributed than their predecessors, and this requires a different approach from an operations perspective.

Traditional monitoring solutions are built for an era in which applications were less distributed and fewer things could go wrong inside the application itself. In the current distributed era, in which components call each other over the network instead of residing on the same machine, chances of failures are a lot higher. This is a paradigm shift we must consider when building our application by adding retry mechanisms and other forms of resilience to our applications. A failing service call does not mean our application doesn't work anymore. It is something that we know will happen because of the way we shape our application.

Because of this we must make a shift from measuring technical failures only to start measuring the impact these failures have on our end-users. It does not mean that any of these technical measurements are useless, they still provide valuable information, but we should not create alerts or dashboards based on these measurements.

[1] DORA 2018 State of DevOps report https://devops-research.com/2018/08/announcing-accelerate-state-of-devops-2018/
[2] Definition of Monitoring taken from the DORA Report
[3] Definition of Observability from Control Theory (https://en.wikipedia.org/wiki/Observability)

Observability vs Monitoring

Traditional monitoring is often done by creating several dashboards that show the current health of an application. Experience tells us that this is no longer sufficient. Have you ever had a customer calling you that something did not work, and when you took a look at your dashboards, everything was still green?

Observability takes a different approach compared to monitoring. Observability is about instrumenting your code to be able to inspect what goes wrong without having to change the application itself. The term Observability comes from control theory. The definition fits modern software development methodologies in which DevOps teams build and run the applications instead of having a team that is dedicated to running and monitoring applications. Because a single team is responsible, they have far more insight in what an application does internally and what could go wrong. They are also motivated to add instrumentation because they will be the team that gets called out of bed when something goes wrong.

The three pillars of observability

Observability is gaining a lot of traction lately and there are many vendors and open source projects jumping into the gap of building solutions to help with observability. In this article we do not want to focus on specific tools because we believe there is no single best solution for all companies, teams and application architectures.

What almost all these tools have in common is that they define observability based on three pillars: logging, metrics and distributed tracing. Depending on whether the tool can cover all of these areas or only one or two of them, the vendor will draw different diagrams of how these pillars relate. There are tools that cover all three areas, and that will tell you all pillars are closely related. And there are tools that focus on only metrics. Combining logging and distributed tracing will tell you these pillars are so different that you should approach them separately with specific tools. As always, the truth is: it depends. There are several factors that influence this: the complexity of your application and your organization, or the number of messages or daily users of your application.

Figure 1 Pillars of observability

Each of the pillars has a specific usage and the combination of them gives you the observability to be able to query applications when things go south.

Logging

Logging is something almost every developer is familiar with. Logs capture events happening in your application and store them so you can query them to get insights. A downside of plain logs is that they are really hard to search, especially in a distributed system in which a complete log of what happened to a user is divided over multiple services. A solution is to use a centralized logging solution. Popular choices when building an application on Azure are the ELK stack or Application insights.

Normal logging:

```
Log.Information("Request by userA
took 35ms");
```

Structured logging (serilog)

```
Log.Information("Request by {User}
took {Duration}", user, duration);
```

In addition to centralized logging, you will also need a way to store your logs in such way that you can actually search your logs easier than searching through plain text files. Structured logging is a way to turn your plain log files into queryable log files. Serilog and log4net are two libraries that can help you create structured logs in the .Net ecosystem, and there are libraries available for most development languages. Most centralized logging systems are made to store and query structured logging and support logfiles created by these libraries.

As you can see in the sample, the method used in structured logging is not that much harder than normal logging. However, the benefit is that the User and Duration object are now queryable in central logging tools, which means that it is a lot easier to see all logs for a certain user for example.

Most of the infrastructure components also have built in logging like web servers, databases, firewalls etc. It can be valuable to also send these logs to your centralized logging system to get a complete picture of what happened in your application.

The main downside of logging is that it can become expensive quickly. It's easy to add logs to your application but what happens when you get thousands or millions of users a day. How long do you store logs, and do you store all of them? Again, this question can only be answered with "it depends".

First of all, we have to make a distinction between operational logging and application logging. Operational logging is logging that is used to track whether an application is working as intended while application logging can be used to see what the application did functionally. Application logging provides data that you always want to store for longer periods of time and you want to store all of it. It does not even have to be in your log system. It could be stored together with the business data itself, such as audit trails or logs of all the statuses an order had.

Don't mix these types of logging with operational logs which help you track down bugs or problems in your application. People tend to store these operational logs for long periods of time as well, but how useful is that? How much information do they provide when you regularly deploy new versions that might have completely different code paths? In general, these logs should not be stored for long periods.

However, even if you store logs for about a month or less, this could be quite expensive to store if you have large amounts of users in a complex application. Because of that, most logging systems have an option to sample the data. This means that only a percentage of the actual logs is stored and the rest is thrown away. This has some downsides to it because you cannot trace back all the issues you had because you do not have all the logs. On the other hand, most logging tools have the option to do dynamic sampling which does not randomly remove logs, but especially keeps all the special events that behave differently from all other operations.

Metrics

Metrics are a way to store aggregated measurements as time series data. This has a large advantage as it is a lot cheaper to store the data, because you aggregate the events together per interval. Also, the growth of the storage is constant because it writes records at fixed intervals.

The way we use metrics is different from logs because we lose some specifics when storing metrics instead of logging all the events separately. For example, you could store the average request duration in a metric and you will only have the average number. When you log the duration of each request you can get more accurate numbers like maximums, minimums and calculate the average as well. Does this mean you should log instead of using metrics? No! Both have their strengths and they should be used in combination. Pick the right solution for the information you want to store.

Averages are often named as examples when talking about metrics. However, averages can hide important indicators when used in large amounts of data. If you track the average request duration and 90% of the requests are taking 0.01 second and 10% takes 2 seconds the average request duration is 0.2 seconds. When you only see a 0.2 seconds average you might think everything is quite okay but those users belonging to the 10% that take 2 seconds per call won't agree with you. Instead of averages you could therefore also look into percentiles.

Distributed Tracing

The last pillar of observability is distributed tracing. Distributed tracing can give you insights in how the flow of your application was. So, whereas logging or metrics could help you measure how long certain requests took, distributed tracing can help you investigate WHY a request needed that response time and which components were used in the flow of this request. This kind of information is useful especially in distributed systems.

Distributed tracing tools provide insights that might look quite familiar when you use the network tab in developer tool-bars, that is included in most browsers.

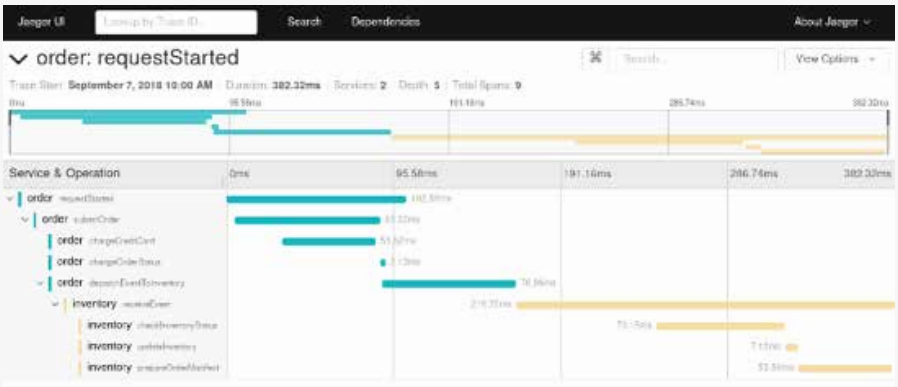


Figure 2 Example of an open source tracing tool (Jaeger)

Since tracing can span multiple systems, components or services, it's important that it is possible to write tracing information. Because of this the Open Tracing project was started to create a vendor neutral API for distributed tracing. OpenCensus is another project aiming for the same thing and the good thing is that these two projects are merging together (opentracing.io & opencensus.io). Asp.Net Core also supports this library from .Net Core 3 onwards out of the box, so services created with this will automatically work together with tools like Jaeger or Zipkin.

What to measure

So now that we know how to measure things using the three pillars of observability, what are the things we should measure? An easy way to remember this is to "USE RED". This is an abbreviation that stands for: USE (Utilization, Saturation & Error Rate) which we measure at a resource scope, and RED (Rate, Errors & Duration) which we measure on a request-based scope.

Resource scope

Measuring things at a resource scope can be valuable to track down performance problems in a system. What do Utilization, Saturation and Error Rate mean?

- Utilization: How much time was this resource actually busy responding to requests?
- Saturation: Is the resource able to handle all requests or is work waiting to be picked up / queued?
- Error Rate: How many errors does this resource produce?

Request-based scope

Measuring on a request-based scope can give you insight in how certain functions are performing. We do this using Rate, Errors & Duration

- Rate: Number of requests per second.
- Errors: Request error rate, what is the percentage of failing requests?
- Duration: Response time, Latency. How long did it take to handle the requests?

Geert van der Cruijsen & Jasper Gilhuis

How observability can change the way you build systems

A very well-known picture in the industry is the infinite DevOps loop. Many occurrences and spin-offs exist. In most of them the term monitoring is present. Now that we understand the differences and gaps, we believe the term monitoring in that loop should also imply implementing observability patterns and practices.

Changes are often made to systems without clearly knowing whether the result will be better. Or if there are no unintended side effects. You want to make decisions that are proven to be working by observability. An often-heard answer is that it is too costly to change your application. But there are some good patterns that you can apply that will help you make the right decisions.

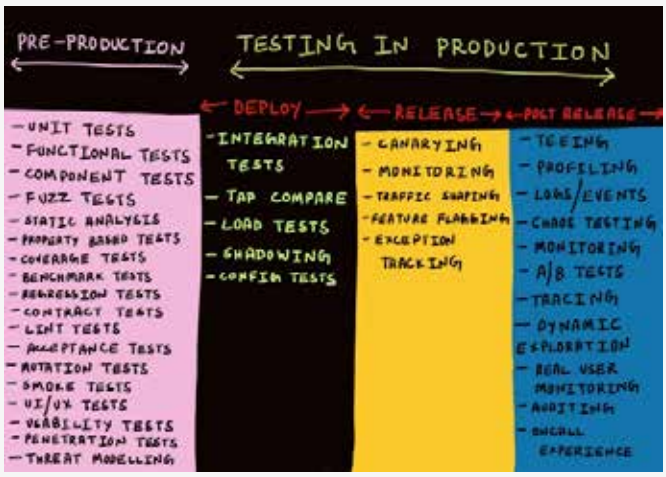


Figure 3 Testing in Production[4]

The above picture[4] shows us a very good overview of patterns and practices that can help you implement an observable system. Building an observable system starts at the beginning of the DevOps loop.

There are numerous types of testing that are aimed at the coding and testing phase, for instance unit tests, static code analysis, mutation tests, UI/UX tests and so on. They can be applied to the application in order to gain confidence in whether is being built is the right thing! If your system is lacking these practices, it becomes nearly impossible to obtain confidence in releases of that software. Just having these kinds of test is not enough.

During the deployment and releasing phase there are many practices that can be applied. Applying Canary releases and using Feature flags to safely release your software without impacting users allow you to release more confidently and use traffic routing to get a percentage of your users hit the new system after switching a feature toggle. Monitoring and observability then become key to be able to determine if your system is in a correct state.

For feedback (i.e. bugs, issues, new features) to transition smoothly there is a need to apply Site Reliability Engineering (SRE) practices to the DevOps loop. Which means that your engineers (Developers, Operations and your on-call support people and all others) should be aligned on how to deal with that feedback.

Conclusion

Many aspects of observability are of a technical nature, so making it visible for the organization and your customers is a challenge. We should apply all these practices and come up with tools and visualizations that show that we are in control of the system and will be able to meet our predefined goals. Many of the professionals and subject matter experts use best of breed tooling to fit their needs. This also implies that there is no single tool to rule them all. In a true DevOps nature we do not want to enforce these choices but we need consolidation on it. Ultimately it is all about what the end-user experiences. </>

[4] Image from blogpost by Cindy Sridharan, Testing in Production https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

“May the Source be with you!”

Jesse Houwing & Sofie Wisse

99% of code isn't yours

99% of your apps and sites are not your code. Your own 1% is under source control, but are you keeping taps on all of the libraries you import each time you do a **`dotnet restore`** or **`npm install`**?

Authors Jesse Houwing & Sofie Wisse

Over the last years there has been an increase in reported supply chain attacks. Attacks where the attacker isn't trying to get access to your source control repositories, but that of one of the many projects you depend on.
A bitcoin wallet was compromised and sent wallet keys to a third-party domain[1] through a nodejs package that changed ownership. Credit card details for thousands of users were intercepted through the chat client embedded in the same pages that handled transactions[2]. And it's not limited to websites and JavaScript apps. Asus had their laptop update tools compromised, causing specific targets to download, and install additional packages as part of driver updates[3].

The same dangers lurk for .NET developers. You may be asking: "how does it work, and how does it affect me?"

A supply chain attacks occurs when someone infiltrates your systems via a third-party service or dependency to exploit a vulnerability in a system. Typically, attackers try to insert malicious code into official downloads and installers of trusted third-party service providers or in dependencies used by developers. Once organizations start using these services, they are automatically exposed to the embedded malware too. Usually, the attackers are after access to source code or sensitive data, and they can access that by finding the weakest link in the software supply chain without ever having to go near their target's servers. One of the advantages for the attackers is that with one piece of malicious code in a dependency, they can target many organizations at once. On top of that it is often difficult for organizations to detect these attacks, since they depend on many third-party services and dependencies.

That is all interesting, but that won't happen to you, right? Well, as it turns out, it might not be as difficult for hackers to insert some malicious code into your project as you think. Here's a small scenario: imagine you are a .NET developer within an organization, and your team is responsible for an application handling sensitive information. You want to focus on the business logic of your application instead of reinventing the wheel for every bit of code you need, so you use NuGet as a package manager. It helps you re-use code from other developers to solve some of your tasks, that way you can spend your time on your application's specific logic.

While this is a common practice, using somebody else's code means that you need to find a way to trust it. Do you always know what is in the packages you consume? What if one of the many dependencies you use in your project is infected with malicious code? What would be the consequences? And how would you detect this at all?

How can this happen?

It isn't hard to be presented a different package when restoring packages across machines. This is the default behavior for most package managers, including NuGet. When you restore packages, it will try to find the versions you're after and will do a best effort attempt to resolve issues[4].

```
## Warning NU1603: Microsoft.
IdentityModel.Clients.Active-
Directory 3.13.5 depends on System.
Net.Http (≥ 4.0.1) but System.Net.
Http 4.0.1 was not found. An
approximate best match of System.
Net.Http 4.1.0 was resolved.
```

An example from one of the open source projects we maintain

[1] https://github.com/bitpay/copay/issues/9346
[2] https://security.ticketmaster.ie/
[3] https://securelist.com/operation-shadowhammer/89992/
[4] https://docs.microsoft.com/en-us/nuget/concepts/dependency-resolution

There are a few cases in which NuGet may not be able to get the same package graph with every restore across machines. Most of these cases happen when consumers or repositories do not follow NuGet best practices[5]:

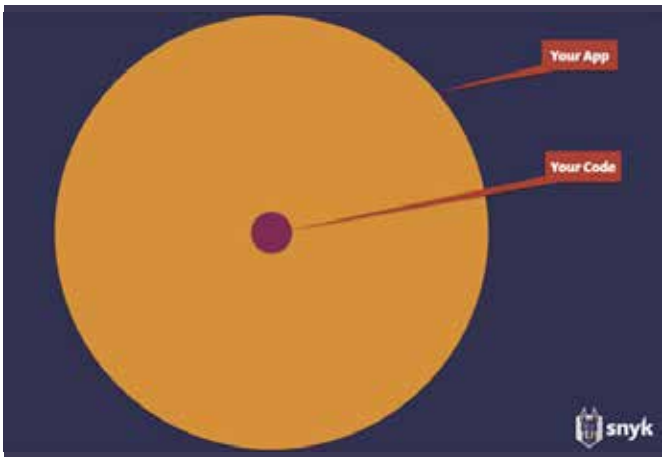
1. **nuget.config mismatch:** This may lead to an inconsistent set of package repositories (or sources) across restores. Based on the packages' version availability on these repositories, NuGet may end up resolving to different versions of the packages upon restore.
2. **Intermediate versions:** A missing version of the package, matching PackageReference version requirements, is published:
   - Day 1: If you specified `<PackageReference Include="My.Sample.Lib" Version="4.0.0" />` but the versions available on the NuGet repositories were 4.1.0, 4.2.0 and 4.3.0, NuGet resolves to 4.1.0 because it is the nearest minimum version.
   - Day 2: Version 4.0.0 gets published. NuGet now restores version 4.0.0 because it is an exact match.
3. **Package deletion:** Though nuget.org does not allow package deletions, not all package repositories have this constraint. Deletion of a package version results in NuGet finding the best match when it cannot resolve to the deleted version.
4. **Floating versions:** When you use floating versions like `<PackageReference Include="My.Sample.Lib" Version="4.*" />`, you might get different versions after new versions are available. While the intention here is to float to the latest version on every restore of packages, there are scenarios where users require the graph to be locked to a certain latest version and float to a later version, if available, only upon an explicit gesture.
5. **Package content mismatch:** If the same package (id and version) is present with different content across repositories, then NuGet cannot ensure the same package (with the same content hash) gets resolved every time. It also does not warn or error out in these cases.
6. **Cache poisoning:** NuGet will check the local package cache before checking configured package feeds (unless `--no-cache` is specified). These will be used in case of an exact version match. If you are using a proxy feed (such as Azure Artefacts), an attacker with access to the feed (or an upstream feed) could publish a specific version to that feed which will be used instead of the one you are expecting.

More and more re-use

If we would only depend on a few dependencies and if they would only change once in a very long while, it wouldn't be hard to manually review the changes. If you had access to the sources. And in that case, you could copy all your dependencies to a manually curated feed.

But we don't live in that world anymore.

As Snyk.io's Liran Tal[6] puts it: vulnerabilities can occur anywhere, but you only have full control over a small piece.



When you create a new Visual Studio 2019 (16.2.2) React.js Web Application project, you end up with 15214 Nodejs packages (686 with known security issues) and 284 NuGet packages (18 with known security issues). If any of them is compromised, you may be adding them to your project the next time you run `npm install` or `dotnet restore`.

Or worse, your local development machine may be fine, but the build server may be fetching all the latest versions. This is especially the case when you use the Azure Pipelines Hosted Pool, since every build uses a fresh image with very few packages pre-cached.What we need is a way to store all our dependent packages in source control in an efficient manner, preferably without having to store all the contents of the packages in source control. Now, while that may sound like a contradiction, it isn't. Instead of storing all package contents and that of all their dependencies, use what npm, NuGet and yarn do. These tools all store the name, exact version, and a hash of the package contents for all packages in the dependency tree in a file. This file is called a lock file, and by committing this lock file to your version control repository, you ensure that:

1. Your build server (and your colleagues) will use exactly the same packages you used on your development machine.
2. You keep an auditable log of all the changes to your dependency tree.
3. You can inspect all changes to the dependencies prior to committing, or as part of the pull-request review process.

Generate lock files for .NET solutions

Your .NET projects won't generate lock files by default. You must also upgrade your project to use the new <PackageReference> format[7]. Then you can instruct the build process to generate the lock file through a command line parameter:

Generate the lock file through dotnet:

```
> dotnet restore --use-lock-file
```

Generate the lock file through msbuild:

```
> msbuild /t:restore /p: RestorePackagesWithLock-
File=true
```

[5] https://devblogs.microsoft.com/nuget/Enable-repeatable-package-restores-using-a-lock-file/
[6] https://twitter.com/liran_tal/status/1067775376229834754
[7] https://natemcmaster.com/blog/2017/03/09/vs2015-to-vs2017-upgrade/

You can also add a Property to your project files to generate lock files on every restore:

```
<Project>
    <PropertyGroup>
        <RestorePackagesWithLockFile>true
        </RestorePackagesWithLockFile>
    </PropertyGroup>
</Project>
```

Note: This behavior is different from npm and yarn, which automatically generate the lock files each time you restore your dependencies.

NuGet will now store a packages.lock.json alongside every project. The file contains all the dependencies, their exact versions, how the dependency was introduced, and a hash of the package contents:

```
    "Microsoft.AspNetCore.WebSockets": {
        "type": "Direct",
        "requested": "[2.2.1, )",
        "resolved": "2.2.1",
        "contentHash": "Ilk4fQ0×dVpJk1a+
        72thHv2LglUZPWL+vECOG3mw+gOesNx0/
        p56HNJXZw8k1pj8ff1cVHn8KtfvyRZxdplNQA=",
        "dependencies": {
          "Microsoft.AspNetCore.Http.Extensions":
          "2.2.0",
          "Microsoft.Extensions.Logging.Abstractions":
          "2.2.0",
          "Microsoft.Extensions.Options": "2.2.0",
          "System.Net.WebSockets.WebSocketProtocol":
          "4.5.3"
        }
      },
```

Commit these files to your source control repository to store the exact dependencies along your other source files.

Restore from the lock file in your CI solution

What we want NuGet to do, is to download the exact same packages we used on our development system. Just storing your dependencies in source control isn't enough. One of the first steps of your CI process is likely `dotnet restore` and unless we do something about it, this will just download a new set of dependencies and then overwrite the lock file.

Instead, we should tell NuGet to restore the exact packages specified in the lock file. And again, this can be done through a command line parameter or an msbuild property.

To restore in locked mode using dotnet:

```
> dotnet restore --locked-mode
```

Restore in locked mode using msbuild:

```
> msbuild /t:restore /p:RestoreLockedMode=true
```

To ensure the Continuous Integration server uses locked mode by default, you can also set this property in the project file:

```
<Project>
    <PropertyGroup>
        <RestorePackagesWithLockFile>true
        </RestorePackagesWithLockFile>
        <RestoreLockedMode
            condition="'$(RestoreLockedMode)' == ''
                && ('$(TF_BUILD)' == 'true'
                || '$(CONTINUOUS_INTEGRATION)' ==
                'true')"
        >
            true
        </RestoreLockedMode>
    </PropertyGroup>
</Project>
```

You're all set, your .NET projects will now restore to a predictable set of dependencies each time you build it, or the build will fail.

Each time you restore locally, you'll see exactly which packages have been updated and you can inspect their contents on your development machine:

Restoring against a different .NET Core version may cause different package contents with the same version. This will be detected and fails your build.

Impact on build times

You may be wondering what the impact on restore times will be when turning this feature on. On the development machine restores will take longer, because the lock file must be generated and the hash for the package contents must be calculated.

On the build server it's less clear-cut. The time to resolve package versions and calculate the dependency tree is reduced to the time it takes to just load the lock file. This may save a lot of time. On the other hand, verifying the package contents will add some time. In our tests, the average times to run the build on Azure Pipelines were faster with the locked mode turned on.

Hands-on: Try the Global DevOps Bootcamp 2019 challenge

The Global DevOps Bootcamp 2019 featured a Supply Chain Attack challenge[8] that lets you experience the effects of a supply chain attack. As part of the hands-on lab you get to generate npm and NuGet package lock files, adapt the build process to perform locked restores, and add a scanner to your build process to detect known vulnerabilities in your dependencies. By applying these techniques, you will be able to take control over what you ship to your customers every time you deploy your latest changes. </>

[8] https://www.gdbc-challenges.com/Challenges/ChallengeDetails/VULNPACKAGE

A recipe for high-quality releases

Shipping applications to a production site, continuously, is becoming more common every day. Deployment pipelines make automatic installation of software onto a site possible. The next step? Releasing value increments continuously and safely.

Author Albert Starreveld

Quality assurance is an important part of continuous delivery. Installing the software on production is one thing. Making sure it does what it is supposed to do, is another. We prefer validating that automatically, within a couple of minutes. It is possible to create a release pipeline that validates all functional requirements of the software, fast, and in a way that convinces non-IT stakeholders.

The deployment checklist

At some point in time, somebody decides to go live. The users of the application can now start using new features. Going live shouldn't cause any problems. Validating some of the following will ensure that:

- All the domain's business rules. Have they been implemented correctly?
- The integration of the software in the IT landscape. Do all the applications that depend on it, and all other applications it depends on, still work?
- Infrastructure. Does the application have sufficient permission to do what it is supposed to do? Are all SSL certificates still valid? Has the application been configured correctly?
- Non-functionals. Like security and performance.

Validating that can't take longer than a couple of minutes. And that's challenging. Especially because there are a lot of business rules to validate, and usually validating business rules requires integration. Too much integration makes it impossible to validate all business rules in less than five minutes.

> **"Unit tests are tests, written by developers, for developers, and they are fast."**
>
> Bob Martin

The target audience of a unit test

Depending on how the unit test has been written, it gathers a different type of information. It either manifests in a meaningless green bulb, or it provides information about what functionality has proven to be implemented in the software.

Unit tests test the if-statements of the code. Some developers write them before they write the code, others do so afterwards.

There are different styles of unit testing. Some developers say a "unit" is a single class. Others say a "unit" may as well be a combination of classes. The "smaller" the unit, the more technical information the tests provide. As a unit becomes "bigger", and the subject under test is a combination of more components, the unit tests tend to provide more functional information. These two schools of unit testing are known as inside-out and outside-in, or London and Chicago style.

Depending on the type of information the development team needs, a different style of unit testing applies. Unit tests only provide the developer with very detailed information about small parts of the system. They assist the development team in judging the fitness of low-level components of the system.

Unit tests are not enough

Considering the target audience of unit tests. The development team should have (some of) the information they require to support their decision to go live. Unfortunately, usually, the development team is only one of many stakeholders in a project. What information do other stakeholders need? And in what format?

Installing the software in a test environment and having stakeholders validate their requested changes there, slows down the deployment process.
Such a scenario indicates manual work, too. Manual work is expensive and time-consuming. Also, this scenario means regression needs to be tested manually. That slows down the deployment process even more!

A proper, manual regression test easily takes days or weeks. All functional requirements of the complete system are validated in such a test. Depending on the desired release frequency, automating that process can save lots of time and money.

Whether or not automating regression tests does save time and money, depends on the amount of effort spent to create the automated regression tests, the amount of maintenance they require over time, and how fast they run. Testability is a software-architectural driver. If releasability is a requirement, proper software architecture supports easy regression testing.

Separate business rules from infrastructure

A common misperception is that integration should be tested at the API level. Invoking a REST endpoint carries out a command to the database and back. This results in tedious, slow tests that require a lot of maintenance. They require databases to be in a given state, configuration to be correct, other systems to be up, and so forth. Such a test has too many responsibilities. It validates too many different things, implicitly and explicitly, and as a result, it is slow.

Invoking an endpoint to validate whether the business rules have been implemented correctly, proves more than intended. It proves a correctly configured API, for example. Otherwise, the endpoint would not have been available. And proving that the API has been configured correctly over and over again for every business rule is redundant and time-consuming.

The Single Responsibility Principle also applies to tests. It is the key to a fast test suite. Make a test responsible for validating only a single thing. Craft the source code accordingly. Creating a fast, functional test is particularly easy when infrastructural concerns and business rules haven't been mixed. Alistair Cockburn's Ports and adapters architecture (also known as Hexagonal architecture) demonstrates one of many ways to properly separate these concerns.

Separate technical test code from functional test cases

Continuous delivery and deployment pipelines are words managers or clients should not need to understand. They probably care about the (strategic) goals of their business and how the new versions of their software help them achieve those goals. Showing what the development team has changed in the software, and how they have mitigated any risks associated with that, can help to gain the stakeholders' confidence. It sounds like a lot of work, but that should be fairly easy with the proper integration tests in place.

Unit tests have proven to be tough to explain to clients. A common practice is to separate (important) test cases from test code. Use Behavior Driven Development (BDD) to do that. Write down the specifications of the software in the Gherkin format and discuss them with the stakeholders. Use BDD frameworks like SpecFlow to implement these test cases and run them in the deployment pipeline during every release. And use plugins like Pickles to generate reports about the team's test efforts, automatically and without any effort.

Take chain testing to the next level
Loads of unit and integration tests provide clarity on the quality of the software. When these tests pass, they indicate that the business rules have been implemented correctly. But there's still the matter of integrating them into an environment of other systems. Most likely the system depends on other systems and other systems depend on it.

Chain tests are extremely time-consuming and expensive. Opening a front-end, going through a couple of scenarios, and validating what appears in other systems, proves that systems integrate correctly. This requires the entire application landscape to be up and running, to be configured correctly, and to be in a given state.

Conceptually, when a user clicks a button, a command is carried out by a system. This system creates other commands and queries, in a given format, which it sends to other systems.

Any system can produce a variety of queries and commands to other systems. They're mocked and asserted upon in the tests. Sharing these mocks and assertions, allows other teams to use them as input for their tests. They can validate the ability to process them, continuously, without having to install any software in any environment. This concept, Consumer-Driven Contracts, allows delayed execution of chain tests in a deployment pipeline and locally. It provides fast feedback and reduces the need for chain testing.

Does it run at all?

It's good to know the business rules have been implemented correctly and that the application integrates into the environment. But that's rather useless if the application doesn't start.

Epic fail.. All tests pass...
But production is broken...

Regardless of all validations mentioned earlier, there's still a chance the application isn't going to work. Faulty configuration, permissions, and dependencies can cause the application to break.

That's easily validated by invoking the most important endpoints and by asserting it doesn't return any unexpected errors.

Unfortunately, invoking some endpoints will cause mutations in data. Everybody knows mutating production data with tests should be avoided. And running these tests in any environment but the production environment doesn't make sense either. It proves the test environments are fine, while we want to check whether the production environment is working properly.

Hence the need of a representational test environment. An environment that's pretty much equal to the production environment. Running these tests there makes the test-results conclusive enough.

Look for a sign of life

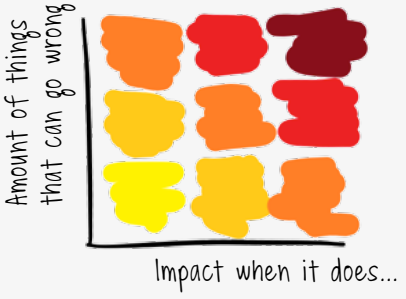
Having a production-like environment, in which all critical parts of the application seem to be operational, makes it likely that the deployment on the production environment will be successful.

There shouldn't be a big difference between a production-like environment and the production environment itself, but there must be some difference... After all, it's production-like, and not the production site itself.

Execute a simple smoke test after installing the software onto the production site. A trivial one, too. Invoke some GET on an endpoint and assert a 200 OK.

Choose carefully

Base the effort spent on testing on the risk that is involved. Not all changes are equally risky. Faults in the software have a different impact. In some cases, it makes a lot of sense to spend more money on testing. In some cases, it doesn't. It depends. Use a probability impact matrix to determine the effort that applies:



A probability-impact matrix helps to estimate the risk

Testing software takes time. And clients want their features at some point in time. Sometimes pushback is appropriate, and clients should wait just a little longer. Use the probability impact matrix to decide when to push back and when to take time for testing. Estimate the probability first: How likely is it that this change will cause issues? Then estimate the impact: How much money, reputation, or any other thing that matters to the company, is lost if it does? The more risk involved, the more testing effort is appropriate.

Summary

Martin Fowler's Practical Testing pyramid shows a different type of automated tests. The top level of the pyramid refers to assertions made on the entire application as a whole. That's the highest possible level of integration. The bottom of the pyramid refers to assertions made on parts of the system in the highest possible level of isolation: Unit tests.

Unit tests are tests written by developers, for developers. They provide developers the information they need to determine the health of the low-level components of the system.

Usually, the other stakeholders can't judge the health of the system by looking at the results of unit tests. They need functional, readable test cases to make that call. And they need to make assertions on combinations of components. Separate functional test cases from technical test code, and make the test cases available to the stakeholders. Isolate the combinations of components that implement business rules from infrastructure to keep the tests fast.

Nonetheless, configuration, permissions and external dependencies can still cause an application to break. Run contract tests, end-to-end tests, and smoke tests in the release pipeline to make sure that it doesn't happen.

Don't get carried away. Base the test effort, per story, on a probability impact matrix to make sure the effort spent is reasonable, compared to the risks involved. </>

"Testing is an information, intelligence or evidence gathering activity performed on behalf of stakeholders to support decision-making."

Paul Gerrard

Albert Starreveld

How API Thinking revolutionizes Healthcare

Have you visited a hospital recently? You probably noticed that the hospital didn't feel high tech. Nurses, doctors and supporting staff often look like data-entry engineers. The user interfaces they work with are old fashioned and their computers are slow. Let's face reality: our standard is the digital enterprise, while many hospitals are still living in the stone age of digitalization. Luckily there's a movement which rapidly changes hospitals: FHIR (pronounced: FIRE)!

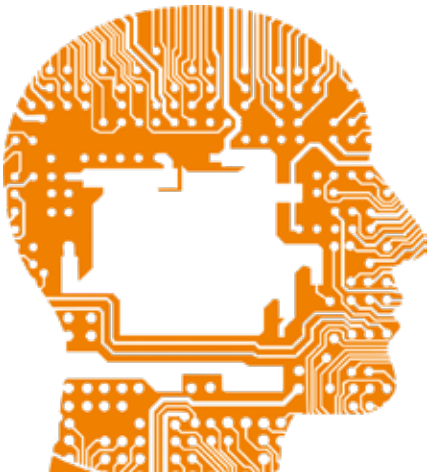
Author Alex de Groot

To understand where this hospital IT backlog comes from, you'll need to take a deeper look into the processes in healthcare. Treatment of a patient hardly ever originates from the hospital itself. Typically, it starts from a general practitioner (local doctor) who creates an initial observation. This observation leads to a referral to a more specialized practitioner. This could either be in a hospital or in a specialized clinic.
When this specialist practitioner tries to perform diagnosis, additional studies such as Imaging or Laboratory Studies are necessary most of the time.
The specialist can't perform this study himself, so, he'll create another referral. Once this study is completed, the specialist receives details about the study and determines a treatment plan. Part of this plan is a referral to a more specialized practitioner. And so the journey continues... Each disease and each human body is different, and thus deserves its own, tailor made treatment. All of the steps in these custom work-flows collect and transfer information.

It gets more complex when you realize that not all specialisms are working inside the same hospital, building or legal entity. Treatments are sometimes even stretched across country borders. Each specialized practitioner comes in with his own processes, tooling and jargon. In the IT industry we learned the Conway Law,[1] that communication structures are reflected in the systems you design. Hospital IT is no exception to this.

Every specialism has its own system, with its own technology and characteristics. Many of these specialisms aren't really part of the hospital; they are associated with the hospital as a separate legal entity, while isolating patient data inside their own systems.

During recent years, large regulatory bodies such as the E.U. have started to form opinions about data ownership. With the intention of protecting civilians against the influence large companies can have on their lives, new protection regulation has been written. Most people have heard about the GDPR, but don't understand the enormous impact on various industries. Inside the healthcare industry a patient already has the right to have a look at his dossier with a summary of the various treatments, but under the new regulations, everyone has obtained the right to change, correct, transfer or erase all information about himself, at no cost (within certain legal boundaries).



[1] https://en.wikipedia.org/wiki/Conway%27s_law

To simplify the handover in the workflows, hospitals have already started automating using Hospital Information Systems (HIS). These are large central systems collecting data about patients. Typically, a HIS contains central appointment management, patient dossiers and workflow registration. Since these systems are responsible for the overview, they only connect to more specialized systems via hyperlinks or application launchers.

When your treatment is inside a single hospital, the HIS typically does its job very well. But once we go beyond the hospital, for example for a second opinion, you immediately hit a wall. The need to share the entire (relevant) patient history, including all of its specialized studies, is a complex question. There's a clear need for a smart way of inter-operability between the two parties.

In the late 80s, the HL7 standard organization was founded. This organization has a clear mission statement: A world in which everyone can securely access and use the right health data when and where they need it. As a result, several standards were published. Unfortunately, the adoption of these standards is relatively slow. In recent years, as the business case started to increase, HL7 made a strategic move, and brought together a group of very active community members.[2]

FHIR stands for Fast Healthcare Interoperability Resources and is basically a domain definition for a REST API. It has abstracted all functionality in building blocks, which can evolve independently. The domain objects use ISO-standards for data formatting and can be queried in a traditional REST way or using GraphQL. The standard contains the latest generation security (OAuth), has an answer for object extensibility, and allows for searching.

Probably the most powerful feature of FHIR R4 is the clear way in which capabilities are structured and secured. The API is transparent about which pieces of the specification are implemented, and how they are made available. Any implementer of the standard only needs to expose the relevant parts. While a consumer can use the API to quickly find out whether a capability is available at all.

Now let's picture a use case in which an AI startup specializes in analyzing complex bone fractures. For a hospital, this can be an enormous cost saver as it enables quicker diagnoses. Using their FHIR endpoint, all they have to do is expose an Imaging Studies-capability to this startup. The startup can then read the patient scans (Imaging Studies) from the endpoint. Once the analysis is completed, an observation is POST-ed back to the hospital. This observation can lead to a formal diagnosis once a specialist in the hospital has approved it.

From an architectural perspective the FHIR standard also opens doors. By having a clear way of showing how systems should communicate with each other, the FHIR API decouples all these systems from each other. It allows for applying Sacrificial Architectural[3] principles on the hospital enterprise architecture. Think about migrating data from a legacy system to a new one, consolidate two systems into one system, or even seamlessly replace one system with another.

Several large cloud vendors - including Microsoft, Amazon and Google – are creating support in their platforms for FHIR.[4] Of course, healthcare is one of the largest industries in the world, but besides the financial advantage, they also see FHIR's contribution to world health. The support of these giants guarantees technical innovation in the long run.[5] [6]

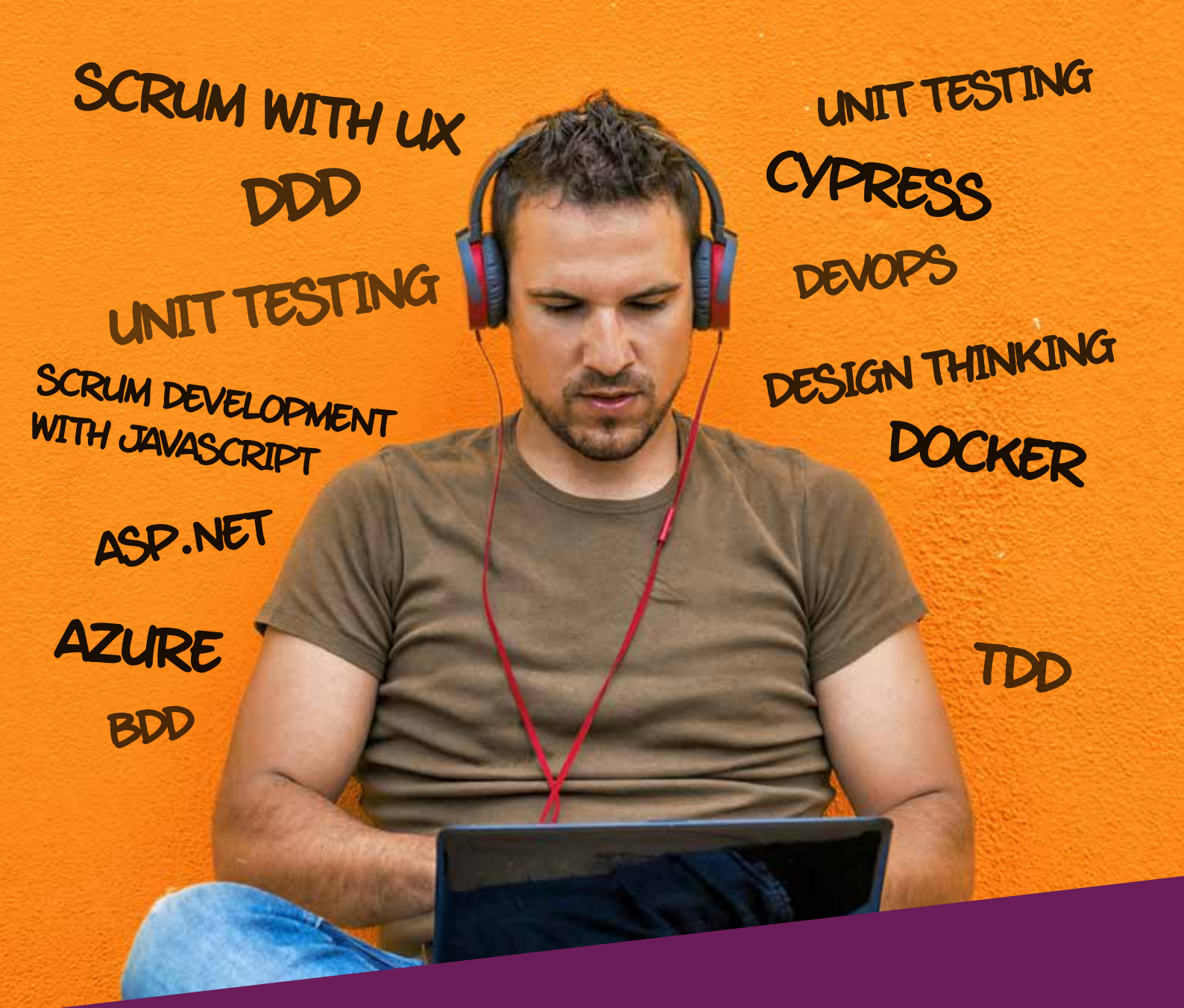
In addition to the adoption by the cloud vendors, you can see an active community creating several innovations and standards related to FHIR R4. Large, industry-wide hackathons such as the FHIR DevDays[7] are delivering great value and new, unforeseen insights. A great example of this is the development of SMART-on-FHIR, which allows applications to start in the context of a specific user. How cool would it be if the assistant can send the right context to the active mobile device the doctor is currently carrying?

Other industries can learn from the steps the HL7 FHIR team is taking. For example, the domain model wasn't built with the purpose of justifying existing systems. Instead, it was developed as part of the community process. The intention of this technology push is clear: it enables a larger audience to adopt FHIR quicker. Above all, the commitment of the community is unprecedented. The delivery FHIR R5 has already started and is expected to be ready in 2020 .

Back in 2013, Harvard Business Review published a clear vision on the future of healthcare. The widely recognized Harvard professor Porter stated that decentralized treatment and patient-centered data management (among a few others) could 'fix' healthcare. FHIR and its community are an enabler for both. With the support of the tech giants and an emerging need due to population growth, healthcare takes the next step. It might be slower than other industries, but eventually we'll all benefit from this technical innovation. </>

Alex de Groot

[2] https://www.hl7.org/fhir/
[3] https://martinfowler.com/bliki/SacrificialArchitecture.html
[4] https://www.geekwire.com/2019/microsoft-amazon-tech-giants-forge-ahead-healthcare-data-sharing-pledge/
[5] https://azure.microsoft.com/en-us/services/azure-api-for-fhir/
[6] https://github.com/microsoft/fhir-server/
[7] https://www.devdays.com/



Skill up for full cycle ownership

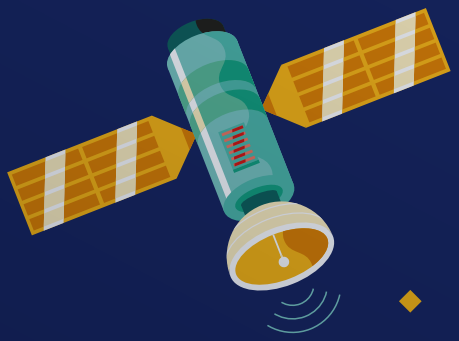
On your way to becoming a full cycle developer? There isn't just one route to full cycle ownership.

That's why Xpirit proudly joins Xebia Academy, so you can broaden your skill set from the best tools Microsoft has to offer to design, testing, deployment, and operations.

For every training you need
training.xebia.com

Xebia Academy

www.xpirit.com



Think ahead.
Act now.

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