Xpirit Magazine 3

Speed is the new currency!

Joseph Hill
Mobile Innovation
Don't wait for the perfect app

René van Osnabrugge
Continuous Delivery 3.0
The next 'next step'

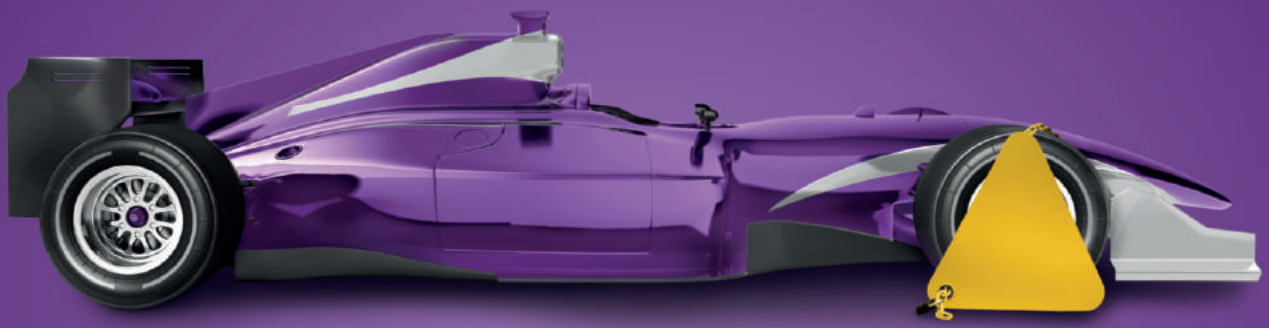
Alex de Groot
How to accelerate
your choices using data

Other articles are: ■ Release management, from technical to functional practice ■ Infrastructure as Code ■ Exploring the Microsoft Bot Framework ■ Continuous Delivery 3.0 for Mobile apps ■ Extending your Build and Release pipeline ■ Technical Debt in your Application Lifecycle ■ Doing Testing Right and at the Speed of Light ■ Containers on the Microsoft platform: the full picture ■ Conquer the world with Azure Machine Learning

Xpirit

Think ahead. Act now.

Microsoft



Don’t let an oversight put the brakes on your projects

Have you ever had that sinking feeling when you realize you should’ve checked something, but you took your eye off the ball and now you’re left paying the price?

We can’t settle your fines, but we can help your development teams avoid similar mistakes at work. Microsoft’s open, flexible and extensible DevOps tools keep your development and operations teams in perfect sync and at peak performance, increasing the quality and availability of applications and services – on any platform.

Microsoft DevOps Solution allows you to:

- Deploy updates faster and with less failures.
- Increase quality and availability of apps and services.
- Speed up provisioning and lower the cost for IT infrastructure and eliminate waste caused by over-provisioning, under-utilization of resources.

The Microsoft DevOps Solution

Don’t let anything get in the way

Edition:
Xpirit Netherlands BV
No. 3 • October 2016

Editorial Office:
Xpirit Netherlands BV

This magazine was made with the help of Pascal Greuter, René van Osnabrugge, Marcel de Vries, Pascal Naber, Peter Groenewegen, Joseph Hill, Chris van Sluijsveld, Geert van der Cruijssen, Jesse Houwing, Jasper Gilhuis, Erik Swets, Viktor Clerc, Alex Thissen, Loek Duys en Alex de Groot.

Contact:
Xpirit Netherlands BV
Utrechtseweg 49
1213 TL Hilversum
The Netherlands
Phone: +31 (0)35 538 19 21
E-mail: pgreuter@xpirit.com

Layout and Design:
Reclamebureau Bij Dageraad
Winterswijk
www.bijdageraad.nl

Copywriter:
TechText

Welcome to Xpirit magazine #3!

Time is passing at the speed of light, and the good news is: at Xpirit we're ahead of time. While our company is soon celebrating its second birthday, our team of professionals is boasting a collective experience of decades. Experience that is based on the interesting and challenging projects we're involved in, and that reaches out to the forefront of market developments. Experience that we'd like to share with you. In workshops, in team meetings, during events, and last but not least, in this magazine.

This third edition of our magazine once again shows our close collaboration with Microsoft and naturally you'll find a lot of information on the role of Microsoft's Visual Studio Team Services, Team Foundation Server, the MS Bot Framework, the use of Containers on the Azure platform, as well as Azure Machine Learning. Because time is a key factor in your success, we're paying special attention to how you can apply your tooling in implementing continuous delivery and testing. In fact, we've entered the age of Continuous Delivery 3.0.

As Microsoft's Joseph Hill explains in his article: you can't wait for the perfect app. There's thousands of companies out there that started out with apps that appeared to be marginal, and then evolved into huge success stories. And as for making your app perfect before launching it, there's an article on embedding testing activities in the development pipeline. Early and ongoing, continuous testing allows you to reduce time to market and outpace your competitors. At the same time you make sure that your app is free of bugs and faults that could lose you customers. For that matter, feature toggles can be a valuable aspect of testing, and that's why you'll find a number of articles covering feature toggles.

Speed is the new currency

The development pipeline is also covered in terms of Infrastructure as Code, the MS build and release engine, and the separation of deploy and release. This separation makes business more independent of the IT department when it comes to releasing functionality to end-users. Our focus is on the optimization of the development pipeline with the latest methods and tools, but we've also taken a brief look at how you can optimize your choices using big data, how you can avoid being held back by history, and how you can reduce technical debt in order to minimize risks and achieve greater results.

Please enjoy and benefit from this issue. If you prefer the digital version of this magazine, please visit our website at: http://xpirit.com/magazine or use the qr-code.

PASCAL GREUTER

MANAGING DIRECTOR
XPIRIT

Table of contents

Continuous Delivery 3.0: The next 'next step'

"Continuous Delivery is the logical evolution of Agile[1]" - this statement was written by Kurt Bittner in 2013 in a report of Forrester. Back in those days Continuous Delivery was not yet as far in the hype cycle as today. In the Application Development hype cycle that Gartner presented in 2015[2], Continuous Delivery was on the rise and nowadays it is in every year plan of any IT-related company.
But what exactly is Continuous Delivery, and what does Continuous Delivery 3.0 add to this concept and what does it have in common with Digital Disruption. In this article you will find answers to these questions and learn why it is important to look beyond the hype cycle.

Continuous Delivery

Continuous Delivery is the fulfillment of the Agile promise. With over 95% of companies practising Agile[3], it is a logical and required step for most companies to look further. Agile brought us the benefits of being able to produce high-quality software produced in a matter of weeks. But the next challenge, delivering the software and delivering business value to the customers, proved to be a whole different ball game.

This is where Continuous Delivery comes in. Continuous delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be released reliably at any time. Its objective is to build, test, and release software faster and more frequently.

Ideally this means that all functionality is planned, realized and released with the same repeatable process, called a pipeline. Quality gates (like unit tests, continuous builds, acceptance tests, and metrics) ensure that the functionality and quality is guaranteed. If something is not right, the pipeline is stopped so that the error can be fixed. Therefore, metrics should be in place to understand, trace and pinpoint the error to make sure this does not happen again. Every piece of software, whether it is a bug fix,

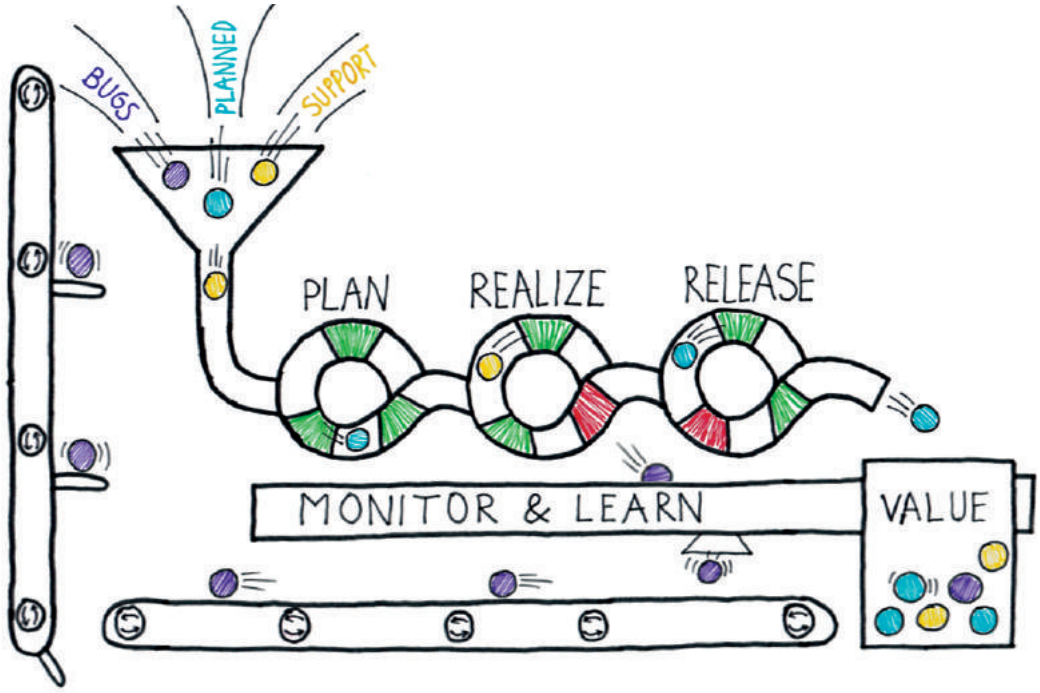


Figure 1

[1] Forrester: Continuous Delivery Is Reshaping The Future Of ALM, Kurt Bittner, July 22, 2013
[2] http://xpir.it/mag3-cd30-1
[3] http://xpir.it/mag3-cd30-2

a new feature or a change request, is treated in this same manner.
This sounds perfect for new projects and new software, but how does this work within existing scenarios and within organizations that have already been building software for many years? Before I dive in to this, I want to talk a bit about digital disruption.

Digital Disruption: rethinking existing scenarios
Digital Disruption has a strong relation with the Law of the handicap of a head start[4], which describes how making progress in a particular area often creates circumstances in which stimuli are lacking to strive for further progress. This results in the individual or group that started out ahead eventually being overtaken by others. In the terminology of the law of the handicap of the head start, what was initially an advantage subsequently becomes a handicap.

Do old constraints still apply?
The question that we need to ask ourselves is: "Why are experienced, rich and large companies not disrupting the market themselves?". While there is no generic answer that applies everywhere, one of the things that makes companies slow and rusty is a lack of self-reflection. Processes and tools have been in place for many years. The ways of working have been in place for many years and these are being optimized over and over again. Instead, companies should ask themselves the question: "Why do we do this?"

The older companies started when hardware was very expensive and limited. Software was written in a matter of months or years. It was very important to write software that was right the first time because it took long to compile and deploy. Procedures were developed to ensure this was the case.

Figure 2

In many (larger and older) organizations we see this happening all the time. Products or services that were once very innovative, are being optimized, expanded, tweaked and polished. All is well until the moment a digital disruptor steps into the market who completely reinvents the product or service, taking over market share very quickly and aggressively. We all know the disruptors around us. Uber who runs a taxi service without taxis, AirBnB who provides accommodation but owns no real estate, and so on5.

The disruptors are usually newly founded companies, bearing no legacy and doing things differently right from the start. Something that is impossible for existing companies. Or not? Because when you think of it, the main thing the disruptors do differently is choosing reinvention instead of optimization.

The hardware was so expensive that it needed to be shared. Because of this, people needed to guard this hardware so that it was used correctly. Procedures were created to deploy software to these machines because this could only be done by certain people, and so on.

This example pinpoints procedures and roles that were created because of limitations or risks in the past. They are still in place because this is how we work. And in many cases that is the only reason left when you look at it closely. Hardware is not expensive anymore, we can write and compile software in minutes. Therefore, this needs a different approach. And this goes for a lot of things, including Continuous Delivery.

[4] http://xpir.it/mag3-cd30-3
[5] http://xpir.it/mag3-cd30-4

Continuous Delivery 3.0

Now that Continuous Delivery is on top of the hype cycle, it is also on top of the minds of many organizations. The assignment to the development teams (or now often called DevOps teams) is simple: move towards Continuous Delivery. This is exactly what most people do: start optimizing the process of delivery.

But now that we have learned that reflection on the reason why and going beyond optimization of your current tools and processes is important, we can apply these same concepts to the delivery process and move beyond the mere optimization and automation of the current process.

Do not optimize what you have, but rethink what you do and think about the next 'next step'!

In order to make a real difference, to really gain speed and quality in the delivery process, do not optimize what you have, but rethink what you do and think about the next "next step". In most cases, optimization and automation alone is not enough. The process needs to be thoroughly looked at and rethought. This is what Continuous Delivery 3.0 is all about. Do not optimize what you have, but rethink what you do and think about the next "next step"! Let me illustrate this with a few examples.

Machines, Virtual Machines, IaaS, PaaS, Containers

10-15 years ago all software was running on physical hardware, which was expensive and hard to scale. This completely changes with virtualization technology. Machines could be created in a matter of minutes, and the scalability issue was solved. Many organizations currently think about the next step. To deliver new functionality in a continuous way they even need more flexibility than Virtual Machines running on physical hardware. Of course the cloud offers this flexibility of unlimited space and unlimited machines, but is moving your Virtual Machines to the cloud as IaaS (Infrastructure as a Service) the right thing to do or is this an optimization?

In the Continuous Delivery 3.0 mindset, moving towards IaaS is perhaps not the right thing to do. When you rethink the possibilities, and move towards PaaS (Platform as a Service) and do not have to maintain machines anymore, moving your application into containers[6] [7](for example Docker) might be the next "next step", and therefore a better choice.

Manual Tests, GUI Tests, Test Automation

One of the most important things within a Continuous Delivery pipeline is the execution of automated tests. Writing automated tests is hard and although a lot of companies invest heavily in the automation of tests, testing is still a very manual job. When moving towards Continuous Delivery, tests need to be automated to ensure quality over and over again. Because manual testing has been done for many years, the logical optimization is to automate these manual tests. In most cases this means GUI testing. But again, this is not what brings you further.

The next "next step" goes beyond GUI testing. Test automation done right should be the area of investment. As the Agile Test Pyramid describes, a large base of fast and small technology-driven unit tests and only a handful of End-to-End GUI Tests allow you to run your tests quickly and reliably, and often this is the better approach.

Rethink your processes

These two examples illustrate how rethinking your process can be of great benefit. Looking at the Continuous Delivery Pipeline, many more examples can be thought of. For instance,

- Feature Toggles instead of branches to disconnect your deployment from a release. Breaking down your monolith into smaller independent components to be able to have different release cycles for different components.
- Building metrics and usage insights into your application to proactively act on user behavior instead of alerts and event to react on it.

These are all examples of how you can think ahead of the hype cycle.

Summary

Continuous Delivery 3.0 is all about thinking ahead. Instead of optimizing the things you already do, start thinking about how you can do it better or differently. Just like the digital disruptors disrupt well-known concepts and companies by being different and acting differently.

The best way to start within you own company is to ask the question: "Why do we do this and do the old constraints still apply?" When you start thinking about the next "next step", new and valuable possibilities will open up, in addition to transforming your Continuous Delivery process to the next level.

RENÉ VAN OSNABRUGGE
ALM LEAD CONSULTANT XPIRIT

René is always looking for improvement on all fronts. By using modern technology, implementing Continuous delivery, DevOps practices and coaching in the domain of Scrum and Agile, he helps companies to improve their software delivery process. As a MVP in Visual Studio and Development Technologies he is an active blogger and speaker at both national and international conferences where he shares his knowledge of his passion: Application Lifecycle Management.

Microsoft® MVP Most Valuable Professional

[6] http://xpir.it/mag3-cd30-5
[7] http://xpir.it/mag3-cd30-6

Release management, from technical to functional practice

Traditionally deployment and release of software have been synonymous to each other. As a result, the IT department is in charge of the time of release, and decides when the business can release a feature to the public. Separating the concerns of release and deployment allows you to improve the performance of continuous delivery pipelines and also empower the business to release features when they want to, without the involvement of IT. In this article I will show how you can apply the concept of separation of concerns in your continuous delivery strategy, specifically targeted at the notion of releasing and deploying your software to production. This enables faster deployment cycles and empowers the business to release features without the need to involve IT.

The difference between deploy and release

Let's start with the definition of a software release and what deployment entails. Releasing software or features is defined as exposing the software or feature to the end-user of the system. So this is the first moment the software or feature becomes available to the end-user. Deploying software includes installing the software on an environment (one or more machines in some kind of coherent configuration) to make it potentially accessible to the end-users. Based on these definitions you already can see why this has been something that has been done hand in hand at the same moment in time, since it all has to do with exposing new software to the end-user. But there is a subtle difference. Deployment is the activity of installing the software on an environment and release is the fact that the end-user can access it. So we can deploy our software without releasing features and we can release our features without deploying the software. It is possible to separate these two activities and carry them out at different moments, and this provides us with a huge number of advantages.

To mention a few:

- We can deploy at any moment in time, since it does not imply exposing new things to end-users, thus enabling deployments without requiring business approval. They are not impacted in any way by our deployments, so why would we need their approval? This enables continuous delivery for our teams without disrupting the business with our deployments.
- We can validate whether the new deployment shows the same behavior after installation and before exposing new functionality.
- We can determine whether the new software – which has new features but that are not yet visible to end-users – has different performance or stability characteristics that might need fixing before we expose it to the end-user. This gives us the ability to remediate issues before they are experienced by end-users, and thus give them more confidence in one-time-right delivery.
- We can empower the business units by enabling them to toggle features on or off at their leisure and when they require it, without the involvement of IT.

Feature toggles as a fundamental piece of the puzzle

How can we realize this separation of concerns? We need to have a way of making the exposure of features independent of installing the software in the production environment. The key to this is the concept of Feature toggles. Feature toggles are also known as Feature flippers, Feature flags, Feature switches, Conditional features, etc. A feature toggle is very simple in concept. It is a mechanism to turn a feature on or off, independent of the installation procedure. New features only become available when they are switched on. Application logic will check for the status of a feature switch and then decide whether to offer the feature or not.

A simple "if" statement that evaluates the feature toggle status is often enough to accomplish this. The real challenge lies in managing a set of feature toggles and how you want to expose the feature in the future. You have multiple options when it comes to how you are going to pick the group of end-users that will see a feature behind a switch. You can define an all-or-nothing switch,

or you can define a switch that only shows a feature to a particular set of customers, for instance a group that you have preselected as your very special customer group.
The use of feature toggles has been around for quite some time, but only recently has it gained in popularity. Instead of avoiding deployment of features that are not part of the release, we now consider it good practice to deploy both old, current and new features.

Of course, you could build your own sophisticated framework around feature toggles to read the status from configuration files or a database. But you might want to take a look at what is already available in open source that you could use in your product. A quick search returns a lot of frameworks, from which I picked a few that I have found useful in various projects. I also added the open source license of the software in the table below (See side note).

How can you implement the feature toggles?
You can incorporate a feature toggle framework in just a few simple steps. To explain in more detail how you can do this, I chose the framework "Feature Toggle" as an example. All other frameworks in the table are implemented in similar ways and each has its own unique set of capabilities. I chose "Feature Toggle" since it has out-of-the-box support for a database-backed feature toggle that works well in distributed systems that can run on multiple different machines.

A feature toggle for a typical ASP.NET MVC application is implemented by adding the nuget package to your project. You could create a specific folder that contains all feature toggles, which makes it easier to locate them during maintenance. Next, create a class that inherits from the base class SqlFeatureToggle as in code sample 1.

Framework Name	License	Link
NFeature	GPL	https://github.com/benaston/NFeature
Feature Toggle	Apache 2.0	https://github.com/jason-roberts/FeatureToggle
Feature Switcher	Apache 2.0	https://github.com/mexx/FeatureSwitcher
FlipIt	Apache 2.0	https://github.com/timscott/flipit

Short note on OSS licenses

When you use open source code in your software, you need to be aware of the license under which the software is published. Roughly speaking, there are three categories of licenses:

Strong Copy Left or so-called "viral" licenses
This includes licenses such as GPLv2 and GPLv3. These licenses involve the requirement that you must pass on all the same rights you received if you redistribute the covered program. This applies not only to the original program, but also to any modifications and additions. The GPLv2 states: "work based on the program", but the term "work based on the program" is vague whereas this is of key importance. It is not clear whether this means just derivative works of the original program or something much broader. So this is a tricky license type that can get you into trouble and potentially requires you to open-source your commercial software, including any potential patents that are part of the software!

Downstream or "weak copyleft licenses"
This includes licenses such as LGPL. The general requirement of these licenses is that, if the covered code is distributed, the same code must be provided downstream under the same license terms. Unlike Strong copyleft licenses, this mandate generally does not extend to improvements or additions to the covered code. The LGPLv2 (the "lesser" or "library" GPL) is classified as both a downstream and a copyleft license. The prevailing wisdom is whether or not the license is "viral" depends on how the covered library is linked to any proprietary code.

Attribution
This includes licenses such as BSD, MS-PL, MIT and Apache. These licenses are very basic and allow any kind of downstream use, including use in a commercial product, as long as the code contains appropriate attributions to the upstream authors.

Be aware!
Carefully evaluate which license you allow for your development and please always check this, since a dependency on open source, for instance a nuget package, is created in a second, but the consequences can be huge!

The next step consists of adding some configuration to the web.config file, where we configure the feature toggle framework. It includes a connection string to the database and a query for each feature toggle to determine whether the feature is turned on or off. To do so, add a section to the config file as shown in code sample 2.

This allows you to evaluate the feature toggle anywhere in the code. Just create an instance of the feature toggle class and then access the property FeatureEnabled to check the feature status. The query per feature gives the flexibility to store this information in any table of the database. Code sample 3 shows how to change the title of a web page based on the feature toggle.

As you can see, implementing these feature toggles is not rocket science. Now let's look at different release strategies and what we need to add in order to make the idea fully come to life.

Release strategies

Now that we have separated the release from deployment we can start thinking about what our options are when it comes to releasing features to the end-users. The way we distribute the new feature to the end-users needs to be based on the goal we want to achieve. There is a set of release strategies you can select from, and based on this, you need to pick the right implementation of your feature toggle. There are many strategies you can apply, some requiring you to add capabilities to the feature toggle framework, while others can be done with out-of-the-box functionality. Let me describe a few to give you an idea.

A/B testing

With A/B testing our goal is to test whether certain changes to your product will yield the result that you want to accomplish. For example, think about adding a special banner to your website with a deal of the day in order to increase conversion ratios. We can validate in production whether this is actually the case by selecting a cohort of users (a cohort is nothing more than a selected group of targeted users to which we expose the feature) and then watch whether the change yields the expected results. In this particular case we need to design our feature toggle in such a way that we can select the right set of users. This means that we need a set of criteria based on which we influence the toggle, or we can implement a toggle that randomizes a percentage of users to whom we want to expose the feature.
Important to note is that apart from the feature toggle, instrumentation also needs to be in place. You might want to track this using Google Analytics. If you are interested in feature usage statistics, then tools such as Microsoft Application Insights can show how a feature is used. However, the design of the telemetry we need for our validation needs to be part of feature development. This way of releasing features to end-users and testing the effects of the feature on your product in production is better known as

1
```
public class HomePagefeatureToggle : SqlFeatureToggle {}
```

2
```
<appSettings>
 <add key="FeatureToggle.HomePagefeatureToggle.ConnectionStringName"
   value="MusicStoreEntities"/>
 <add key="FeatureToggle.HomePagefeatureToggle.SqlStatement"
   value="Select status from featuretoggles where name ='HomePagefeatureToggle'"/>
</appSettings>
```

3
```
<div id="header">
 @{
  var featureToggle = new MvcMusicStore.Featuretoggles.HomePagefeatureToggle();
   if (featureToggle.FeatureEnabled)
   {
     <h1>
       <a href="/">ASP.NET MVC MUSIC STORE-With feature toggle in action</a>
     </h1>
   }
   else
   {
     <h1><a href="/">ASP.NET MVC MUSIC STORE</a></h1>
   }
 }
</div>
```

A/B testing. It has been used in the online marketing space for many years. Applying this to continuous delivery, our goal will be less targeted towards marketing, but more towards pipeline optimization and ensuring that our software budgets are spent wisely on features that are actually used and loved by our end-users.

Canary releasing

Another release technique is focused on discovering whether the new functionality we want to expose does not interfere with the standard use of your application and whether it does not introduce performance and scalability issues. In this scenario we build a new feature and include telemetry to measure the impact of the feature on usability, scalability and performance. Microsoft Application Insights can track the performance and scale impact. Moreover, by adding custom telemetry metrics we can also track the usage of the feature and see if people find the feature and use it as expected. We also need to allow a specific group of users to use the new feature in a controlled and highly monitored environment. To do so, you can use traffic management from Microsoft Azure, or "Testing in Production" when using Azure Web Apps. We call this technique Canary Releasing.

This term comes from the early days when miners brought a canary with them into the coal mines. A canary is more sensitive to toxic gases than the miners. If the canary suddenly died, then the miners knew to get out of the mines as quickly as possible. The same concept applies here, since we are validating in production in a controlled environment and we are closely watching our telemetry data. As soon as we see indicators that show that things are going wrong on the servers that host the new features, we can bail out and move all traffic back to the servers with only the current features.

Dark launching

The other common way of releasing is by exposing the feature to the end-user without him noticing. You will enable the complete dataflow, but not the actual UI. This means that you exercise the complete feature and closely measure the performance, scale and expected outcomes, without the end-user noticing this. You can think of this way of releasing as a headless canary release. You might think, ok so how do I expose a feature without an end-user noticing? Take two examples: a new calculation engine for your product or a new feature in which you show users where they are, based on their IP address. In these cases you can submit the data that is required for the calculations or location determination, without showing this to the end-user. You can measure the costs of compute and the impact on the scale. We only release the UI to the end-user when the feature shows what we expect and does not have a big impact and we do so by flipping a second feature toggle. In these kinds of releases, you typically have multiple switches exposing one layer in your architecture at a time. Dark launching is typically used by developers and IT to assess whether new features have significant impact on scale and performance. This enables them to ensure that when the business unlocks the feature for the end-user, no IT involvement is required.

Conclusion

Applying the concept of feature toggles allows us to create the ability to separate the concerns of deploying and releasing software. This empowers IT to deploy software to production any time they want, since it will not affect the end-users. This is a fundamental step in enabling continuous delivery and removing waiting times in the delivery pipeline for deployments. It also empowers the business to define release moments, since IT does not have to be involved when a feature is released to end-users. It enables a set of new release strategies that can help us be more effective in the development of new functionality while eliminating waste at the core, and to only build features that end-users like and use. Finally, it also provides more stability in our deployment process and enables us to be more in control of the whole design, build, deploy and release process, given we have built in the required telemetry to track what is going on.

MARCEL DE VRIES
CTO XPIRIT

Marcel spends most of his time looking at how new emerging technologies, a shift in mindset and a new way of working can help organizations get software in production faster.
Helping organizations transform towards a high speed, innovative and productive organization has become his passion. You can find him speaking regularly at industry events around the world, e.g. Visual Studio Live, Tech Days, Dev Intersection and Techorama.

Microsoft® Most Valuable Professional

Microsoft Regional Director

Infrastructure as Code

Your team is in the process of developing a new application feature, and the infrastructure has to be adapted. The first step is to change a file in your source control system that describes your infrastructure. When the changed definition file is saved in your source control system it triggers a new build and release. Your new infrastructure is deployed to your test environment, and the whole process to get the new infrastructure deployed took minutes while you only changed a definition file and you did not touch the infrastructure itself.
Does this sound like a dream? It is called Infrastructure as Code. In this article we will explain what Infrastructure as Code (IaC) is, the problems it solves and how to apply it with Visual Studio Team Services (VSTS).

Infrastructure in former times

We have radically changed the way our infrastructure is treated. Before the change to IaC it looked like this:
Our Operations team was responsible for the infrastructure of the application. That team is very busy because of all their responsibilities, so we have to request changes to the infrastructure well ahead of time.
The infrastructure for the DTAP environment was partially created by hand and partly by using seven PowerShell scripts. The order in which the scripts are executed is important and there is only one IT-Pro with the required knowledge. Those PowerShell scripts are distributed over multiple people and are partly saved on local machines. The other part of the scripts is stored on a network share so every IT-pro can access it. In the course of time many different versions of the PowerShell scripts are created because it depends on the person who wants to execute it and the project it is executed for.
The configuration of the environment is also done by hand.

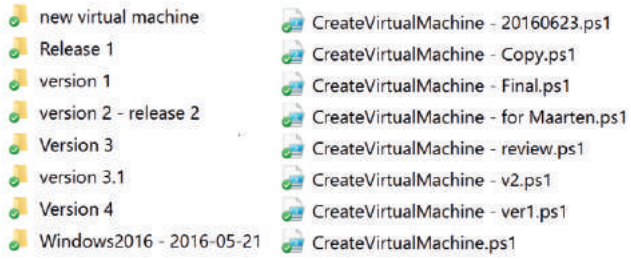


Figure 1: A typical network share

This process creates the following problems:
- Changes take too long before being applied.
- The creation of the environment takes a long time and is of high risk, not only because manual steps can be easily forgotten.
- The order of the PowerShell scripts is important, but only a single person knows about this order.
- What's more, the scripts are executed at a particular point in time and they are updated regularly. However, it is unclear whether the environment will be the same when created again.
- Some scripts are on the work machine of the IT-Pro, sometimes because it's the person's expertise area, and sometimes because the scripts are not production code. In either case, nobody else has access to it.
- Some scripts are shared, but many versions of the same script are created over time. It's not clear what has changed, why it was changed and who changed it. It's also not clear what the latest version of the script is. (See figure 1)
- The PowerShell scripts contained a lot of code. The code does not only contain the creation of resources, but also checks whether resources already exist and updates them, if required.
- The whole process of deploying infrastructure is pretty much trial and error.

As you can see, the creation of infrastructure is an error-prone and risky operation that needs to change in order to deliver high-quality, reproducible infrastructure.

Definition of Infrastructure as Code

Infrastructure as Code is the process of managing and provisioning computing infrastructure and its configuration through machine-processable definition files. It treats the infrastructure as a software system, applying software

Infrastructure as Code characteristics

Our infrastructure deployment example has the following infrastructure provisioning characteristics, which will be explained in the following paragraphs:
- Declarative
- Single source of truth
- Increase repeatability and testability
- Decrease provisioning time
- Rely less on availability of persons to perform tasks
- Use proven software development practices for deploying infrastructure
- Idempotent provisioning and configuration

Declarative

A practice in Infrastructure as Code is to write your definitions in

a declarative way versus an imperative way. You define the state of the infrastructure you want to have and let the system do the work on getting there. In the Azure Cloud, the way to use declarative code definition files are ARM templates. Besides the native tooling you can use a third party tool like Terraform to deploy declarative files to Classic Azure and to AzureRM. PowerShell scripts use an imperative way. In PowerShell you specify how you want to reach your goals.

Single source of truth
The infrastructure declaration files are placed in a source control repository. This is the single source of truth. All team members can see and work on the files and start their own version of the infrastructure. They can test it, and then commit changes to source control. All changes are under version control and can be linked to work items. The source control repository gives insight into what is changed and by whom.

Increase repeatability and testability
When a change to source control is pushed, this initiates a build that can test the change and after that publish an artifact. That will trigger a release which deploys your infrastructure. Infrastructure as Code makes your process repeatable and testable. After deploying your infrastructure, you can run standard tests to see if the deployment is correct. Changes can be deployed and tested in a DTAP pipeline.
This makes your process of deploying infrastructure reliable, and when you redeploy, you will get the same environment time after time.

Decrease provisioning time
Everything is automated to create the infrastructure. This results in short provisioning times. In many cases a deployment to a cloud environment has a lead time of 5 to 10 minutes, compared to a deployment time of days, weeks or even months.

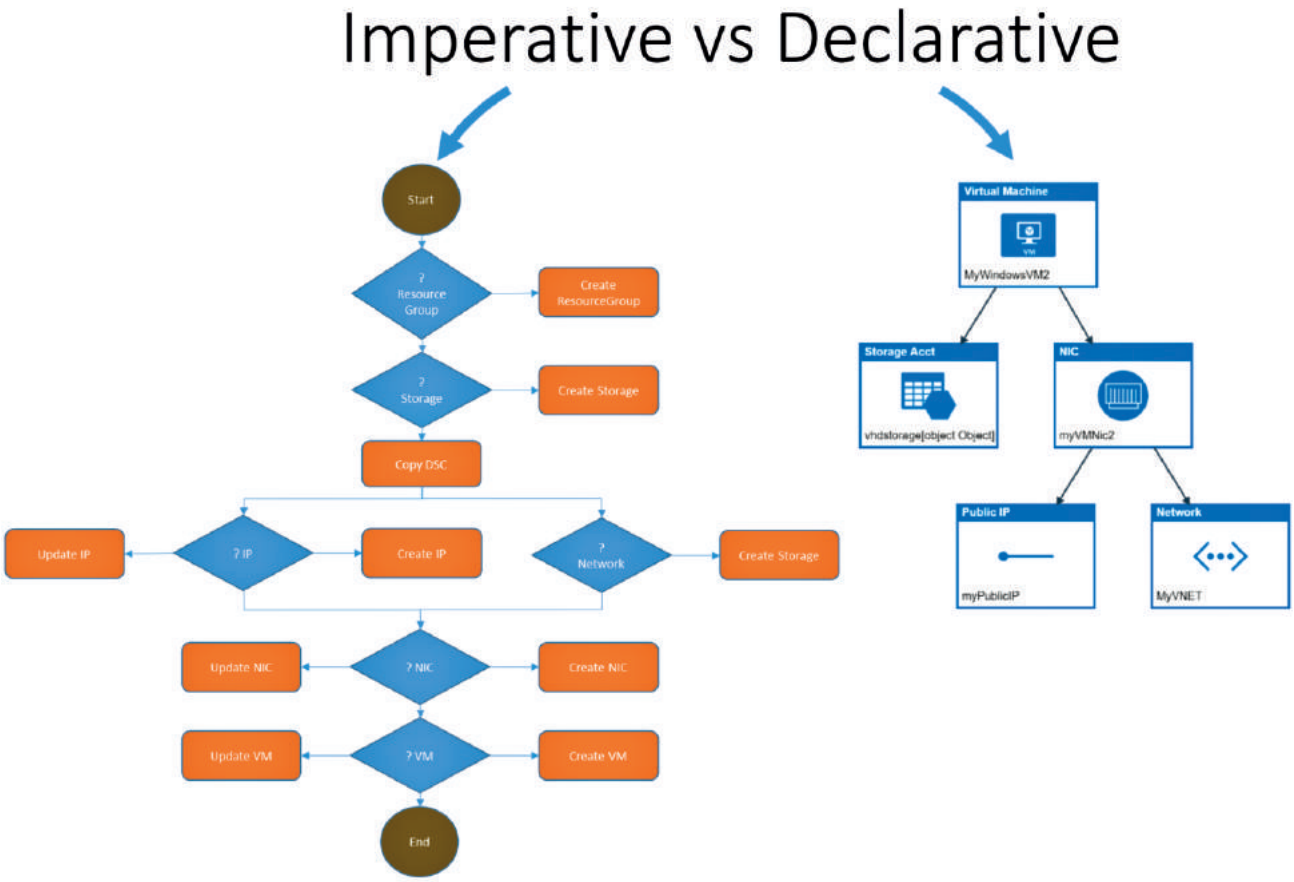


Figure 2: Schematic visualization of Imperative vs Declarative

The link to the work item can tell you why it was changed. It's also clear what the latest version of the file is. Team members can easily work together on the same file.

This is accomplished by skipping manual tasks and waiting time in combination with high-quality, proven templates. The automation creates an environment that should not be touched by hand. It handles your servers like cattle instead of pets*. In case of problems there is no need to logon to infrastructure to see what is going wrong and trying to find the problem and fix it. Just delete the environment and redeploy the infrastructure to get the original working version.

Rely less on availability of persons to perform tasks
In our team, everybody can change and deploy the infrastructure. This removes the dependency on a separate operations team. By having a shared responsibility, the whole team cares and is able to optimize the infrastructure for the application.

This will result in more efficient usage of the infrastructure deployed by the team. Operations is now spending more time on developing software than on configuring infrastructure by hand. Operations is moving more to DevOps.

*Pets vs Cattle
Is a widely used metaphor for how IT operations should handle servers in the cloud.

Servers are like pets
You name them and when they get sick, you nurse them back to health.

Servers are like cattle
You number them and when they get sick, you get another one.

Use proven software development practices for deploying infrastructure
When applying Infrastructure as Code you can use proven software development practices for deploying infrastructure. Handing your infrastructure in the same way you handle your code, helps you to streamline the whole process. You can start and test your infrastructure on each change. Using Source control as a team is a must. The sources that it contains should always be in the state in which they can be executed. This results in the need for tests such as unit tests.

Idempotent provisioning and configuration
Creating an idempotent provisioning and configuration for provisioning will enable you to rerun your releases at any time. ARM Templates are idempotent. This means that every time they will be executed the result will be exactly the same. The configuration is set to what you have configured in your definitions. Because the definitions are declarative, you do not have to think about the steps on how to get there; the system will figure this out for you.

Creating an Infrastructure as Code pipeline with VSTS
There are many tools you can use to create an Infrastructure as Code pipeline. In this sample we will show you how to create a pipeline which deploys an ARM template with a Visual Studio Team Service (VSTS) build and release pipeline. The ARM Template will be placed in a Git repository in VSTS. When you change the template, a build is triggered, and the build will publish the ARM template as an artifact. Subsequently, the release will deploy or apply the template to an Azure Resource group.

Prerequisite
To start building Infrastructure as Code with VSTS you need a VSTS account. If you don't have a VSTS account, you can create one at https://www.visualstudio.com. This is free for up to 5 users. Within the VSTS Account you create, you then create a new project with a Git repository. The next step is to get some infrastructure definition pushed to the repository.

ARM template
ARM templates are a declarative way of describing your infrastructure. ARM templates are json files that describe your infrastructure and can contain 4 sections: parameters, variables, resources and output. To get started with ARM templates you can read "Resource Manager Template Walkthrough"[1].

It is possible to create ARM templates yourself by choosing the project type Cloud → Azure Resource Group in Visual Studio. The community has already created a lot of templates that you can reuse or take as a good starting point. The community ARM templates can be found on the "Azure Quickstart Templates"[2]. ARM templates are supported on Azure and also on-premise with Microsoft Azure Stack.

Figure 3: VSTS source control, build and release

In our example we want to deploy a Web App with a SQL Server database. The files for this configuration are called "201-web-app-sql-database[3]". Download the ARM template and parameter files and push them in your Git source control repository in your VSTS project.

VSTS Build
Now you are ready to create the build. Navigate to the build tab in VSTS and add a new build. Use your Git repository as the source. Make sure you have Continuous Integration turned on. This will start the build when code is pushed into the Git repository. As a minimum, the build has to publish your files to an artifact called drop. To do this, add a Copy Publish Artifact step to your build and configure it like this:

[1] Resource Manager Template Walkthrough http://xpir.it/mag3-iac1 .
[2] Azure Quickstart Templates http://xpir.it/mag3-iac2
[3] 201-web-app-sql-database http://xpir.it/mag3-iac3

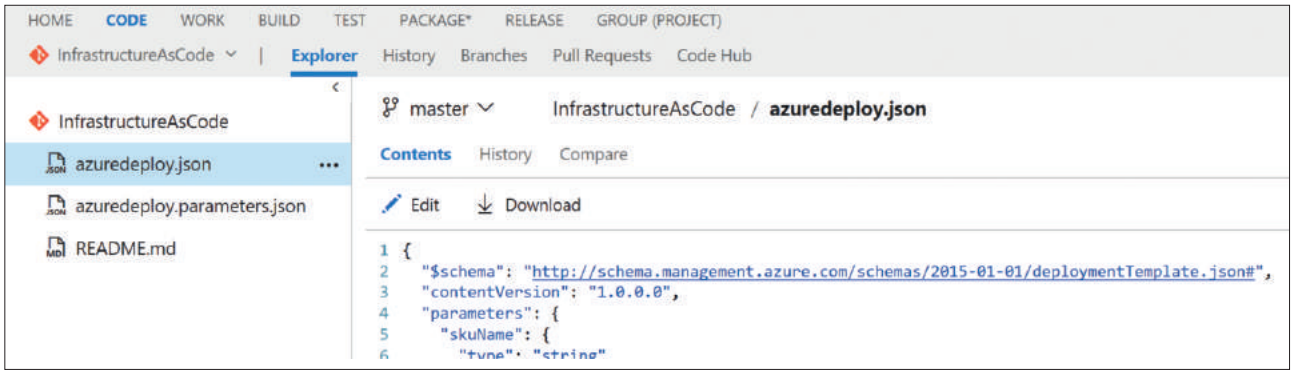


Figure 4: ARM template in Git

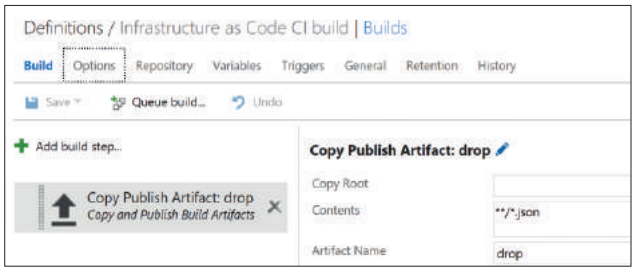


Figure 5: Copy Publish Artifact configuration

VSTS Release

The next step is to use VSTS Release for deploying your infrastructure to Azure. To do so, you navigate to release and add a new Release. Rename the first environment to Development and add the task Azure Resource Group Deployment to the Development environment. This task can deploy your ARM template to an Azure

Resource group. To configure your task, you need to add an ARM Service Endpoint to VSTS. You can read how to do this in the following blogpost: http://xpir.it/mg3-iac4. Now you can fill in the remaining information, i.e. the name of the ARM template and the name of the parameters file (fig. 6).

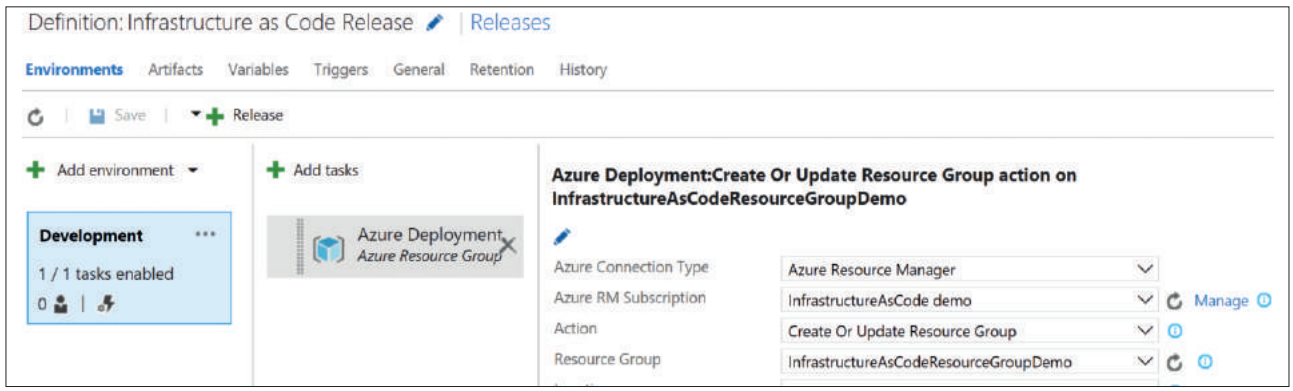


Figure 6: Azure Resource Group deployment configuration

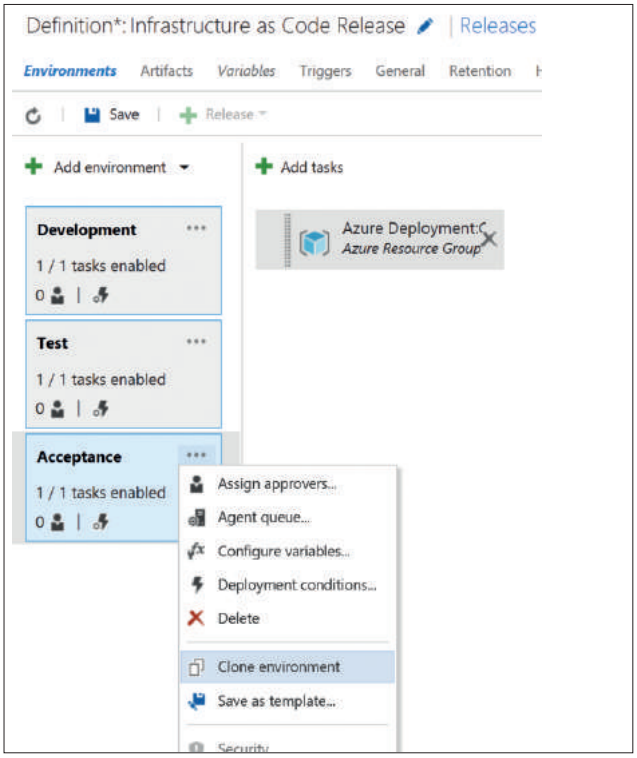


Figure 7: Clone an environment in Release

DTAP

At this point you only have a Development environment. Now you are going to add a Test, Acceptance and Production environment. The first step is to create the other environments in VSTS release manager. Add environments by clicking the Add environment button or bycloning the development environment.

Infrastructure as Code will help you to create a robust and reliable infrastructure in a minimum of time.

Each environment needs separate parameters, so you need to create a parameter json file per DTAP environment. Each environment gets its own azuredeploy.{environment}.parameters.json file, where {environment} stands for development, test, acceptance or production.

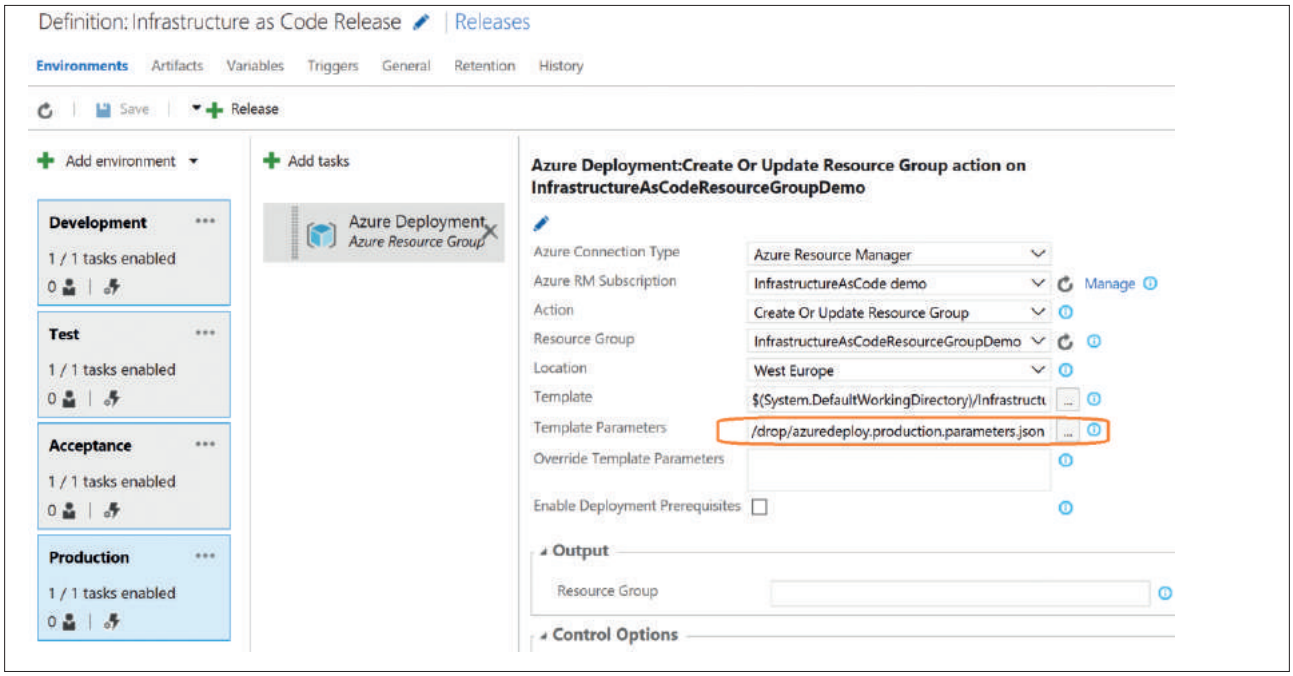


Figure 8: Configure each environment to a different parameters file

The deployment can be changed to meet your wishes. For example, deploy to a separate ResourceGroup in Azure per DTAP environment. Now you have your first version of an Infrastructure as Code deployment pipeline. The pipeline can be extended in multiple ways. The build can be extended with tests to make sure the infrastructure is configured as it is supposed to be. The release can be extended by adding approvers, which makes sure that an environment will only be deployed after an approval of one or more persons.

Conclusion

Infrastructure as Code will help you to create a robust and reliable infrastructure in a minimum of time. Each time you deploy, the infrastructure will be exactly the same. You can easily change the resources you are using by changing code and not by changing infrastructure.

When you apply Infrastructure as Code, everything should be automated, which will save a lot of time, manual configuration and errors. All configurations are the same, and there are no more surprises when you release your application to production. All changes in the infrastructure are accessible in source control. Source control gives great insight in why and what is changed and by whom. A DevOps team that applies Infrastructure as Code is self-contained in running its application. The team is responsible for all aspects of the environment they are using. All team members have the same power and responsibilities in keeping everything up and running, and everybody is able to quickly fix, test and deploy changes.

PASCAL NABER
CLOUD CONSULTANT XPIRIT

Pascal is a passionate software developer and technical software architect who primarily focuses on quality and simplicity. He has a great interest in everything that's related to Microsoft software development and new technologies.

PETER GROENEWEGEN
CLOUD CONSULTANT XPIRIT

Peter is Lead Azure Consultant. He is helping software development teams with software development, UI Testing, Unit testing and their CI/CD pipelines on the Azure Stack. He has an interest in new technologies like the Hololens and he is an active Stack Overflow contributor. In 2016 Peter was awarded the Microsoft Most Valuable Professional award for Visual Studio and Development Technologies.

Microsoft MVP Most Valuable Professional

Powered by

Microsoft NeXtGen

.NET Rocks!

nc comms

DEVintersection europe 2016

14-16 NOVEMBER 2016

HAARLEM, NL

ASP.NET
Visual Studio
Angular 2 • Security
Windows 10 • Xamarin
ES6/TypeScript • C#
Architecture
SPA
and more

<anglebrackets/> europe 2016

FOR LOVERS OF THE OPEN WEB

THE INTERSECTION OF TECHNOLOGY

DEVintersectionEurope.com

BRAD GREEN
Engineering Director
Google

DAVE MENDLEN
General Manager at Microsoft in DX
Microsoft

BARRY DORRANS
ASP.NET Security Analyst
Microsoft

JUVAL LOWY
Founder
IDesign, Inc.

GIL CLEEREN
.NET Architect
Ordina

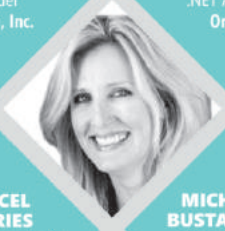
RICK VAN ROUSSELT
Senior SharePoint/ O365 Consultant
Rivaro Consultancy

DANIEL ROTH
Senior Program Manager, ASP.NET Team
Microsoft

RADOSLAV KIRAV
Angular Core Team
Google

LAURENT BUGNION
Senior Director
IdentityMine

MARCEL DE VRIES
Co-founder and CTO
Xpirit



MICHELE L. BUSTAMANTE
CIO & Architect
Solliance, Inc.

HANS LARSEN
Software Engineer
Google

KATHLEEN DOLLARD
.Net Coach
Crystal MEF Lab

RICHARD CAMPBELL
Host
RunAs Radio

CARL FRANKLIN
Franklins.net

DAN WAHLIN
Developer, Google GDE and Microsoft MVP
Wahlin Consulting

DOMINICK BAIER
Trainer and Software Developer
Independent Consultant

BRIAN NOYES
CTO and Architect
Solliance, Inc.

ROB WORMALD
Developer Advocate
Google

0044 1925 758565
DEVintersectionEurope.com

DEVintersection U.S. Events

October 25-28, 2016
MGM Grand
Las Vegas, Nevada

May 21-24, 2017
Walt Disney World Swan
Orlando, Florida

DEVintersection.com

Mobile Innovation: Don't Wait for the "Perfect" App

JOSEPH HILL
XAMARIN CO-FOUNDER & PRINCIPAL DIRECTOR OF PROGRAM MANAGEMENT AT MICROSOFT

In less than a week, Ninatic's Pokémon Go took Nintendo's popular franchise into the real world and became the planet's most popular app. By combining with Augmented Reality, location-based services, gestures and touch mechanics, as well as a cloud-based logic and storage, Niantic produced a uniquely mobile gaming experience that redefined its category while collecting record earnings. .

The mobile home run

Mobile developers everywhere are looking for their Pokémon Go – that unique, transformative, "showstopper" experience that combines the best of what mobile devices offer with an untethered use case to create something entirely new. And they're not as rare as you might think. While consumer apps like Pokémon Go and Uber get most of the attention, there are thousands of B2B and B2E apps that are every bit as successful and innovative. For example, Novarum DX built an app that uses smartphones and their cameras to process tests that detect the presence of diseases such as Ebola, or harmful chemicals in water supplies. The app replaces sophisticated laboratory equipment and requires no additional hardware, allowing field workers with minimal training to analyze tests on the spot, bringing life-saving testing to areas that were previously inaccessible.

While saving lives with a mobile app is an obvious win, not every business has the right set of circumstances to replicate that kind of success. But that doesn't mean they're locked out of the kinds of digital transformation mobility can bring. In fact, if you're only looking for headline-making apps, you're probably missing out on a lot of immediate innovation right in front of you.

Incremental transformation

In the long term, businesses succeed and fail on the thinnest of margins. Whether you're a five-person startup or a Fortune 500 enterprise, reducing costs or increasing productivity by even a fraction of a percent can make a significant difference in near-term success. And over time those incremental efficiencies can add up to something much bigger.

Incremental efficiencies can add up to something much bigger.

We see this constantly with customers looking to improve their core business processes with mobility. These projects may not make headlines, but the opportunity they represent is substantial. Bloomin' Brands lets its Outback Steakhouse customers receive coupons, get on the guest list, and pay for their meals from mobile devices. McKesson helps doctors replace time-consuming paper forms with electronic records and a mobile app. Xactware, which processes 80% of U.S. home property claims, created a mobile app to help insurance adjusters receive assignments, build 3D models, estimate costs, settle onsite, and instantly upload all materials for processing while in the field. While these companies may not have created entirely new businesses, they all built dramatically more efficient, delightful experiences for their employees and customers.

Cultural reinvention

In most cases, the immediate financial results of a truly mobile strategy are enough to justify the investment – happy customers are profitable customers, and efficient workplaces mean increased productivity. But there's a secondary benefit, as well. Mobility is culturally transformative. It helps executives shift from "How much" to "What if." It allows IT to add quantifiable value, moving from a cost center to a profit center. And it helps engage developers who might otherwise look elsewhere. Recently, a large, traditional healthcare company launched a small proof-of-concept mobile program with a very limited budget. As the project succeeded, developer interest from outside the team soared as the project offered an opportunity to engage in what the project lead called "the cool new thing we had going on", without leaving the security of their current company.

Build fundamentally mobile experiences

Of course, not every mobile story has a happy ending. You don't have to look hard to find cautionary tales of mobile apps that failed to deliver. There are as many reasons for this as there are apps, but the common thread through most of them is a lack of understanding of the mobile use case. If mobility is just a checkbox – a rehash of your desktop application crammed into a smaller screen – the project is headed for mediocrity or worse, and you'll probably create as many problems as you'll solve.

Our mobile devices have to be more than just PCs with limited storage, small screens, and small keyboards. Porting desktop or Web apps (yes – even "responsive" websites) to those devices plays to

the worst of the platform and ignores the best. By all means, businesses must extend their systems to mobile endpoints, but they should do so by creating mobile-first experiences that integrate with existing systems of record. That means embracing mobile-only features like biometric authentication, push notifications, voice communication, GPS, and other sensors. It means offsetting form factor limitations by embracing cloud-based intelligence services like face recognition and predictive analytics. And most of all, it means building beautiful, performant native applications that are every bit as satisfying to users as the latest consumer apps they downloaded from their app store.

What's Next

The scope of mobile development is growing at an astounding rate, covering as much ground in a month as desktop development did in a year. In just the 8 years since the launch of the iPhone, we've gone from email, calendar, and PIM to 3D interfaces, voice-driven controls, and intelligent services that predict and surface information before you know you need it. As we continue that journey toward more immersive, natural interfaces on increasingly diverse types of devices, two trends will become increasingly important. The first is deeper integration into operating systems and hardware, to wring maximum performance out of devices and provide native UI on every platform. The second is an increasingly intelligent, responsive cloud service that delivers seamless, modern experiences to the widest possible number of platforms without sacrificing developer efficiency.

Act Now

A lot of interesting technology is on the way, but don't wait to get started. It's been said that "the perfect is the enemy of the good," and that's never been more true than today. For every game-changing mobile pure-play that makes headlines, there are hundreds of companies that built steady streams of innovation over time, and that all starts with one app. Then another, and another...

JOSEPH HILL
XAMARIN CO-FOUNDER & PRINCIPAL DIRECTOR OF PROGRAM MANAGEMENT AT MICROSOFT

Joseph Hill is a Xamarin Cofounder and Principal Director of Program Management at Microsoft. Joseph has been an active participant in the Mono community since 2003, and has also contributed to several open source .NET applications. As a professional developer, he has worked with several Fortune 500 companies in designing and implementing .NET applications. In January 2008, Joseph joined Novell to serve as the Product Manager for Mono, ultimately driving the product development and marketing efforts to launch Xamarin's commercial products.

Exploring the Microsoft Bot Framework

At the Microsoft Build Conference in 2016, Microsoft announced something they call "Conversations as a platform". This stated human interaction and machine learning as the next computing interface.

The goal is to turn a conversation into a more powerful tool that is contextually rich while improving productivity. As Satya Nadella said, the human language is the new user interface. Microsoft is investing substantially in this area with Cortana, Cognitive services and Bot framework. In the end Microsoft sees a convergence from messaging with people to messaging with services.

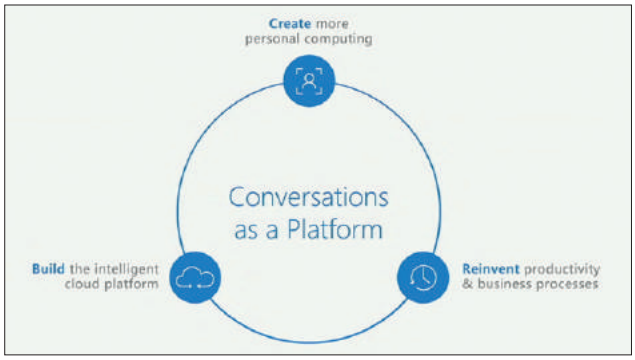


Figure 1

The convergence towards conversations starts with the Cortana Intelligence Suite, which consists of 3 main parts.

- Bot Framework. We will explore this framework in more detail in this article.
- Cognitive Services. For example, Language Understanding Intelligent Service (LUIS)
- Machine Learning. Which is the basis for the language understanding of LUIS.

Bot Framework

One of the main building blocks for this next computing interface are bots. Microsoft announced the bot framework to help developers build rich bots with easy integration to many channels and other services. The bot framework consists of three main parts, as seen in the following diagram.

Bot builder SDK

The Bot Builder SDK is an open source SDK that provides all the building blocks you need to develop your Bot using .NET, NodeJS.

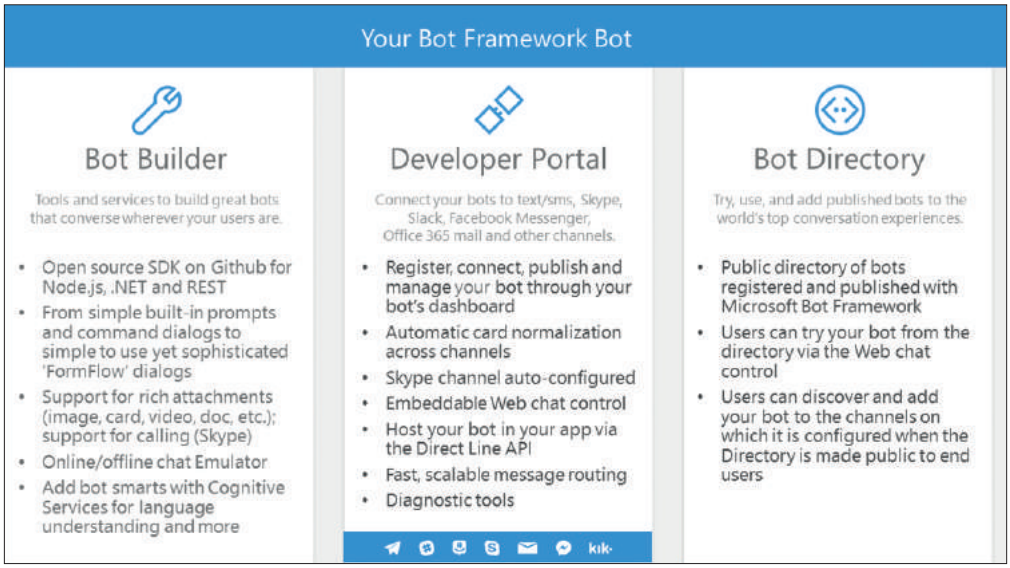


Figure 2

It also consists of a chat bot emulator and Visual Studio templates. The emulator makes it easy to test your bot while it is being developed, but in the end the bot is just an API. You could also use postman or any other tool to test your bot.

Developer Portal

The developer portal is the place where you can register your Bot and connect it to a wide range of communication channels. Here you get a step-by-step guide on how to add more channels to your bot. The Skype and Web Chat channels are added by default when you first register your bot in the developer portal.

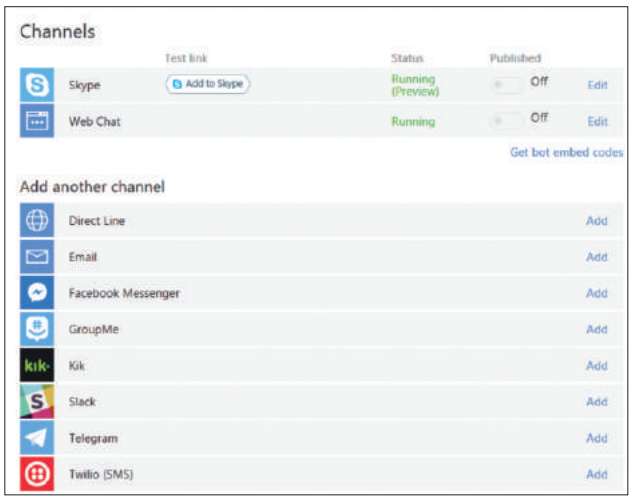


Figure 3

Bot directory

When you are done building your Bot you can publish your Bot to the Bot Directory, which contains different Bots which are already published and are ready for use. When your Bot is submitted to the Bot directory it usually takes a few days to get through the validation process. After that other users can add your bot to their channels.

Building a Bot

When you first create a new Bot in Visual Studio, the project should look very familiar if you previously developed a Web API. There is a Controllers folder and a MessagesController which handles the requests coming into the bot. The Post method gets an Activity class which describes the message/event the bot receives. The HandleSystemMessage method allows you to handle different Bot events such as users entering the conversations or leaving.

Dialog

A dialog is a class with state and methods that use Bot Builder-defined types to manage interactions. The Bot Framework manages conversations by sending JSON messages through the Bot Connector and to/from the user and bot. These messages hold the state of an ongoing conversation. To facilitate managing state with the Bot Framework, your dialog type must be serializable. To support this, the code is decorated with the [Serializable] attribute. Additionally, the class contains fields, representing the state of the conversation. The Bot framework dialog manages all the state for you if you follow this simple guidance.
When implementing a dialog, you simply chain interactions and guide the user through the conversation one step at a time. You have to implement the Start method for your dialog and from there you can chain methods to guide the user to the end of the dialog. A code snippet from a dialog is seen below (code 1).

FormFlow

FormFlow is a Bot Builder SDK library that lets you declare the type of information you need and then it does the bulk of the work

1
```
public async Task StartAsync(IDialogContext context)
{
    context.Wait(ConversationStartedAsync);
}

public async Task ConversationStartedAsync(IDialogContext context,
    IAwaitable<IMessageActivity> argument)
{
    var message = await argument;
    await context.PostAsync(message.Text);

    PromptDialog.Choice(
        context: context,
        resume: ResumeAndPromptRatingAsync,
        options:
        Enum.GetValues(typeof(Consultants)).Cast<Consultants>().ToArray(),
        prompt: "Welcome to the Xpirit Feedback Bot! Please select the consultant
                 you want to provide with feedback:",
        retry: "I didn't understand. Please try again.",
        promptStyle: PromptStyle.Auto);
}
```

of managing the conversation and moving the user from question to question automatically. In Dialogs, you have the responsibility of writing the methods to prompt the user and collect the results, but FormFlow simplifies the entire process. Another benefit of FormFlow is that it automatically formats questions to be more readable and it tries to process partial answers if it can.
When rebuilding the Xpirit FeedbackBot to use FormFlow instead of Dialogs the whole implementation is an entity with the fields you wish to gather from the user. So instead of specifying the flow yourself, the Bot Framework handles that for you. (code 2)

So while you have to build the dialog yourself using Dialogs, the FormFlow does that for you and you only have to focus on the information that you want to gather. There are a lot of additional options available to customize how the FormFlow method renders things, as you can see in the above code sample with the [Prompt] attribute.

Conclusion

Microsoft has made it very easy to get started with building and exploring the bot world. However, there are also a lot of things you can add to your bot when you start combining the Bot framework and LUIS or other cognitive services. Microsoft has even made it very easy to use LUIS (Language Understanding Intelligent Service) from the Bot Framework by providing base classes and attributes to integrate LUIS into your Bot. As seen in the code sample it is very easy to include LUIS into your Bot dialog. (code 3)

All the sources for the bot build in this article are available on Github.

2

```
[Serializable]
public class XpiritFeedbackForm
{
    [Prompt("Choose the {&} to give feedback {||}")]
    public Consultants Consultant { get; set; }
    public Ratings Rating { get; set; }
    public string FeedbackMessage { get; set; }


    public static IForm<XpiritFeedbackForm> BuildForm()
    {
        OnCompletionAsyncDelegate<XpiritFeedbackForm> processForm = async (context,
                state) =>
        {
            await context.PostAsync("We are currently processing your feeback. We will message
                                    you the status.");
        };

        return new FormBuilder<XpiritFeedbackForm>()
                .Message("Welcome to Xpirit Feedback bot!")
                .OnCompletion(processForm)
                .Build();
    }
}
```

3

```
[Serializable]
public class XpiritFeedbackDialog : IDialog<object>
{
    ...
}

[LuisModel("key", "key")]
[Serializable]
public class XpiritFeedbackDialog : LuisDialog<object>
{
    ...
}
```

CHRIS VAN SLUIJSVELD
CLOUD LEAD CONSULTANT
XPIRIT

As a Lead Consultant at Xpirit, Chris is helping customers with implementing a microservices-enabled platform using Service Fabric, API Management and API design guidelines. Chris loves to experiment with new technology and tweets, and then blogs about this on the internet. Chris is keen on adopting new technologies and investigating how they can deliver more value for the customer.

[1] http://xpir.it/mag3-bot

Continuous delivery 3.0 for mobile apps

Continuous delivery is a hot topic in the current world of software development, especially for web applications. Most teams and companies I know of are starting to experiment with it or are in the process of implementing continuous delivery. In my role as consultant I get to see a lot of different companies and their ways of working. A lot of companies have some automated testing and automated deployments to certain environments in place. However, when you look at mobile application projects, you often don't find any sign of continuous delivery.

Setting up some basic principles of continuous delivery for mobile isn't that hard, although there are some extra hurdles for mobile applications compared to web applications, for example the deployment process towards app stores. The first steps in implementing continuous delivery could consist of setting up continuous integration with automated builds and unit tests running at every code check-in. Another step could be to set up automated deployments towards private stores (with tools such as Hockeyapp), so your test team always has a new, daily version of the app on their test devices. This article is not about setting up the basics of continuous delivery which we call continuous delivery 1.0. Instead, it is aimed at creating a new mindset about continuous delivery. Welcome to the world of continuous delivery 3.0.

What is continuous delivery 3.0?

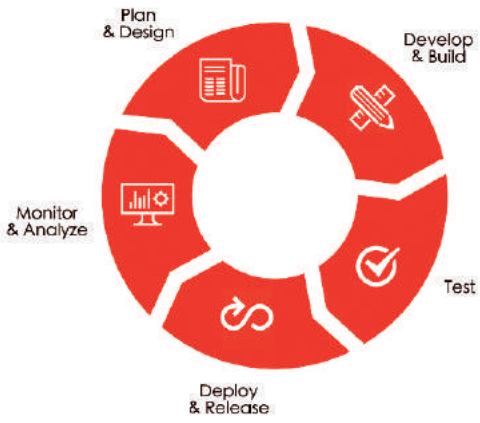
Continuous delivery 3.0 is a term my Xpirit colleague Rene van Osnabrugge came up with to define the next level of continuous delivery. It's hard to say exactly what continuous delivery 1.0 or 2.0 is. However, what we mean to say with CD 3.0 is that when you really want to get to the next level of continuous delivery, you have to rethink certain parts of your software delivery pipeline instead of just putting some automated deployments in place. This article covers three areas you might want to rethink when implementing continuous delivery for mobile apps.

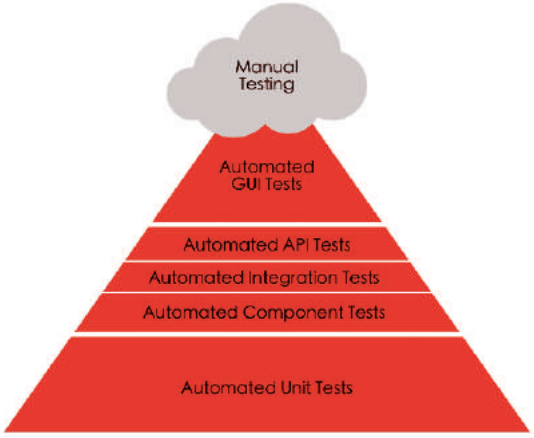
Rethink testing

Every developer will agree that proper automated testing is important in all software development projects. However, it may be even more important in mobile app projects than in web projects. On the web it's possible to implement a fast update in the application on your web servers, but with native mobile apps you'll have to go through app store submissions and reviews, and this can take up to several days. What's more, competition within the app stores is huge and it's easy for users to just open the store and download the app from your competitor if your app contains bugs or doesn't work properly. If you lose users on a mobile platform, it's really hard to get them back. They are also very likely to leave 1-star ratings and never return after switching to a competitor.

Automate everything

To be able to maintain high quality levels at every moment in time, automated tests are key. Mike Cohn developed the concept of the agile test pyramid which describes how to create a properly balanced set of tests. The key point of the test pyramid is that you need to create a large foundation of unit tests that can run fast, as well as a much lower number of end-to-end test cases that run through the GUI.





Structure your application so you can mock or stub all internal and external dependencies in your code and build unit tests to cover all code paths within your solution. The unit tests form a base of tests that you should be able to run as often as possible. This is why they need to be fast, and with fast I mean being able to run hundreds of tests within a few seconds. If these kinds of tests take too long, people will stop running them and the goal of these tests is lost.

In addition to the unit tests there is a set of component tests that connect several units together, as well as integration tests and API tests which test all server side functionality of your mobile apps.

Setting up a clear API layer is key for your architecture and this splits up testing responsibility between your mobile app layer and back-end layer of the application. The APIs form a contract of communication between these two layers and if you test the APIs by themselves, it should be possible to release them separately. By having a good API stub framework, the automated mobile app tests only have to communicate to these stubs instead running tests from the UI to the back-end system.

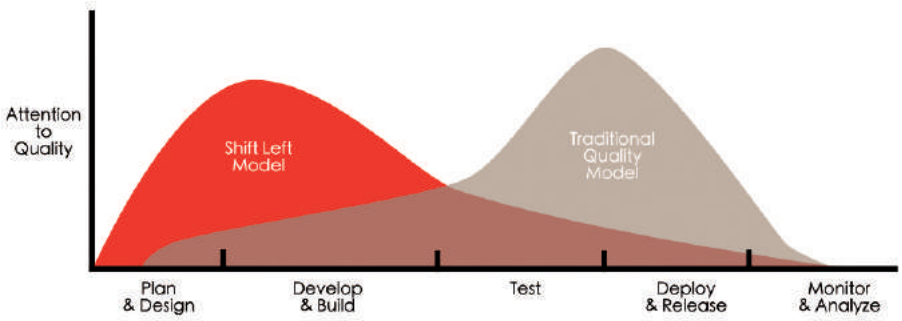
The mobile device landscape involves an enormous amount of complexity with different types of operating systems, OS versions, screen sizes, processor and memory differences. Manual testing of all these device combinations is impossible. Solutions such as Xamarin Test Cloud can help you by providing a cloud farm of thousands of devices, running all possible combinations of operating systems and versions. Tools like this integrate with your continuous integration builds, so you can run these tests with every automated build.

These methods of automated testing allow you to focus your manual testing efforts on edge cases which are hard to automate. Think of scenarios like receiving a phone call while running your app, or suddenly losing reception.

Automating your current tests is not enough

While automating your test effort is a good start, it isn't actually rethinking your test strategy. Rethinking your test strategy means that you need to make a shift to focus on quality earlier in the software development process instead of only reducing the test effort.

This means that the role of your testers is going to change. There is no room for testers who write test scripts based on specifications or who try to break your application by monkey testing.



Making a shift to an earlier phase of your development process is important in order to have high-quality code at every moment of the cycle. Instead of writing test scripts, testers should be helping the business with writing testable requirements that can be used in automated test scripts. Popular ways to do this are ATDD (Acceptance Test Driven Development) or BDD (Behavior Driven Development). Below is an example of a BDD specification written in the Gherkin language. Tools such as Specflow can turn these specifications into automated tests. It is possible to combine Specflow tests and the Xamarin test cloud to have your requirements tested automatically on thousands of devices. (code 1)
Combining these kinds of tests with the automated unit tests create a stable level of quality right from the start of the development process up to the first release, as well as future releases.

Rethink releases

If the quality of your code is measurably high at all times, it should be possible to deploy the application at regular intervals. However, deploying is totally different from releasing.

1
```
Feature: Login
  In order to access my account
  As a user of the website
  I want to log into the website

Scenario: Logging in with valid credentials
  Given I am at the login page
  When I fill in the following form
  | field | value |
  | Username | xtrumanx |
  | Password | P@55w0Rd |
  And I click the login button
  Then I should be at the home page
```

For more details, please read the article by Marcel de Vries on separating deployments and releases, also published in this magazine. Mobile apps have a number of extra difficulties regarding releases. When you create a public app in the stores you'll have to go through the official review cycles by Apple, Google and Microsoft, and this can take up to several days. This makes quality in your app even more important because there is no way to quickly release bug fixes as they have to go through the same review cycle.

Public beta releases

A lot of companies I visit use tools to distribute the apps that are in development to their testers. The most common tool for this job is Hockeyapp. Hockeyapp works great for test app distribution and is easy to integrate into automated build or release tools such as VSTS. When you're rethinking releases you will end up trying to release as often as possible and also make early releases available to end-users. Hockeyapp can support such beta releases. However, it is not designed for large public beta tests because managing large sets of users takes a lot of manual effort. Apple, Google and Microsoft all have features for releasing beta apps to groups of users but they all involve a number of disadvantages: Apple still requires you to go through a review cycle that can take several days and a beta release only has a lifetime of 60 days.
Another option would be to publish two apps to the stores, one with a beta tag and one without. This option is valid for Android where you could do early releases to the beta version in the store and use the app without a beta tag for a more formal release at a slower pace. However, this is not an option for iOS because Apple forbids adding beta versions of apps to iTunes. So is there any other way to add beta functionality to releases that are released to the store?

Dark Launching & Feature toggles

Feature toggles are an option that allows you to integrate features into your app without enabling them immediately when you technically release an app. When you create feature toggles, you start with adding a configuration option to enable or disable a new feature without implementing the feature in your code. You should be able to release the app as long as the configuration remains disabled. Now you can implement the feature and switch the feature on as soon as it is ready to use. If you can control the switches of the app from a central location, this can be extended even further by letting your business users decide which features will be enabled. There are several tools in the market that help you with the management of these feature toggles to be able to dark launch your app's features. Many of these tools also have features that allow you to enable new features to only a small set of users in order to see the impact of the feature before releasing it to everyone.

Rethink analytics

Having health monitoring in your apps seems obvious. There are several tools to implement this and every customer I visit seems to have some basic health monitoring installed. Rethinking analytics is thinking about the extra value analytics can offer you.

Functional insights

The first step in rethinking analytics is to start using the analytics features to only see health checks. It can be really useful to see which pages are used most, which features are used most often, or the keywords your users are using for searching within your app. Most mobile analytics tools are able to do this and it can give you valuable insights that will enable you to improve your app.
An even more advanced scenario would be that you set up your analytics in such a way that it measures differences between certain functionality and automatically switches feature toggles when errors occur, or that it measures decreased use of certain features after enabling a particular feature toggle.

Data is driving more and more decisions and that's why you never want to throw away any data. The same goes for your analytics data. Combining the functional analytics of, for instance, the moments when your app is used most frequently, the most popular features and other things that can be traced with your company's big data can lead to new insights. Machine learning is becoming available for almost everyone and finding correlations and making predictions is becoming easier and easier. Find ways to do a continuous export of your analytics data to your big data environment and get insights that you never thought of before.

Conclusion

Software quality is even more important in mobile projects than on the web because of the releases through the store, the wide diversity of devices and competition of other apps. Setting up a good continuous delivery process will increase your overall software quality and will enable you to release more frequently and with less risk.
Taking the first steps should be easy. There are lots of guides for setting up continuous integration builds, running automated tests and doing automated deployments to tools, for instance Hockeyapp. On my blog http://mobilefirstcloudfirst.net I've written a couple of guides on how to set up automated builds, tests and distribution to Hockeyapp for your mobile apps using VSTS to get you started with continuous delivery 1.0. Hopefully this article has inspired you to think a bit further than just these basic steps and will help you to evaluate your current development lifecycle with new ideas on how to rethink your testing, releasing and analytics steps to move more to continuous delivery 3.0.

GEERT VAN DER CRUIJSEN
MOBILE LEAD CONSULTANT
XPIRIT

Geert van der Cruijsen is a mobile software architect with years of experience in building applications using Xamarin & .Net technologies for the iOS, Android and Windows platforms. As a Lead consultant at Xpirit, Geert helps Xpirit's customers with defining and improving their mobile strategy, vision and technical implementation regarding mobile application development and mobile ALM. Geert is also a Xamarin Training partner, delivering Xamarin University training to mobile developers. Geert is an active member of the Xamarin and Windows UWP community, has spoken at several conferences and is co-organizer of the Dutch Mobile .Net developer's meetup.

Xamarin University

Disruptors don't follow traditional rules. They deem the impossible possible.

“In the new world, it is not the big fish which eats the small fish. It’s the fast fish which eats the slow fish.” (Klaus Schwab)

In our current digital age we see traditional business processes and priorities dethroned.
If you fail to value speed above all else, your company will fall behind!

Therefore we think ahead, but act now!”

CTO XPIRIT

Extending your Build and Release pipeline

The new build and release engine that ships with Visual Studio Team Services and Team Foundation Server 2015[1] provides a whole new way of configuring your Continuous Integration and Deployment pipeline. A much improved and simplified way.
You may have invested in the build and release features of previous versions, or you may be treading into uncharted (or currently not yet supported) territory for your own applications. That territory isn't a swamp nor is it a desert; it's actually not that hard once you know where to start. Let this article be your guide along the way.

Run existing scripts

The existing build and release library of tasks is already quite extensive and consists of a set of predefined tasks that you can use to author your build and release pipelines. The tasks are divided into five categories: Build, Utility, Test, Package and Deploy. You have total freedom in choosing which ones you want to use and in what order. Most of the tasks come with a simple configuration UI, which allows you to tune their behavior.
If your build process is relatively simple, you will probably not need to extend the process at all. You will only need to pick the right tasks from the library, put them in the right order and configure them.

There are also a number of standard tasks that offer your first extension points:

- The **Build** category contains tasks that allow you to run a number of standard build and package management tools, ranging from MSBuild, NPM, Gradle, Grunt to Ant and Gulp.
- The **Utility** category contains tasks that allow you to run PowerShell and Batch scripts on Windows as well as Shell scripts on other platforms.
- The **Deploy** category contains a number of tasks that allow you to interact with remote machines, including Azure PowerShell, Docker and Chef.

The common element of all of these tasks is the fact that they execute scripts that you've included in your source control repository. This has the advantage that it's easy to version these scripts with your code, but it also introduces a coupling between your build pipeline and the scripts in source control. Given that their contents are not visible when looking at the pipeline, they can also make the build and release process harder to understand.
However, what's nice about these extension points is that you can use the technologies you are already familiar with to extend your build pipeline. It also allows you to port over your existing build pipeline without too much trouble.

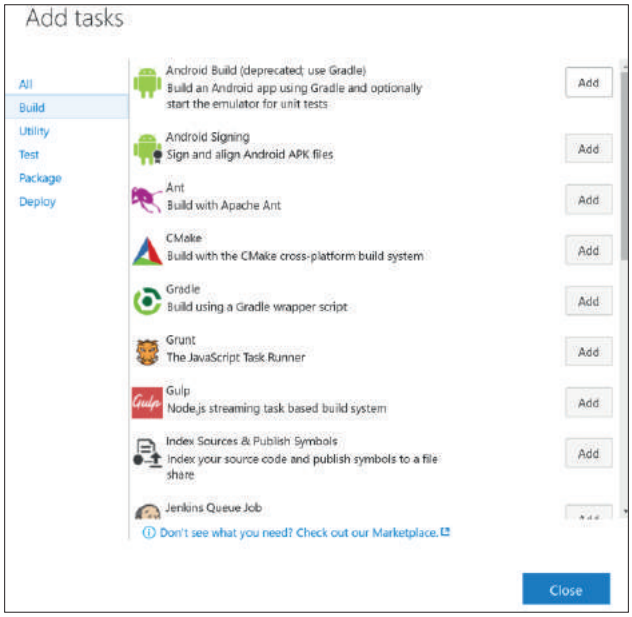


Figure 1

Tip: In order to quickly iterate while writing your script in the context of the Build system, you'll find the Run Inline Power Shell[2] and the Shell++ task very useful. They allow you to run a script defined completely in the task UI. However, after you've debugged your script, it is recommended to check it in and use the existing PowerShell and Shell Script tasks to run them.

Grab an extension

If you don't have an existing script that does what you want, and writing such a script isn't an easy thing to do, you should check out the Visual Studio Market Place to see if there is an existing extension that offers the functionality you're after.

[1] The Release Management bits shipped with TFS 2015 update 2
[2] http://xpir.it/mag3-extendpipeline1
[3] http://xpir.it/mag3-extendpipeline2
[4] http://xpir.it/mag3-extendpipeline3

Extensions are installed into your Visual Studio Team Services account of Team foundation Server (requires 2015 update 2 or higher). Extensions can carry one or more build tasks as well as other types of extensions, such as Dashboard Widgets or 3rd party features.

Tip: You may also find an extension which provides most of the features you need. Since most extensions are released as Open Source, they may provide a starting point to build your own extension, or you may be able to provide the added functionality in the form of a pull request or raise an issue with the developer on their GitHub account.

There are over 100 extensions in the Build and Release category of the Marketplace, so it is likely you will find what you are looking for.

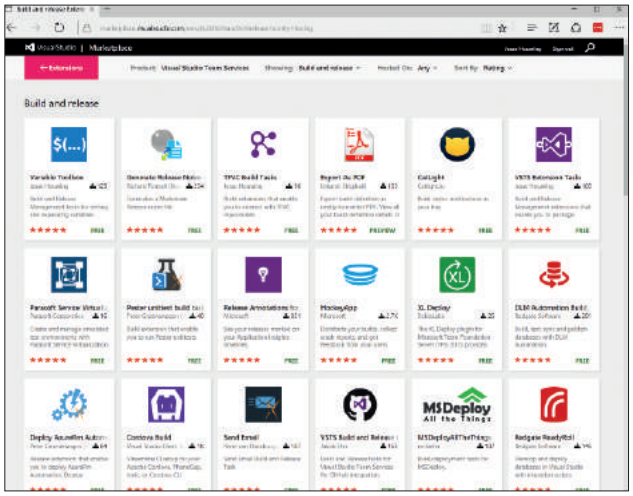


Figure 2

A side note on variables

When you start making your scripts re-usable in different contexts, you will likely want to pass data into your scripts. There is a long list of pre-defined (well-known) variables available which you can reference directly from your existing build scripts through environment variables.

It is highly recommended to use these well-known variables It is considered bad practice to rely on the availability of custom variables, as these will be hard to discover for other people. Instead, consider passing these values as arguments to your scripts.

Wrap your script in a task or create your own

If you can't find a task in the standard library nor an extension in the Marketplace to extend your Build and Release pipeline, you have the option of writing your own task. You may also want to wrap your existing scripts in a task to provide a nice UI user interface to make it easier for others to consume.

Tasks can be written in two languages: TypeScript (which compiles to Javascript) and PowerShell. TypeScript will run on all platforms, whereas PowerShell will only run on the Windows Platform. If you consider sharing your task to the Visual Studio Marketplace it is recommended to use Typescript in order to reach the broadest audience.

From your task you can call out to other technologies, any executable or shell script. It is also possible to call directly into .NET assemblies from the PowerShell host. Communication between your task and the agent's host process is done through specially formatted log messages[5] or through the corresponding methods of the VSTS Task SDK.

The minimal components required for a task are:

- `task.json` – The task manifest which describes the task's input parameters and the main entry point.
- `A .js or .ps1` file – The actual script that is executed as configured in the task.json.

1
```
{
  "id": "3333333-3333-3333-3333-333333333333",
  "name": "PingTask",
  "friendlyName": "Ping",
  "description": "Pings a remote host to see if it is alive.",
  "helpMarkDown": "[More Information](http://www.example.org)",
  "category": "Utility",
  "author": "Jesse Houwing",
  "version": { "Major": 0, "Minor": 1, "Patch": 0 {,
  "minimumAgentVersion": "1.83.0",
  "inputs": [],
  "instanceNameFormat": "Ping $(RemoteHost)",
  "execution": {
    "PowerShell3": {
      "target": "$(currentDirectory)\\myscript.ps1",
      "workingDirectory": "$(Build.SourcesDirectory)"
    },
    "Node": {
      "target": "$(currentDirectory)\\myscript.js",
    }
  }
}
```

[5] http://xpir.it/mag3-extendpipeline4

If you want to interact with the task host - to receive input parameters and read any variables - you will also need to reference the vsts-task-lib[6] matching your chosen script technology.
A build task can package both PowerShell and a Javascript. Depending on the agent's capabilities it will select the appropriate implementation based on the order defined in the task.json.
You can include either PowerShell3 or Node, or both. Be sure to give each task a unique id and to increment the version number each time you publish a new version. The minimal task.json looks like this: (code 1)

There is little documentation on the `Task.json` options and structure; the best reference to date is the vsts-tasks library on GitHub and the repositories of the open-source tasks.

You may encounter a third task execution target called: "Power Shell" (note the lack of a version number). This is the task host that shipped with TFS 2015 and is considered deprecated with the release of TFS 15 and the latest VSTS build agents.

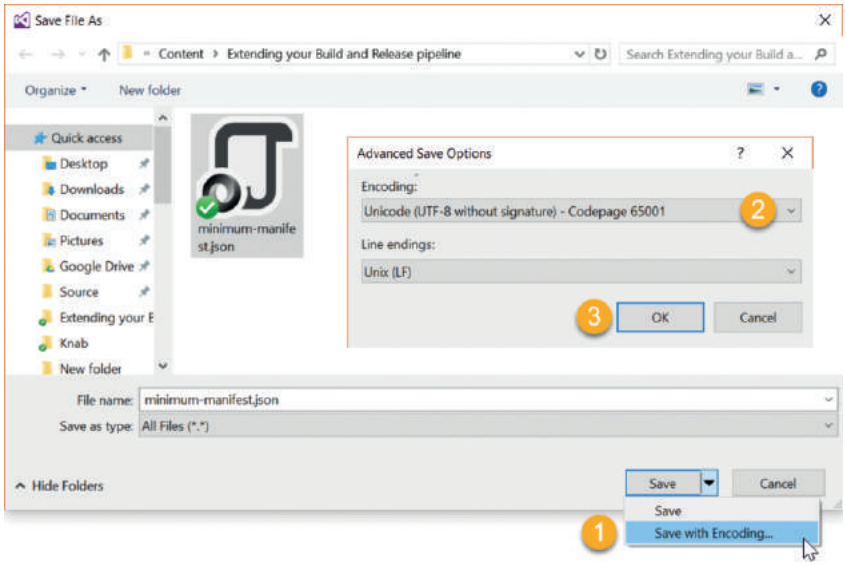


Figure 3

Note: be sure to save the task.json as UFT-8 with no Byte-Order-Mark (Signature), or your task may fail to upload or will not show up correctly: (figure 3)

Put the logic you need in the Javascript or PowerShell file. This small example tries to ping a host and returns the result: (code 2)

Tip: You can use the Cross Platform TFS Command line tool to create a skeleton task using tfx build tasks create.

2
```
$remotehost = www.example.org
[string] $output = (ping $remotehost)

if ($output -match "Pinging")
{
    if ($output -match "\(100\% loss")
    {
        Write-Output "##vso[task.logissue type=error;]Could not reach $remotehost"
        Write-Output "##vso[task.complete result=Failed;]FAILED"
    }
    elseif ($output -match "\(0\% loss")
    {
        Write-Output "##vso[task.complete result=Succeeded;]DONE"
    }
    else
    {
        Write-Output "##vso[task.complete result=SucceededWithIssues;]DONE"
    }
}
else
{
    Write-Output "##vso[task.logissue type=error;]$output"
    Write-Output "##vso[task.complete result=Failed;]FAILED"
}
```

[6] http://xpir.it/mag3-extendpipeline5
[7] http://xpir.it/mag3-extendpipeline6
[7] http://xpir.it/mag3-extendpipeline7

Upload your task to Visual Studio Team Services or TFS using the cross platform command line tools[7]:

```
tfx build tasks upload --task-path
c:\folder\containing\task.json\file
--service-url
http://youraccount.visualstudio.com/
DefaultCollection
```

You will be prompted for a Personal Access Token when you are uploading to Team Services or for your account credentials for TFS (use Fiddler or an NTLM authentication proxy to authenticate against TFS 2015[8]).

The task will then be available for all users of the target account / project collection.

Extend your task with a User Interface

To extend your task with a UI, update the `task.json` and define one or more input elements. Inputs can be of different types, e.g. filePath, string, boolean, picklist, radio or multiLine). The UI does not provide a lot of ways to do validation, so the script included with your task will have to provide for this. To make the remote host configurable in our current task, add a required string-type input: (code 3)

You will need to package the PowerShell or Typescript task SDK with your extension. Since the example uses the PowerShell3 host, you will need to download the VstsTaskSdk module[9] and put it in a subfolder named `ps_modules\VstsTaskSdk`, in alongside the `task.json`.

We can now use the commands provided by the Task SDK[10] to retrieve the input value and to set the outcome of the task: (code 4)

3

```
...
  "version": { "Major": 0, "Minor": 2, "Patch": 0 },
  "inputs": [
    {
      "defaultValue": "www.example.org",
      "helpMarkdown": "Hostname or IP address to ping.",
      "label": "Remote Host",
      "name": "RemoteHost",
      "required": true,
      "type": "string"
    }
  ],
  "instanceNameFormat": "Ping $(RemoteHost)",
...
```

4

```
$remotehost = Get-VstsInput -Name "RemoteHost"
[string] $output = (ping $remotehost)

if ($output -match "Pinging")
{
    if ($output -match "100\% loss")
    {
        Write-VstsTaskError "Could not reach $remotehost"
        Write-VstsSetResult -Result "Failed"
    }
    elseif ($output -match "0\% loss")
    {
        Write-VstsSetResult -Result "Succeeded"
    }
    else
    {
        Write-VstsSetResult -Result "SucceededWithIssues"
    }
}
else
{
    Write-VstsTaskError "$output"
    Write-VstsSetResult -Result "Failed"
}
```

[9] http://xpir.it/mag3-extendpipeline8 and http://xpir.it/mag3-extendpipeline9

[10] http://xpir.it/mag3-extendpipeline10

If your task depends on the presence of third party tools, you should either package them as part of your task or define a demand in your task.json. If you ship a third party component, validate that its license is permits redistribution. When using a demand, your task will signal the consumer to install the third party prerequisites on the build server.

Release your task as an extension

The Visual Studio Marketplace makes it easier to create tasks and to share them with the community. To share your build task as an extension, follow the steps outlined on MSDN .
You can also use the Build and Release Tasks for Team Services Extensions to automate the build and release pipelines of your extensions.

Conclusion

As you can see, the process of writing a simple build task is easy. Although the documentation is currently still evolving, there are enough examples available, by means of the standard tasks as well as tasks shared by the community.

The fact that both the Build and Release functionality depends on the same task library makes investments in custom tasks even more useful.

If you feel your tasks are useful to others, share them to the community by pushing them to GitHub and publishing them as an extension to the Visual Studio Marketplace.
If you find gaps in existing extensions, join the effort by submitting issues, or better yet, submitting a pull request.

Everybody benefits!

Build & Release Extensions published or contributed to by:

Jasper Gilhuis
- Token Comparer

Jesse Houwing
- Build and Release Tasks for Team Services Extensions (contributed)
- Extension Tasks
- TFVC Tasks
- MsBuild Helper
- Variable Toolbox

Pascal Naber
- Azure WebApp Configuration

Peter Groenewegen
- Zip and unzip directory
- Pester Unittest task
- Twitter
- Deploy AzureRm Automation
- Run Inline Powershell & Azure Powershell

René van Osnabrugge
- Send Email

JESSE HOUWING
LEAD CONSULTANT & SCRUM TRAINER XPIRIT

Jesse's passion is to support agile teams along their journey of continuous improvement by helping them tweak their tools, tighten their collaboration, and learn new skills. In the past year he has developed a number of tasks that extend the new build system. Jesse is an active blogger, community contributor to StackOverflow and MSDN, and a Professional Scrum Trainer.

Professional Scrum Trainer Scrum.org

MVP Microsoft® Most Valuable Professional

Technical Debt in your Application Lifecycle

Addictions are a serious problem. Using a survey assessing alcohol usage and personal behaviors can quickly help discover the seriousness of the problem. If only assessing the impact of technical debt was that easy. Addictions are usually ignored until the effects of the addiction are really bad. The same can be applied to technical debt in your application lifecycle. It is there, but often it only surfaces when gaining control is difficult, requiring a lot of work and cost. This article provides guidance on acknowledging technical debt and helps you to regain control.

Acknowledging technical debt

Today, many teams spend a lot of time and effort in optimizing processes of their Application Lifecycle. We often look for ways to improve our continuous integration, automated deployment, testing, architecture, continuous delivery, agile practices and automated provisioning. Many of these topics require technical skills or require organizational stamina to result in change. Technical debt is just as important as the other topics, but it rarely gets the attention it deserves. One of the characteristics of technical debt is that it is spread throughout the application lifecycle. A shortcut built in to the codebase can be perfectly OK at the moment of implementation, but we tend to overlook consequences for the future. In practice, the less visible these choices are, the more they can hurt. Nowadays, effective and well organized teams can ship software towards production in a fast manner. However, effects of shortcuts can easily result in costly production issues.

When teams do have code quality on their radar, they tend to focus on code syntax only. But that only covers the easy parts. Substantial rework or refactoring is often not accounted for, resulting in technical debt being built in. Acknowledging technical debt is the first step, just like admitting the addiction, but how do we convince our product owner and stakeholders that we need to undertake additional efforts for something that currently seems to do the job?

What is technical debt?

Technical debt can be summarized as 'During the planning or execution of a software project, decisions are made to defer necessary work'. Over time this infects the maintainability of the codebase and will require effort to keep resolving issues. Several forms of this debt can exist and are referred to as architectural debt, and design debt. The practices and approaches described in this article are applicable to most forms of technical debt.

Getting in control

While acknowledging is the first step to improving, taking actual measures is the next step. As a development team, we spend time refining our backlog, m focusing mostly on getting the right amount of functional detail into the user story. However, getting user stories right can be challenging.

There are many ways of getting your stories better. There are many techniques available for working on stories; one great example is provided by Gojko Adzic[1]. He has taken a hamburger as a model for breaking down user stories. It helps you to identify tasks and possible other options, providing you with an overview. It enables you to sort these on complexity and effort, allowing you to take a conscious decision on how to approach the story. Using techniques like these allows you to approach stories in a way in which both technical and non-technical people – e.g. a product owner – understand the mechanics of the story.

As stated before, it is of key importance that we acknowledge technical debt. When working out the details of a story you should be able to focus on existing technical debt or places it may occur. A conscious mind is required for this, danger can be just around the corner, for example the lack of domain knowledge or just the rush to get things done.

Quantify technical debt

Identifying and measuring technical debt can be a daunting task. Fortunately, there are tools available to help you achieve this. One of them is SonarQube[2], an open source tool that helps you to identify and measure technical debt. You can integrate it in your builds and get it to automatically generate technical debt measures.

While having this at build time is perfect for many teams, some like to have this feedback even sooner. Tools like SonarLint for

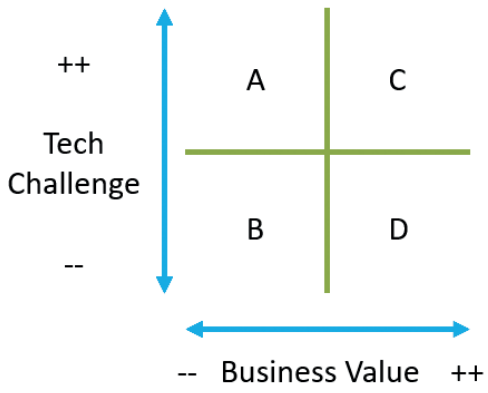
[1] http://xpir.it/mag3-techdept1
[2] http://www.sonarqube.org/

Visual Studio[3] can help developers identify critical issues on-the-fly. Rule sets defined by the team can be enforced immediately to prevent the occurrence of known technical debt issues. Remember, facing the facts is a good thing!
Besides direct feedback during development, feedback based on telemetry in your application is equally important. Insight into what parts are actively used compared to less active parts helps determine the actual need to fix certain technical debt items in these areas. It provides developers as well as business and stakeholders with valuable insights they may not have had previously. Prioritizing on what to do first can now be based on actual data.

Technical debt will be in your application lifecycle. Acknowledging that is important, fiercely trying to prevent any form will get you nowhere. Budget is never an unlimited resource, which means choices need to be made. Technical debt is something that needs to be worked on to prevent it from increasing to unacceptable levels. Although at first sight this usually does not add 'real value' for the product owner or stakeholder, the development teams are aware of its importance.

Making it visible
Having technical debt measured by tools allows you to add significance to it. Making product owners and stakeholders aware of this significance is of great value. To do so, teams need to make sure that their technical issue is well understood and that the team should be able to quantify or compare it to other (mostly functional) items on the product backlog. It is considered good practice to make stories out of technical debt items and provide these with an estimate. Being able to compare stories in your backlog makes it easier to 'sell' it to the business.

The following diagram may help to make technical debt in a particular area understandable and comparable. Every technical debt issue should be plotted in this diagram.



A debt story that holds little business value and requires little technical challenge to resolve it (B), may sound like a quick win, but may not be worth the effort. Technically challenging items with low business value (A) may result in significant time spent on something that has little results. Technically challenging and high business value items (C) sound like a challenge but could easily be worth the effort to the business. Some technical debt may occur in areas where fixing is simple and business value is high (D).

Measuring Effects
Acting on technical debt is important and the metrics provided by tools will allow you to verify whether your efforts are actually helping. Retrospectives can be used to zoom in on how the technical debt is evolving in your product. What kind of measures would be helpful?

A very important metric is your team's velocity. A stable velocity implies a steady flow of business value delivered over time. Handling technical debt in a sustainable way should improve the overall quality of the product. However, spending time on debt is time that is not spent on actual value. Therefore, velocity may drop a bit at first, but could well increase over time while your application is better equipped to stand the test of time. Teams should also be comparing their velocity to business value delivered. Solely looking at velocity or value could be deceiving. A little drop in velocity could very well deliver more value.

Measuring your technical debt and comparing this to your velocity and your business value is crucial to your application lifecycle.

Summary
As a developer you want to write high-quality software with minimal technical debt, resulting in high value being delivered. User stories require enough attention from a technical and non-technical viewpoint. Various techniques are available on getting it right. Everyone involved in the application lifecycle should understand the importance of technical debt management and should put his best effort into minimizing risks and achieving great results!

JASPER GILHUIS
ALM LEAD CONSULTANT
XPIRIT

Jasper combines his deep knowledge of the Microsoft ALM Platform with several years of (agile) process experience to enable long-lasting change in the software delivery process.
He uses his enthusiasm and his genuine interest in people to make a difference at any level of the organization. Jasper provides consultancy, training, and guides workshops in the area of ALM and Scrum.

[3] http://xpir.it/mag3-techdebt2

Doing Testing Right and at the Speed of Light

Most organizations think they are ready for the future when they engage in agile software development practices and ALM. However, if these organizations do not change their traditional, after-the-fact approach towards testing, they will not be able to deliver their desired value at the speed of light. This article describes a different approach towards testing, presents steps to properly introduce testing and test automation, and shows how the Microsoft eco system can be employed to achieve this: integration with all the relevant test tools by standardizing their output formats. We exemplify this by using various concrete examples obtained along the way.

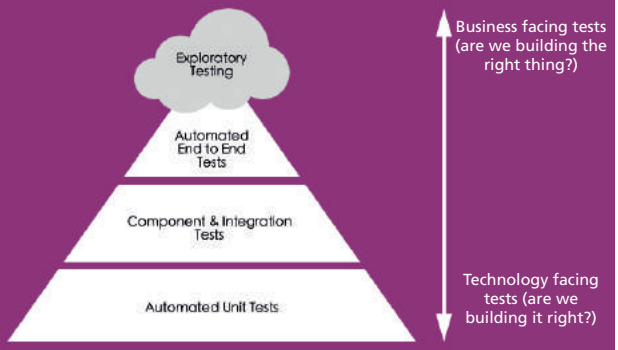
Wake up!

IT organizations that still feature an after-the-fact approach to testing need to radically change their software testing approach if they are to stay in business. New ideas, Minimum Viable Products, apps, and novel integrations are what businesses should be built on – without any compromise on quality. In an era in which acceleration of the delivery cycles is necessary to shorten the time to market, traditional testing is one of the last bastions that needs to be conquered. Testing these ideas, MVPs, apps, and integrations as quickly and often as possible becomes a mere necessity: the sooner we can invalidate assumptions, the sooner organizations can come up with the right, game-changing innovations. Organizations simply cannot survive without it.

Testing is a unique discipline which requires a clear, quality-oriented mindset. Traditionally, test activities are performed by dedicated teams of test professionals. In turn, these test professionals will commence testing the software only once it is built. As such, test activities typically are performed after a sprint's end. Often they are labor-intensive, and in the case of findings, rework is injected into subsequent sprints. This negatively impacts the velocity of the project and can ultimately lead to the project's grinding halt: what is built in two weeks' time, often takes two months (or more) to be put into production. This needs to stop. Organizations should be able to deliver high-quality software to production as often as possible and as quickly as possible, so there really is no use in separate testing activities with late feedback cycles.

In short, it's time to do it differently. Testing and checking should commence before the production software is built and should rely on close collaboration between the whole team. It's a rather clear adagio: "we specify, develop, and test our functionality together, and we deliver software that is production-ready as soon as it's ready."

Mike Cohn's agile test automation pyramid



The pyramid defines several layers on which tests can be automated. Unit tests are the lowest-level tests, counterparts of designated source code functions. Automated end to end tests should (only) focus on end-to-end integration scenarios rather than testing individual functionality. Likewise, Component & integration tests serve to test functionality of the application, typically exposed through services, without having to rely on the user interface of the application.

One should strive for automating at the lowest level possible ('Unit tests'), since this enables quick feedback cycles, detailed feedback information ("the bug occurs at line 42 of the Customer Service module"), and the corresponding technical ecosystem between test code and production code.

A plea for a straightforward test strategy

Some of the most important questions related to testing include: "What do we want to test? And why?". If the rationale for testing is clearly defined, different test types can be logically derived; a focus on integration testing, performance testing, or resilience testing. As a next step, we define a current and a desired state on how these tests are actually performed (automated or not), where the testing takes place (see sidebar on the agile test automation pyramid), what technologies are used and by whom (the test-oriented team member, or development-oriented team member).

Start right, and start now!

Every organization can start with moving towards a sound agile testing and test automation strategy. Organizations that use the Microsoft stack can immediately implement the right tooling in order to speed up their feedback cycle.

Welcome to Visual Studio Team Services and Team Foundation Server (TFS) cum suis. New required functionality (user stories) is defined within TFS as Product Backlog Items (see Figure 1), each of them broken down into one or more tasks. Proper operation of the functionality needs thorough testing. As such, one or more test cases can be linked to a product backlog item. The test cases are managed within Microsoft Test Manager (MTM). Within MTM, these test cases do not pose any restrictions as to how the test case is executed (manual or automated) and with regard to the technical ecosystem; the test case merely serves as a description of the test activities.

Traditionally, testing using a TFS-/MTM-based setup as outlined above occurs by running the application and (often) manually performing the steps of the test cases that are attached to a specific work item. This is a suboptimal approach because of the distance between testing and coding – both temporal distance (first we need to develop code, only then can we perform the testing activities) and technical distance (test (plan)s and test results reside in a separate system, namely Microsoft Test Manager). Furthermore, testing occurs within big chunks (complete sprint results as the unit of test) rather than intermediate feedback on partly-implemented Product Backlog Items. This makes it more difficult to address test findings (find the needle in the haystack), which again delays the feedback cycle.

This approach, which is often encountered in many organizations, requires a significant speed-up in order to achieve the true merits of continuous delivery (integration and testing). So, how can we use readily available solutions to perform proper testing as quickly and as often as possible? Let's take the next steps to reap the full benefits.

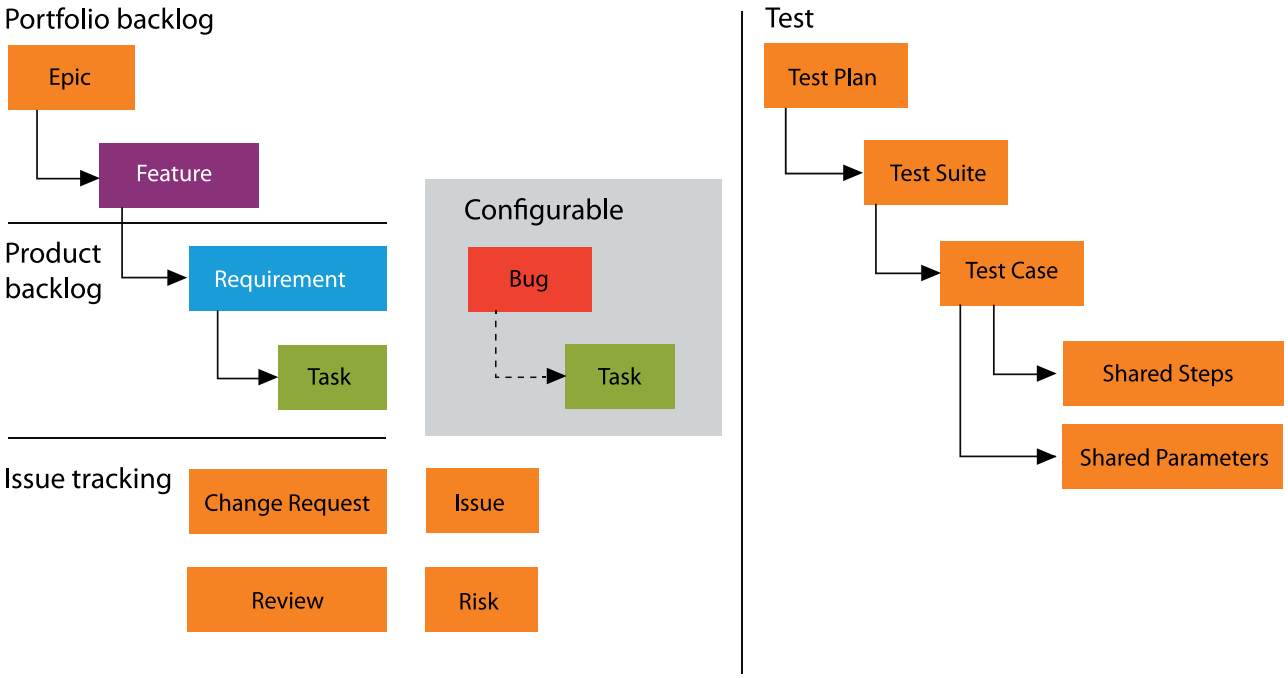


Figure 1 Work Item definitions in Team Foundation Server

Taking the next steps

1. Automate your tests

Automate existing test cases within TFS, or develop integration adapters and attach some assembly to the test cases in Microsoft Test Manager. This can be used to link Protractor (or similar) test code for an AngularJS front end (cf. Xpirit Magazine #2, p. 40-46). Currently, only MSTest-based test cases can be associated, and increase of this coverage may not be expected. Microsoft díd announce a new capability to link test results that were generated during a build run with any arbitrary adapter, to requirements, for instance user stories[1]. Linking test results to test cases seems like an obvious next step – this feature is requested.

2. Truly bring your tests to the code

We can take this even one step further. Why not make the test case itself reside within the source code – right next to the production code? This enables tests to be run as soon as code is developed – possibly even before: Gherkin-style specifications (given – when – then) can be used to enable the business to speed up refinements and clearly describe the acceptance criteria for each product backlog item. This notation also helps to achieve common understanding between all stakeholders of the required functionality. Below, we share an example of shopping cart functionality that verifies the total article count.

```
Given I am logged in
When I add a product to my shopping cart
Then the cart total is updated
```

The Gherkin specifications are transferred to source code once development commences, and glue code is developed to link the test case and code to the production code. Both the production code and test code are maintained by the team. All code is part of the same source code repository, nicely placed together. (code 1)

As soon as parts of the functionality are tested and committed, all tests can be executed automatically. Issues are spotted as quickly as possible, because the right tests are run as part of your pipeline at each commit. As a result, lengthy analysis steps to find the issues are reduced to quickly spotting the issue, actually fixing it, and observing a green test.

3. Test and develop as one in an integrated pipeline

Within the VSTS/TFS Build and release, we can construct a pipeline where all the tests reside. Regardless of the technological choices for our applications' platforms (Angular, iOS, C#, Java), we enable this pipeline to drive all development and test activities.
This allows us to write test code in the same technical context as the production code (i.e. we use Protractor for Angular tests, and nUnit for C# unit tests), because we simply integrate the test results based on their standardized output formats (such as xUnit, jUnit or NUnit); this prevents the community from having to write (and maintain) adapters. Even more importantly, using standardized test output formats allows us to integrate all automated tests in a heterogeneous and partially non-Microsoft ecosystem. There is an open standards world out there whose benefits can readily be reaped.

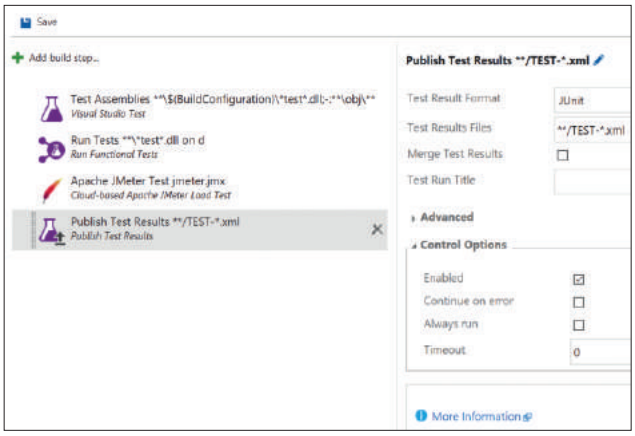


Figure 2 Running tests within a pipeline

Implementing your test strategy in the pipeline

The build pipeline acts as continuous integration and allows us to chain all kinds of tasks so that production-ready software is

1

```
[When(@"I add a product to my shopping cart")]
public void WhenIAddAProductToMyShoppingCart()
{
    home.addProduct(0);
}

[Then(@"the cart total is updated")]
public void ThenTheCartTotalsUpdated()
{
    Assert.AreNotEqual(initialProductsInCart, home.CartQuantity());
}

public HomePage addProduct(int index)
{
    _driver.FindElements(By.CssSelector(".btn-primary"))[index].Click();
    return this;
}
```

[1] Please see the following blogpost for more details: http://xpir.it/mag3-testing1

achieved. This includes development and test tasks. The following section describes how some of the types of tests from the agile test automation pyramid (see sidebar) can be implemented within a pipeline concept.

- Unit tests are performed during development, prior to committing changes and in the build pipeline, prior to building the component consisting of the various units.
- Integration tests test specific integrations between separate components. These tests can happen internally and externally. Internal integration tests may consist of test routing within a service. External integration tests may test compatibility with data structures in code and database schema, or how an application handles unavailability of a database.
- Business-facing component tests test the user interaction with one or more parts of the system.

Implementation (simplified) code 2

Test type	Description	Current state	End state	Developed in	Responsibility	Mainly performed by
Unit	Tests the smallest pieces of testable code - C# (RESTful services) - Typescript (front-end) - RDBMS (if needed)	Code	Code	C#, Typescript	Team	Developers
Integration	Test the interaction between components (service layer ⇔ database layer, or front-end layer ⇔ service layer)	N/A (no tests)	Code	C#	Team	Developers
Component	Tests independent components of a larger system, e.g.: - API (RESTful service) - The view (front-end)	N/A	Code (BDD for view)	C# OR Typescript	Team	Testers, Developers
End-to-End	Test whether an entire integrated system meets its business goal	Manual (using MTM)	MTM+BDD	C#, OR Typescript	Team	Testers
Exploratory	Manual testing involving continuous learning based on application feedback	Manual (using MTM)	MTM	Manual test	Team	Testers

Table 1: Overview of test types, where tests are implemented (current state and end state), the programming language, and responsibilities.

2

```
describe('shopping cart filter', function(): void {
    it('finds on partial cart ref', function() {
        var result: any[];
        result = $filter('cartFilter')(cartItems, 'World of Warcraft');
        expect(result.length).toBe(1);
    });
});
```

These tests can also be nicely specified using the earlier-described Gherkin format. If we develop an AngularJS-based front end, we can describe the front-end tests in JavaScript using Protractor, and we simply consume the results and integrate them in TFS. Suppose we want to run cross-browser tests ("does our software function properly in various device/OS/browser combinations?"). Then we can also use a Selenium grid to spawn the different combinations, which means we are able to interpret the results within our pipeline.

The test results from each of the test tasks in the build pipeline are examined to determine whether the software is of sufficient quality to promote to a next phase. Only software that is properly unit-tested can be offered for component-testing or integration-testing, allowing these tests to focus entirely on the component or the integration between components, rather than on the low-level functionality. We should now have that one covered. As such, the build pipeline is a clear implementation of the above-mentioned agile test automation pyramid and test strategy. Table 1 shows a specific example of the test strategy, namely the implementation of the various test types. By including a current (actual) and end state (desired), test improvements can be managed and measured promptly.

If we have a pipeline using TFS and VSTS Build that runs all tests for the various test forms as quickly and as often as possible, we can constantly interpret the results and make the right go-live decisions. Possibly, this can be split across multiple pipelines: a "build pipeline" for unit tests and a "release pipeline" for end-to-end tests. Ultimately, we know that we've built the right product right.

Summary

Organizations should place their testing activities at the center of their development efforts. Only then can development teams obtain the feedback they deserve and need to deliver high-quality software. VSTS/TFS Build and Release enables organizations to incorporate the automated tests to cover the required test forms. What's more, it allows for seamless integration with test tools from other ecosystems by standardizing test output formats. This allows all technologies to be injected into the pipeline, which offers new possibilities to test apps and speed up time to market.

ERIK SWETS
AGILE TEST CONSULTANT
AT XEBIA

Erik combines profound technical knowledge with analytical and communication skills to help IT organizations change the way they organize, approach, and perform testing activities. He is experienced in setting up continuous delivery initiatives and implementing test automation as one of the most crucial parts of his customers' software development activities.

VIKTOR CLERC
UNIT MANAGER TEST AUTOMATION
AT XEBIA

Viktor is a strong customer-focused IT manager with a keen eye on delivery quality. Using modern tools and techniques, he strives to dramatically reduce his customers' time to market and improve the quality feedback cycle to all relevant stakeholders.

Containers on the Microsoft platform: the full picture

Containerization as a new application paradigm

It is hard to miss the emergence of container technology and its effect on application development and architectures. Containers as lightweight hosts for small applications are quickly associated with a micro-services architecture (MSA). Applications based on micro-services are split into small autonomous systems instead of monoliths. Small services ask for light-weight hosting where creating a multitude of hosted services is simple and cheap. This approach benefits greatly from the high density of the services on host machines, and containers fit nicely into these types of applications. This is why the reinvented architectures around micro-services and the new hosting models have driven the popularity of containerization of modern applications.

Containers 101

At first sight, the concept of containers appears rather abstract. Quite often a comparison is made with virtual machines. Although there are some similarities, it is also a dangerous comparison. Virtualization is not the same as containerization and virtual machines are not containers.

Containers are essentially virtual fences created to perform isolation of shared system resources and processes of a host system. The processes in the host and within the containers all exist at the host level. Each container thinks it has its own unique set of system resources assigned to it. A container is oblivious to anything outside of its fence; it cannot see anything outside of itself and any access to the container's resources does not affect other containers or their content. The net result is that each container is its own silo and can be thought of as a way of creating a high density of sub-computers within a host system.

This is perhaps the reason why containers appear to be similar to virtual machines. The isolation is definitely a recognizable attribute, but the key difference is that a container runs directly in the host system and nothing is virtualized, only isolated. Consequently, the containers run their processes in the same operating system that the host uses. A Linux host system will have Linux-based application processes running and the resources behave as in Linux, such as /dev/root. A Windows host will be able to create Windows based containers running .NET and Win32 applications utilizing Windows resources, such as the C:\ drive and the registry. This is an important aspect of containers: container and host OS are always one and the same. Unless virtualization comes into play.

Each container is its own silo and can be thought of as a way of creating a high density of sub-computers within a host system

Docker as an industry standard

Container support has been a feature in Linux and Unix operating systems since the beginning. The popularization of container technology and container-based application architectures has started with the advent of Docker, both a company and tooling with the same name. Before Docker, the challenge involved the semantic differences of the container APIs inside the operating systems and the way that applications are deployed to containers. Docker standardized the methods of creation and interaction with containers. Docker offered a uniform API and Command-Line Interface (CLI) for controlling containers. It allowed a user and application to start and stop container instances. It also defined a standard for building container images, which contain (pun intended) resources and settings inside a newly created container. Container images are packages that are used to deploy your application assets for use in containers. Docker envisions container images to consist of layers. Each image is based on a parent image, going back to a (nearly-empty) base image, and only has the changes from the parent. This approach provides a highly efficient way to build container images. However, it is important to remember that the root parent image is always intended for the particular operating system it was created for.

The popularity of Docker has made it the de-facto standard for working with containers across a variety of platforms. Docker started in the open source Linux eco-system, intended for and targeted to Linux-based applications, but has expanded to Windows.

Docker Toolbox

The first available option to run containers on Windows-based host systems is to use 'Docker Toolbox'. It offers Linux-based containers and images on Windows. This seems like a contradiction from the previous remark that host and container OS are always the same. Docker Toolbox uses Oracle VirtualBox to create

a virtualized host system running a Linux distribution, in which the container instances are hosted. Admittedly, it does sound like a scene from the movie Inception. Until then, Windows itself was not capable of hosting containers, so Linux had to be the host operating system. Virtualization helped out by offering Linux container technology to Windows.

Docker for Windows

The second offering from Docker Inc. for the Windows platform was 'Docker for Windows'. In principle it is a natural progression from Docker Toolbox. The main difference is that it uses Windows Hyper-V technology as the virtualization layer, instead of the aforementioned VirtualBox. This brings Docker much closer to the Windows platform, but still leverages a Linux-based virtualized host to create containers.

Windows Server containers

Windows Server 2016 (Technology Preview 3 and onwards) was the first Windows operating system that offered native Windows containers. Put differently, Windows Server 2016 is capable of creating container isolation as part of the operating system. Like before, this implies that the containers are Windows-based and run directly in Windows, sharing resources and processes from the Windows Server host.

Microsoft has also adopted Docker as their container interaction API. A PowerShell module and the CLI are available, so you can choose your preferred interactive terminal, and write PowerShell scripts or batch files to automate simple Docker management.

Hyper-V containers

The Windows Server 2016 operating system has a unique feature with regard to containers. It knows how to combine virtualization and containers to provide an even higher level of isolation.

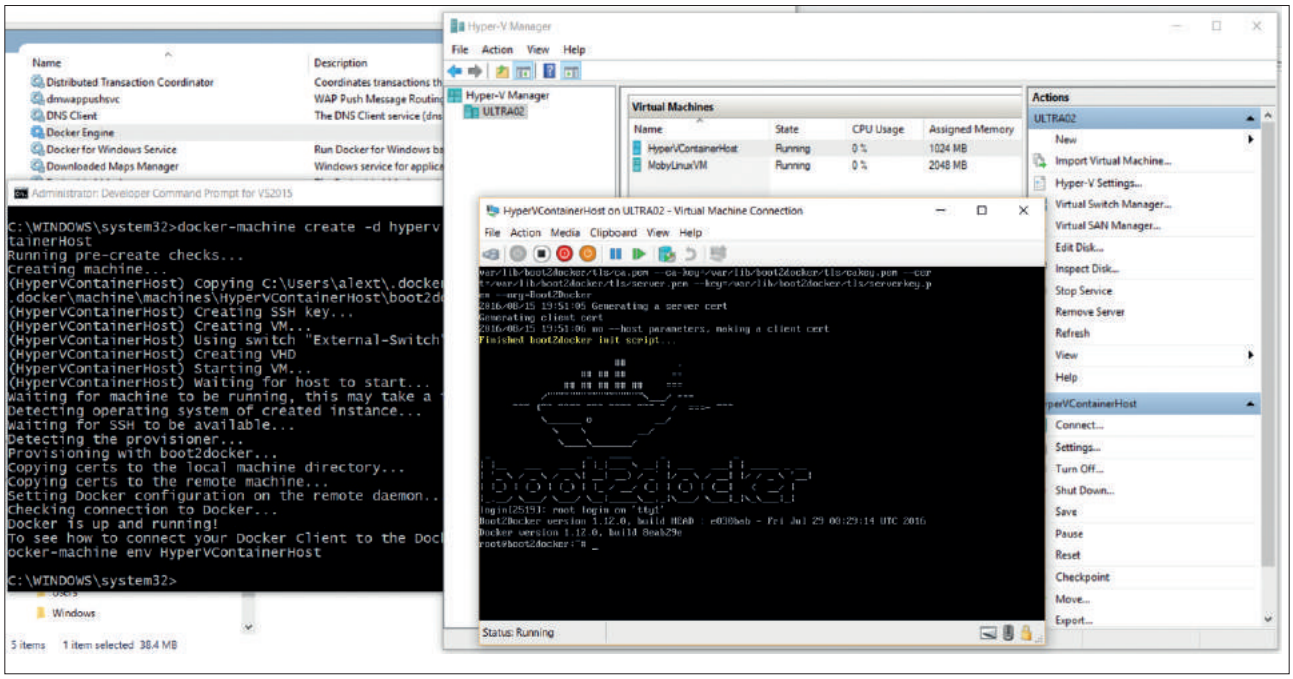


Figure 1: Docker for Windows using Hyper-V to run a Linux container host

```
C:\WINDOWS\system32>docker version
Client:
 Version:      1.12.0
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   8eab29e
 Built:        Thu Jul 28 21:15:28 2016
 OS/Arch:      windows/amd64

Server:
 Version:      1.12.0
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   8eab29e
 Built:        Thu Jul 28 21:15:28 2016
 OS/Arch:      linux/amd64
```

Figure 2: Docker Toolbox shows a Windows client running a Linux-based host

```
C:\WINDOWS\system32>docker version
Client:
 Version:      1.12.0
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   8eab29e
 Built:        Thu Jul 28 21:15:28 2016
 OS/Arch:      windows/amd64

Server:
 Version:      1.12.0
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   8eab29e
 Built:        Thu Jul 28 23:54:00 2016
 OS/Arch:      windows/amd64
```

Figure 3: Windows Server containers shows a Windows client running a Windows-based host

Container isolation is a good thing, but there is a potential risk of applications breaking out of their container and accessing the host system or the insides of another container. To mitigate this risk in a hostile multi-tenant situation or when trying to regulate workloads, you might require additional isolation. Windows offers the option to start a Hyper-V virtualized host with a minimal Windows OS optimized to run containers. These containers are called Hyper-V containers, in which a high degree of isolation is achieved at the expense of a small performance impact.

Two special version of Windows Server 2016 were created as an operating system for Windows Server containers: Windows Server Core and Nano Server. These trimmed-down versions of the full operating system discard unnecessary features in order to provide a fast, lean and mean container host basis.
Container images created for Windows Server container also run in Hyper-V containers and vice versa. So, you can decide the level of isolation as an afterthought instead of having to decide upfront. The same does not hold true for container images used with Docker for Windows and Windows Server containers. The reason for this is that the former utilizes Linux based container images, where the images for Windows Server are Windows based.
Recently, the Hyper-V container feature made its way to Windows 10 Professional and above, giving Windows containers to developers' machines.

Containers in Azure

The Azure platform of Microsoft brings many of the container options together in a number of Infrastructure-as-a-Service (IaaS) offerings. Obviously, with Windows Server 2016 virtual machines in Azure, you can leverage both Windows Server and Hyper-V containers. Additionally, virtual machines running a Linux distribution such as Debian or RedHat, offer Linux-based containers. Both options are nicely integrated into the Microsoft developer experience. Docker container hosts running in Azure virtual machines can be accessed as if they are local on your machine. Microsoft has provided Docker drivers to connect a local Docker client to the Docker daemons and engines running in Azure-hosted machines.

Azure also has a complete production scale container hosting offering in Azure Container Services (ACS). Essentially, ACS provides a cluster of virtual machines in a scale set for containers to be hosted on. ACS uses Docker Swarm or DC/OS under its covers in order to provide a container cluster management, monitoring and resource governance platform for the set of virtual machines. When you create the ACS cluster, you choose either Docker Swarm or DC/OS.

Docker Swarm combines multiple Docker container hosts into a virtual single host for hosting containers. DC/OS builds upon many open source based tools and frameworks, such as Apache Mesos and Marathon. It keeps track of the available resources on all virtual machines in a cluster, it can build and deploy Docker container images, and control and monitor the running container instances.

ACS takes care of installing and configuring typical small and large deployments of Docker Swarm or DC/OS clusters for test and production scenarios. It is available from the Azure Portal after a few clicks, and it offers more functionality than simple IaaS, but it does require you to keep and maintain the virtual machines in the cluster yourself. That brings it rather close to the offering of a Platform-as-a-Service.

Development with Visual Studio and containers

Because Microsoft has chosen Docker as its standard for containers, you will encounter Docker tooling throughout the family of Visual Studio products.
Visual Studio 2015 Update 3 ties into the developer workflow for creating and hosting applications in containers after you have installed the Docker Tools for Visual Studio. The tooling allows a developer to easily deploy a .NET-based application to a Docker container. To do so, it will build a Docker image that is uploaded to an existing or newly created Docker container host which can be local or remote. Images are built from a Docker file describing the alterations from a base image. The tool Docker Compose facilitates in creating Docker images and environments, and is the underlying mechanism leveraged by PowerShell scripts.

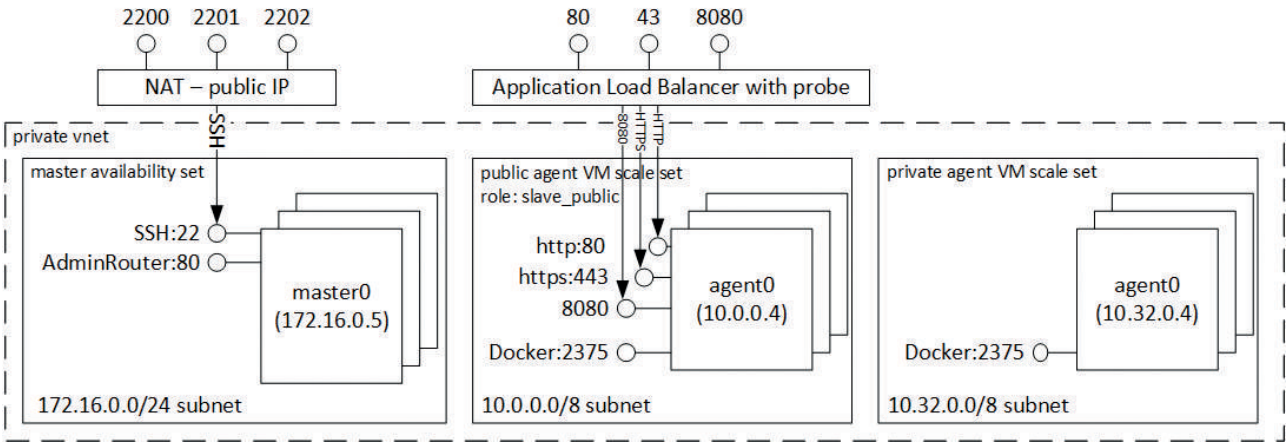


Figure 4: Azure ACS physical architecture provisioned for a DC/OS cluster

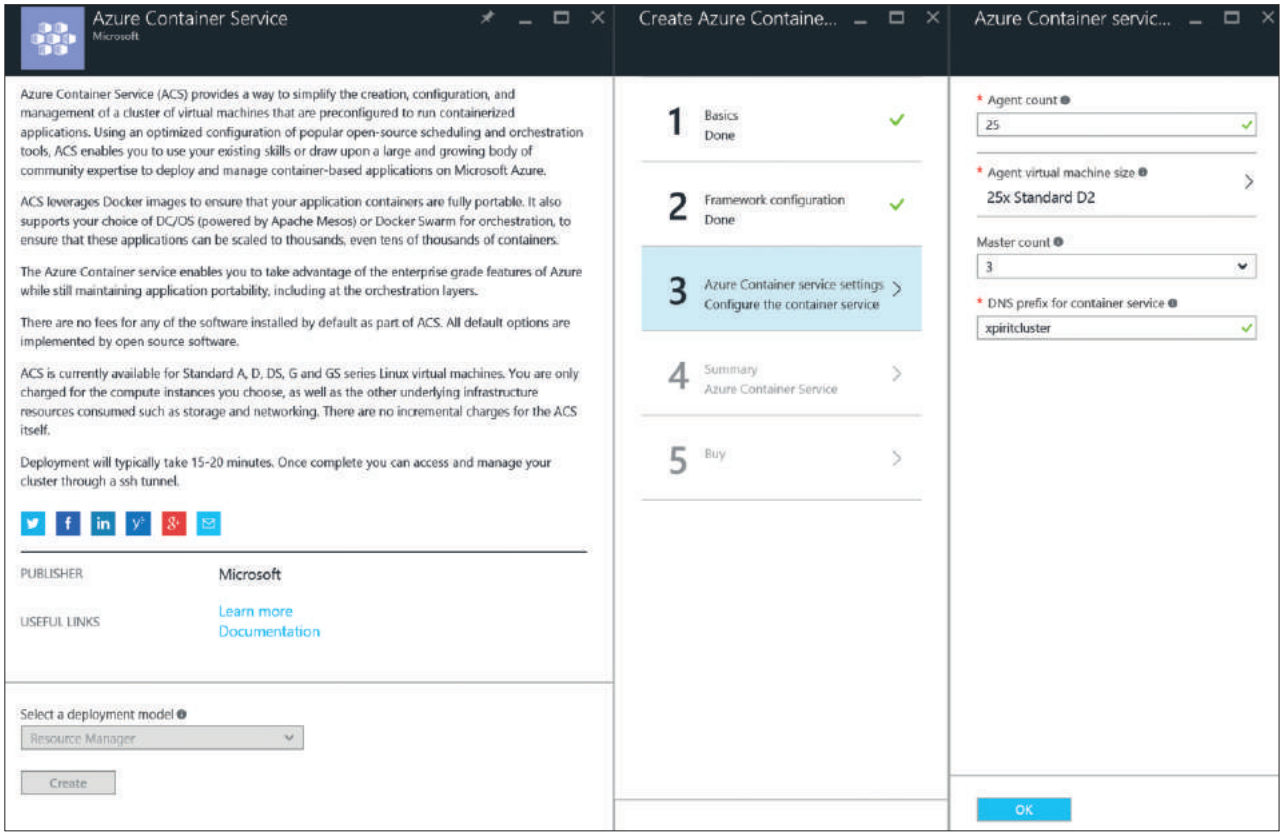


Figure 5: Provisioning an ACS cluster from the Azure Portal

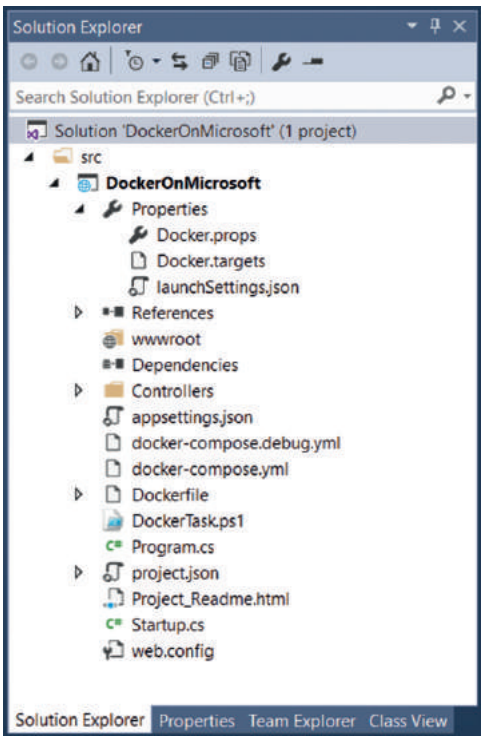


Figure 6

Visual Studio Team Services (VSTS) takes the Docker container workflow a step further and provides continuous deployment and release capabilities. The Docker Integration tooling will integrate Docker in agile and DevOps workflows with easy and transparent management and distribution of the Docker images it creates. The tooling can be found in the VSTS Marketplace and, once installed, will add additional build and release tasks and service endpoints. The service endpoints connect to a Docker Registry and a Docker container host from VSTS. The 'Docker' task allows you to build, push and run a Docker container image, or run a Docker command. The other task 'Docker Compose' can use a docker-compose.yml file to run Docker Compose-defined commands.

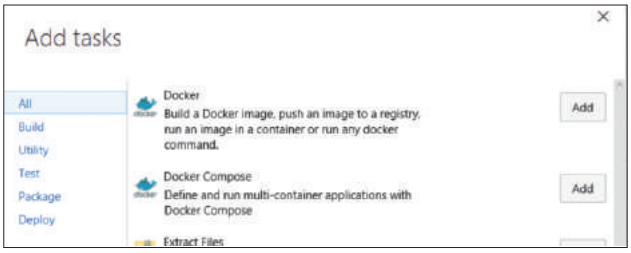


Figure 7

The combination of these tasks allows you to set up a build and release pipeline on your platform of choice. A pipeline that builds a Docker image must run on a VSTS build agent hosted in the same operating system as the one targeted by the container image.

In other words: a Linux-based container image must be run on a Linux-hosted VSTS build agent, and likewise for Windows-based images and build agents.

.NET Core and cross-platform containers

The two main .NET platforms of this moment are .NET 4.6.2 and .NET Core 1.0. The programming experience for both is practically the same with corresponding .NET Frameworks and the languages C#, Visual Basic and F#. The full .NET Framework can only be installed on Windows machines, as it is limited by and dependent on certain Windows features. This implies that applications that are built for .NET 4.5 can only be run or hosted on Windows machine and Windows containers. Windows Server and Hyper-V containers are an excellent choice for hosting .NET applications.

.NET Core 1.0 is a reimplementation of the .NET Framework and its runtimes, and is designed to be light-weight, modular and most importantly, cross-platform. Even though .NET Core 1.0 is not on par in terms of features with the full .NET Framework 4.5+, it is a very attractive option in your choice of application framework. It offers a reasonably smooth transition of your application from Windows to Linux or the other way around. Admittedly, .NET Core is also targeted for OSX, but this is more geared towards developer station scenarios. Developers can build and run their .NET Core applications natively on OSX, but will probably need either Windows or Linux to host the application in a production environment.

Choosing your container strategy and platform

The abundance of choice makes it an all but trivial task to choose your strategy in container technology and the hosting operating system within the Microsoft platform. The following guidance might help in deciding.

Start by realizing that the overall interaction with containers, regardless of their environment, is through the Docker tools CLI and Docker Compose for both Windows and Linux. The choice of PowerShell is more aligned with the Windows platform and is a good alternative if Windows is a given and there is already an investment in PowerShell scripts.

The choice for the operating system is largely determined by the framework that the application (or subsystem) uses. The application can be just about anything: a Web application, Web API, service or command-line tool or otherwise. If these are created for a Linux-based system, you must choose one of the Linux distributions as your container host. For applications built on .NET 4.5 and higher you must choose Windows Server or Hyper-V containers. If you want to develop for multiple types of container hosts (Windows and Linux) then .NET Core offers the flexibility required.

On your development machine it doesn't really matter what you choose and it almost comes down to personal preference. You have most options when you run Windows 10 and use Visual Studio 2015. Docker Toolbox requires Hyper-V to be turned off and is the oldest tooling and might be limiting in its reach. Docker for Windows and Hyper-V containers utilize Hyper-V and are available side by side on Windows 10. If you are able to host any virtual machines in Azure and have access to them during development, you can choose whatever you like. Just remember that you need a separate image for each operating system.

Looking towards hosting in a production environment you will need some higher level governance of all containers and resources. Azure Container Services is currently the only offering available in Azure and its containers are always Linux-based. This narrows your options to targeting your application to either (open source) frameworks for Linux, Mono or .NET Core. Both targets will be able to be deployed as Linux-based container images. .NET Core or Mono seems the most obvious choice for a .NET developer.

At this point in time, Windows Server and Hyper-V containers are best suited for .NET and Win32 applications and non-production scenarios. However, as soon as the container monitoring and governance tooling sets start supporting Windows-based containers, you will have to re-evaluate your choice. The enterprise-grade stability and support for the Windows Server platform is an important factor to take into account.

Summary

The Microsoft platform has plenty to offer for creating and hosting containers. Whether these containers need to run on Windows or Linux, Microsoft can be your platform of choice. Microsoft Azure, Windows Server 2016 and Windows 10 can host all types of containers. Azure also offers Azure Container Services for production scenarios. Docker tooling, the Visual Studio development environment and Visual Studio Team Services allow developers to adopt agile and DevOps workflows, resulting in continuous delivery and release pipelines for container-hosted applications. In short: Microsoft is a one-stop shop when it comes to creating and hosting container-based applications and architectures.

ALEX THISSEN
CLOUD LEAD CONSULTANT
XPIRIT

Alex helps companies build web applications and back-end solutions using Microsoft technologies and frameworks. He helps migrate existing architectures to modern standards and designs and cloud solutions running on platforms such as Azure. Alex cares about security and informs organizations and development teams about secure coding and best practices.

Microsoft® MVP Most Valuable Professional

Conquer the world with Azure Machine Learning

What a great idea! You've decided to convert your brick and mortar store into an online supermarket. You'll create intelligent business processes that will make shopping easy. They will allow you to run the web shop efficiently! You'll make Amazon and Ali Baba jealous. Let's investigate how using Azure Machine Learning can help you.

Creatures of habit

Most people regularly buy the same items. For example, imagine a customer who regularly buys some cheese. It would be nice if the system would advise him to put it in his shopping cart. But he probably won't need cheese during each visit to the shop, only when he has run out. So the system should be smart enough to figure out if he needs it. If you repeat this process for all of his regular products, you can produce an automatically generated grocery list. Which leads to one-click shopping! Now how's that for a time saver?

Out of cheese?

Azure Machine Learning (ML) provides a way to answer this question. First you need to identify the type of problem you need to solve. Your problem has two possible outcomes: either your customer will buy cheese, or he won't. Because the possible outcomes are limited to just two categories here, this problem describes what is called a Binary Classification. Fortunately Azure

ML provides some great out-of-the-box classification algorithms to solve these kinds of problems. You use a binary classification algorithm to make predictions based on past data. For instance, the fact that a certain customer buys cheese every week will stand out as a pattern. Azure ML can detect this pattern and predict whether the customer will buy some cheese when he's in your web shop.

Been there, done that

So far we've determined that it's likely that your customer buys cheese regularly. He'll probably get bored eating the same kind over and over again. Wouldn't it be nice if your system could recommend some excellent alternative brands?

What else have you got?

Azure ML also has a way to answer this question. Again you start by identifying the type of problem you need to solve. This problem has many outcomes; there are many types of cheese your customer might like. The goal is to select just one or two items he'll like best and recommend those. Finding similar products in a catalog can be done using a Clustering algorithm. You can get an answer to the question by looking at possible ratings of previous purchases by that user, and also by seeing what similar customers (same age, gender, etc.) prefer. The Matchbox Recommender algorithm uses both approaches, making it an excellent choice to use in your Azure ML model.

Azure Machine Learning

Azure ML is a Microsoft Cloud platform that offers Machine Learning as a service. This is all about finding patterns in existing data and using them to predict future events. These predictions can make your software smarter.

Stuck in traffic

Let's pivot from the customers into logistics. You can only make a good profit if you don't waste money.

Your company should always deliver orders in the most efficient manner. Predicting heavy traffic can help you do that. So when there is a great deal of traffic on the highway, you'll want to take an alternative route. You can make an informed decision based on traffic information combined with intelligent processing.

What's the best route?

In order to compute the optimal delivery route, you'll need to determine the delay if you take the highway. Again we first need to identify the type of problem we need to solve. The delay itself can be any value (minutes, hours), so classification won't help us here. Predicting such a numeric requires a Regression algorithm. You can use your knowledge about past deliveries, combined with real-time traffic data as inputs. The delay can be compared to the extra time it will take to use alternative routes.

Algorithms

To create predictive models,
Machine Learning uses various types of algorithms.

Some examples are:

- Anomaly detection - For finding unusual data points
- Clustering- To discover structure
- Regression- Prediction of values
- Classification - Prediction of categories

Sold out

Efficient planning of inventory will help reduce waste. You don't want to keep too many perishable goods in stock. Everything past its best-before date will become waste. You need to know how many items of each product you're likely to sell tomorrow. In order to be able to do this, you'll need to have information on what you have sold in the past. Based on information from products that were sold in the past, your system can predict what will be sold tomorrow. The more information you put into the model, the more accurate the predictions will become. For instance, if your data set contains the date at which ice cream was sold, Azure ML will be able to determine that more stock is needed in summer. It would be helpful if you were notified about possible mistakes while ordering new stock. So when you are about to order only ten boxes of ice cream instead of the required 100, the system should warn you about it.

Data sets

Azure ML prediction models can only work well if plenty of accurate information about the past is available.

I can't let you do that Dave

Predicting the quantity of ice cream that will be sold also requires a Regression algorithm. Using information from the registers as input, your system will be able to make accurate predictions about future sales. An algorithm that detects instances of you breaking a pattern is an Anomaly Detection algorithm. Azure ML provides two of them. You can use them to monitor planned orders and detect possible user errors.

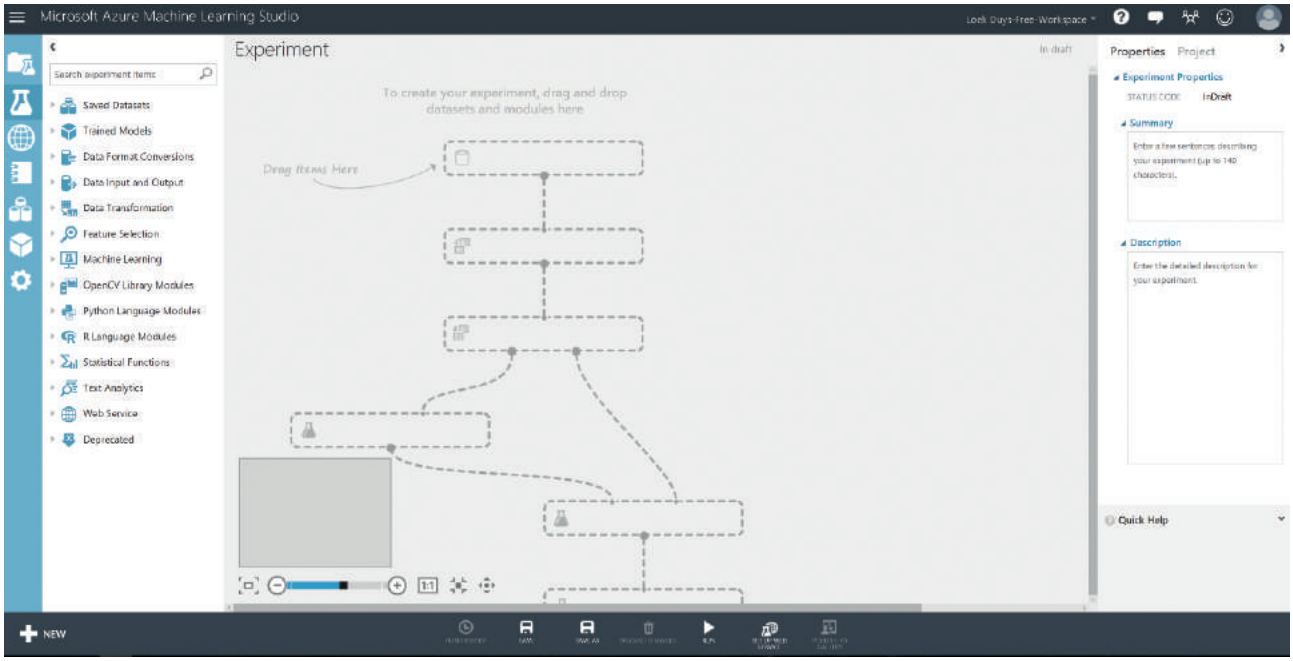


Figure 1

How to get started

If you want to try using Azure ML, you can go to the portal at https://studio.azureml.net and sign up for a free trial. In the portal, you start by creating a workspace. In that workspace you add projects and experiments. For developers, this set-up is somewhat similar to having a team project in which you add solutions and projects. Once you've created an experiment, you can drag and drop components into it and start connecting them in order to create a Machine Learning model. An example of this is shown in Figure 1.

Heads up

Adding some components to a model is easy; the difficult part is finding out if your model can actually predict the correct things. Optimizing your model starts with cleaning the input data set. Missing values can lead to incorrect patterns.

Providing too much information can lead to poor performance. Fortunately, Azure ML provides a number of tools to analyze the accuracy of your model. You can even compare the performance of multiple algorithms. Interpreting the results does require some knowledge about statistics. A Data Scientist can help you clean your data sets and optimize your models.

To sum up

We've discussed a number of concepts found in Machine Learning. All examples were based on finding patterns in existing data, in order to predict something about the future. You can't predict your future without knowing your past. Machine Learning is no longer something only large companies use, it has become a commodity. It is quite simple to add some cleverness to your application using Azure ML. By using some trial and error you can quickly create some working predictive models. However, finetuning them still requires the help of an expert.

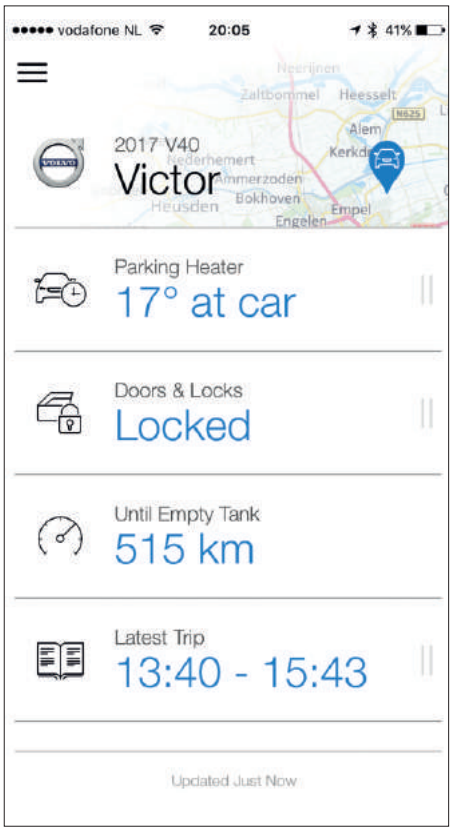
LOEK DUYS
CLOUD CONSULTANT XPIRIT

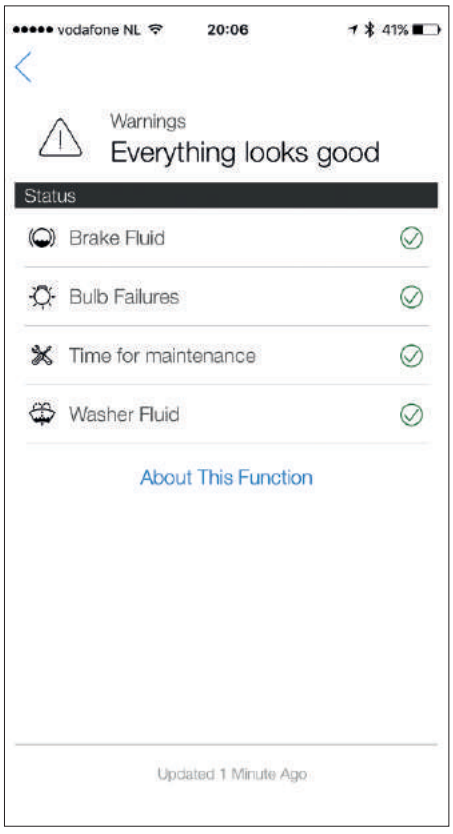
Loek is a Consultant and Cloud Solutions Architect who focuses on creating secure, scalable, available and maintainable systems. He's always looking for ways to leverage the latest additions to the Microsoft stack (Azure, .NET) to create better solutions. Loek likes to exchange technological knowledge, and to promote software security awareness.

How to accelerate your choices using data

A few months ago, I arrived at the Volvo dealer to pick up my new car. I was very excited. The car dealer tried to explain all of the car's features but I couldn't care less. All I wanted was to hit the road, but he insisted on installing an App on my phone before I took off. Installing this App would allow me 'to get to know my car better'. Five clicks later, all was set up, so finally I could hit the road with my new baby!

A couple of days later, I opened the app, and when I opened it, I was completely surprised by the amount of data it displays. It turns out that my car is no longer just a brutal engine. Instead, it's a mean IoT machine! It collects all sorts of data that give me insight into the usage of the car. It helps me to detect problems early and control a range of settings, and it even allows me to start the heater from my bed when it's freezing.
At first it was hard to understand how collecting data could impact me in a positive way. I became more curious when I realized that my navigation system is updated regularly, but I've never seen an update screen. When I asked the dealer, he explained that the navigation is updated only when my car is 'idle'. The car determines the ideal moment by analyzing the collected trip history.
He also explained that the car collects my street crossing approach speed. By detecting patterns in my behavior, Volvo gets a better understanding of how I use my car. They can now prevent me from having accidents by applying preventive maintenance on the brakes or by offering drive assisting features (brake assist, impact braking, lane corrections, etc.). Volvo has seen the light, similar to the likes of Tesla, Toyota and BMW. Data will allow them to run their business more effectively, while I get a safer car. What do you use to improve your business?





What can you get out of data?

Now, just collecting data will not help you with anything. Your storage facilities may contain petabytes of data without adding any value to your customer or your business. To deliver the desired impact, you will need to transform data intoinformation, but this is easier said than done. Before turning big data into something valuable, you'll need to understand what you can transform it into and how to achieve it.

Typically, when we gather data from a database, a developer will be very descriptive in his needs. He'll be explicit about the source and most likely he'll define precisely how it will be formatted. An example of this is the traditional poor men's reporting using a SQL statement:

```
SELECT Date, AVG(SpeedPerHour), AVG(TyreProfileReduction)
FROM CarUsageData
GROUP BY Date ORDER BY Date ASC
```

In this example we will combine the average speed per hour and the average tyre profile reduction, and group this by date. From a business perspective, we want to gain insight into yesterday's performance. The generally accepted name for this type of analytics is 'descriptive analytics', and the request is usually expressed in query languages.The second type of analytics I'd like to introduce you to, is very close to its 'descriptive' brother. It is called 'diagnostic analytics'.

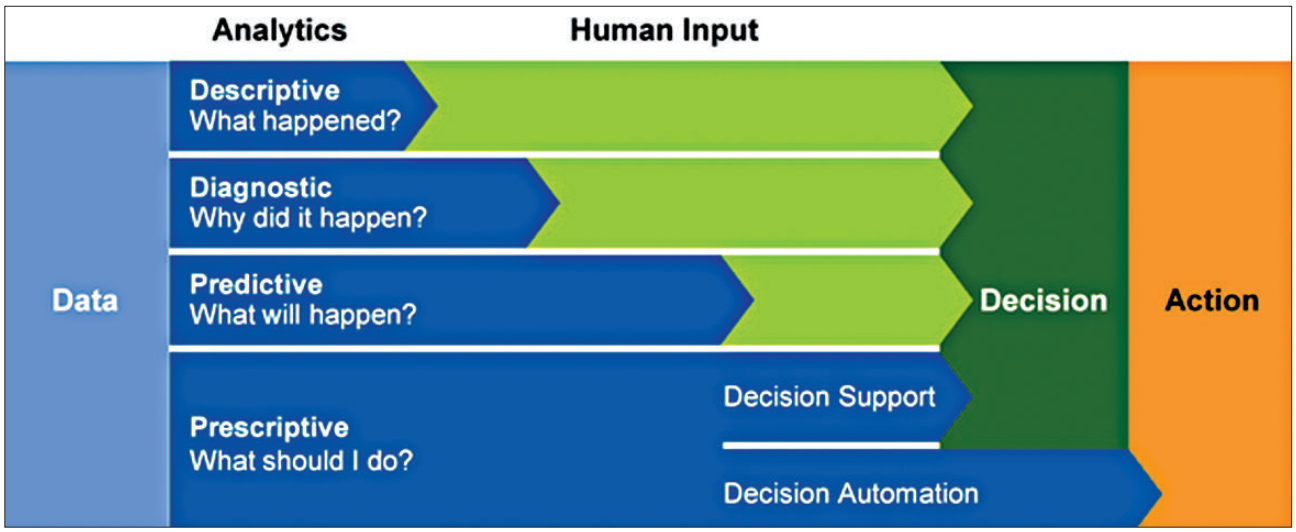
The focus of this type of analytics is still on the past, and the main difference lies in the underlying business question. In a descriptive world we ask ourselves the questions 'what happened', whereas in a diagnostic world we wonder about the question 'why did this happen'? So rather than searching for the occurrence of an incident, your question is now focused on identifying the (root-) cause of the incident.

Once you understand that your performance has decreased (descriptive) and you understand the cause of this(diagnostic), it's time to make the next step. Using the knowledge you gained, you want to be able to predict the future performance. This is the moment when you turn yourself to 'predictive analytics'. This kind of analytics helps you to predict future data points using models, and you train these models by feeding them historical and reference data.

Now that you understand what the future looks like, you'll probably want to understand how you can influence the result. This is when 'prescriptive analytics' can show you the way. An example: your sales director is confronted with low predictions for the next quarter. The predictive models have shown the urgency, and now he has to act accordingly. He can stimulate his market with a new product launch, but what is the appropriate size of his salesforce? And what is the best pricing strategy? Does he need to open a new shop on Bond Street in London or should he invest in better delivery options for customers outside of big cities? By simulating what will happen, the prescriptive models will advise what's the best possible combination of choices, and using this advice, the sales director can make a properly informed decision in order to gain the best result for his organization.

If it's a split second decision, prescriptive analytics can potentially even help to make the decision itself. The feedback gathered directly after the decision can feed directly back into the models. This allows the business to quickly understand whether a certain promotion text is more effective than another.

In addition, they don't have to worry whether the marketer is online at that particular moment to disable the less effective advertisement.



Gartner's way to visualize the different types of Big Data Analytics

[1] http://xpir.it/mag3-analytics1

Technical Limitations

For a very long period, technical challenges stood in the way of making proper analytics solutions happen. For the past 20 years, IT departments have been facing a financial wall, which disallowed them to handle large chunks of data in their infrastructure. Luckily, cloud-based large data stores, instant data processing power, and advanced access control are now very affordable.
Limitations at application level are also disappearing quickly. The three main cloud providers deliver very competitive solutions in the domain of machine learning. These solutions are updated constantly and offer a low entry level. Advanced web-based dash boarding techniques, such as PowerBI, speed up the time to market significantly, and they take away the visualization headache. These tools also reduce the time to market, which in turn allow business users to access their data a lot quicker. Algorithms and reference data are nowadays often publicly available. For commercial usage, several standard packages can be acquired from public marketplaces requiring limited investment.

Making the difference

Now that the technical barriers can be broken down by newly available capabilities, it's time to bring this advantage to your organization. Large analyst firms such as Gartner, predict[1] that between now and 2018 more than half of the large organizations will try to disrupt the industry by using advanced algorithms and analytics. They even expect that this growth will accelerate beyond this point. Knowing that these models become smarter while feeding them data, the urgency to start immediately becomes bigger every day. You can either allow your competitor a head start or you can jump on the boat and take a leading position.

It's important to understand that not every business question can be answered by using advanced analytics. Especially decision making involving your own employees is still considered a no-go; when there's Personable Identifiable Information in the process, it's recommended to take a very defensive approach with advanced analytics technologies. Typically, I believe that you can make the biggest difference in the following areas:

- Creating transparency in your organization about business results
- Enabling experimentation in customer needs & understanding variation in those needs·
- Segmenting customer bases to allow better and more targeted offers
- Replacing human decision making in order to reduce operational costs
- Innovating business models, products and services.

Making a head start

Your organization is not the first one to attack the problem. To avoid making the same mistakes and stand up on the shoulders of giants, we've listed the most important recommendations that will help you to keep gaining speed.

- When choosing a cloud vendor
  AWS, Google and Microsoft all have competing big data offerings. When choosing from one of them, realize that you'll marry this vendor. Even though it's technically possible, you'll never move 5 petabytes of data.
- Involve your legal department
  Your legal department can help you to understand what you can do with data and where you reach limits such as privacy and legalities. By involving them at an early stage, you can look at possibilities, rather than restrictions.
- Start storing raw data today
  Well-structured data in application databases is often optimized for a specific purpose. Since you don't know how you'll use this data in a model, don't process it. Instead, save the raw data.
- Obtain reference data
  It's going to be impossible to train your intelligent business models in a few months. You will need more data to train these models. Therefore, try to obtain relevant reference data on data marketplaces. This is both more effective and often cheaper than collecting it yourself.
- Train or hire talent?
  Data Science is a large part of computer science. It also requires a completely different mindset. To get started, hire the talent while you start building up the internal competence.
- Generate new input
  Consider everything as a new source of information. Consider tools such as Microsoft Cognitive Services to gain new types of data, even from human conversations.
- Not pretty
  Many times, the output of the numbers aren't that pretty. Rather than focusing on the visualization, allow self-service business intelligence using Excel or a dashboard. This allows you to focus on the quality of the data.

Conclusion

Getting started with big data analytics is never easy. Even though it became a lot more affordable, the investment in handling large pieces of data is still significant. In order to make sure that the investment is worth the money, you need to find an achievable, well-designed business objective, and you'll need to prove to your organization that this investment will yield returns[2]. In this game of figures, you must not overlook the possibility of adding customer value. Volvo has always committed itself publicly to passenger safety. With their investments in Big Data, they are proving very clearly that their value proposition is not just an advertisement, but that it's part of their product experience. And I hope it will keep on proving that, every day when I hit the road on my way home...

ALEX DE GROOT
CONSULTANT XPIRIT

Alex is focused on bringing simplicity and flow to software teams. He gets excited when talking about product thinking, feedback loops and cognitive load. Having a geeky interest in IoT, compilers and data structures, you'll find him exploring new technologies such as Cognitive Services and SmartDust.

[2] http://xpir.it/mag3-analytics2

DASA

DEVOPS AGILE
SKILLS ASSOCIATION

A UNIVERSAL QUALIFICATION PROGRAM FOR DEVOPS AND AGILE SKILLS DEVELOPMENT

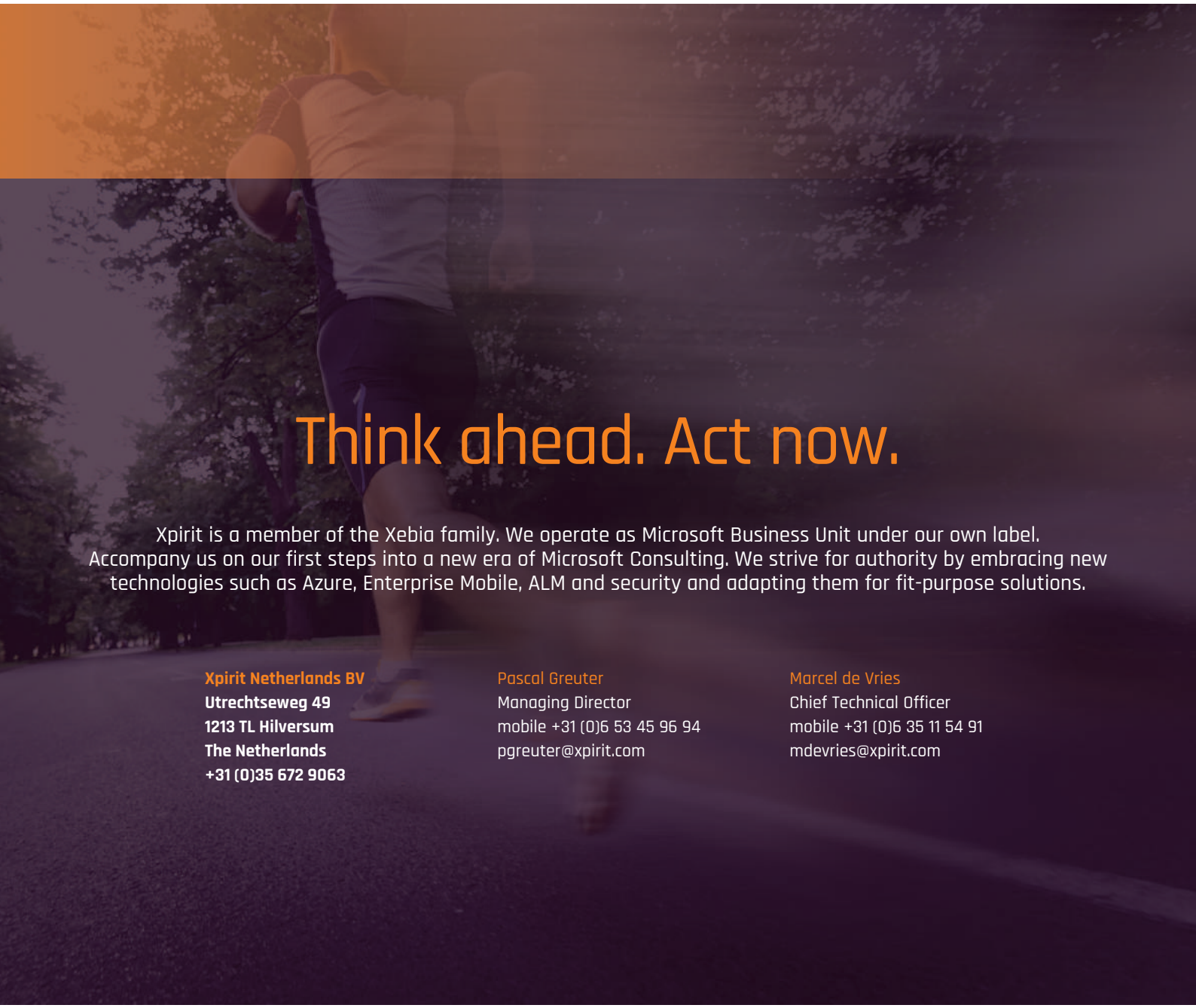
DASA - Empowering IT Transformational Change

DEVOPS AND AGILE GO MAINSTREAM

The DevOps Principles that guide us support the ultimate search for flow in the delivery of IT Services.

1. Customer-centric action (Courage to act, Innovate)
2. Create with the end in mind (Product & Service thinking, Engineering mindset, Collaborate)
3. End-to-End responsibility (Live your accountability, Concept to Grave, performance support)
4. Cross-functional autonomous teams (T-shaped profiles, complementary skills)
5. Continuous Improvement (If it hurts do it more often, experiment, fail fast)
6. Automate everything you can (Enhance quality, maximize flow)

www.devopsagileskills.org



Think ahead. Act now.

Xpirit is a member of the Xebia family. We operate as Microsoft Business Unit under our own label. Accompany us on our first steps into a new era of Microsoft Consulting. We strive for authority by embracing new technologies such as Azure, Enterprise Mobile, ALM and security and adapting them for fit-purpose solutions.

Xpirit Netherlands BV
Utrechtseweg 49
1213 TL Hilversum
The Netherlands
+31 (0)35 672 9063

Pascal Greuter
Managing Director
mobile +31 (0)6 53 45 96 94
pgreuter@xpirit.com

Marcel de Vries
Chief Technical Officer
mobile +31 (0)6 35 11 54 91
mdevries@xpirit.com

Xpirit

Think ahead. Act now.

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